Computer Physics Communications 263 (2021) 107849

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

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

ARTICLE INFO

Article history:
Received 28 April 2019
Received in revised form 23 December 2020
Accepted 5 January 2021
Available online 28 January 2021

Keywords:
Multiphase flows
Multifluid flows
Free-surface flows
Navier–Stokes equations
Front Tracking
Volume of Fluid
Surface tension

ABSTRACT

Paris (PArallel, Robust, Interface Simulator) is a finite volume code for simulations of immiscible multifluid 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 an explicit projection method with a first-order or second-order time integration scheme. The interface separating the different fluids is tracked by 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.

Program summary

Program Title: PArallel Robust Interface Simulator — Paris
CPC Library link to program files: https://doi.org/10.17632/5cb2yrfx7r.1
Licensing provisions: GPLv3.
Programming language: Fortran95/2002. Parallelized using MPI and domain decomposition.
Nature of problem: Paris is a free code, or software, for computational fluid dynamics (CFD) of multiphase flows (or computational multiphase fluid dynamics (CMFD)), specialized in the numerical simulation of interfacial fluid flows, involving droplets, bubbles and 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, including a surface tension force. 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 first-order or a second-order in time predictor–corrector method with an explicit projection step for the pressure. Spatial discretization is by finite volumes on a regular cuboid grid. Interface tracking is performed with the Front-Tracking (FT) method or the Volume-of-Fluid (VOF) method. In the VOF version Paris uses either the Lagrangian-Explicit (LE) advection method or the exactly mass-conserving method of Weymouth and Yue [2]. The normal computation is performed using the Mixed-Youngs-Centered (MYC) scheme. A mass–momentum advection method has been also implemented that is consistent with the VOF 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.

☆ The review of this paper was arranged by Prof. N.S. Scott.
☆☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: zaleski@dalembert.upmc.fr (S. Zaleski).

https://doi.org/10.1016/j.cpc.2021.107849

If the dynamics of a phase can be neglected, PARIS can also run as a free-surface code by specifying a homogeneous pressure, at most varying with time, in the neglected phase. In the case of atomizing jets, an algorithm has been implemented in PARIS that can detect isolated droplets, advects them as Lagrangian point-particles and possibly merge them again with the main stream
Additional comments: PARIS is extended from or inspired by the following codes:

- FTC3D: Front Tracking code for 3D simulations by Gretar Tryggvason and Sadegh Dabiri.
- SURFER: VOF code for 3D simulations by Stephane Zaleski, Jie Li, Ruben Scardovelli and others.
- GERRIS: multiphase flow solver with Adaptive Mesh Refinement (AMR) by Stephane Popinet.

References

[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] T. Arrufat, M. Crialesi-Esposito, D. Fuster, Y. Ling, L. Malan, S. Pal, R. Scardovelli, G. Tryggvason, S. Zaleski, A mass–momentum consistent, Volume-of-Fluid method for incompressible flow on staggered grids, *Computers & Fluids*, 215, 104785, 2021.

1. Introduction

Computations of the unsteady motion of multifluid 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 simulations of a validated system of equations that include non-trivial length and time scales.

A few authors of PARIS [3] have been involved in the development of *free codes* for DNS of two-phase flows, such as SURFER [4], GERRIS [5], dating back a couple of decades, and more recent ones such as BASILISK [6]. The new project PARIS is a joint effort with the aim to illustrate most of the methods described in [2] (the latter book will be denoted "TSZ" in what follows to simplify citations and we will often refer to "TSZ" for developments about the numerical methods) and it relies heavily on the previous codes FTC3D and SURFER. The software can run independently either with the Front-Tracking method or the VOF method. The mass–momentum consistent advection method, the free-surface option and the Lagrangian point-particles (LPP) algorithm have been specifically designed for PARIS and were not present in the previous codes.

Furthermore, a software platform such as Paris can make it possible to explore in the near future the development of combined models based on both Front-Tracking and Volume-of-Fluid methods.

While DNS of multiphase flows are becoming increasingly common, most research groups need to devote a considerable amount of time to code development. For new groups, the need to develop a suitable simulation tool can be a significant barrier to entry. PARIS is a free code that is intended to be relatively simple to use and modify even by beginner users, yet it has sufficient capabilities to allow state-of-the-art studies of typical multiphase flow problems.

2. Navier–Stokes equations with interfaces

2.1. Basic equations

In the three-dimensional space, the locus of the interface is a smooth surface S which separates the two fluid phases. Accordingly, we assume 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 literature, 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 a conservative form, expressing the momentum balance, or in a non conservative, Lagrangian 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 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 the stress tensor whose expression for incompressible flow is

$$\mathbf{D} = 2\,\mu\,\mathbf{S}\,, \quad \mathbf{S} = \tfrac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right), \tag{7}$$

and 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 surface tension coefficient, $\nabla_S\sigma$ its surface gradient, $\mathbf{n}$ the unit normal perpendicular to the interface, κ the sum of principal curvatures and δ_S a Dirac distribution concentrated on the interface. Other equations are required to specify the dependence of σ from other physical quantities, such as temperature, the presence of surfactants or electro-magnetic fields, therefore in the tests section we will assume a constant 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 $\mathcal{L}_2 = \mathcal{L}_{\text{adv}} + (\mathcal{L}_{\text{diff}} + \mathcal{L}_{\text{cap}} + \mathcal{L}_{\text{ext}})/\rho$. 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 have been defined above.

The conservation of momentum, on an elementary control volume moving with the interface S, leads to the following *jump conditions* across the interface

$$-\left[-p + 2\mu\,\mathbf{n}\cdot\mathbf{S}\cdot\mathbf{n}\right]_S = \sigma\kappa \tag{11}$$

and

$$-\left[2\mu\,\mathbf{t}^{(k)}\cdot\mathbf{S}\cdot\mathbf{n}\right]_S = \mathbf{t}^{(k)}\cdot\nabla_S\sigma \tag{12}$$

where $\mathbf{t}^{(k)}$, $k = 1, 2$, are two independent tangent vectors and the notation $[\ldots]_S$ denotes the jump of a physical quantity across the interface S. The Continuous Surface Force (CSF) [7] formulation smoothens the surface tension force a few cells across the interface and does not require the jump conditions (11), (12), that are instead implemented in the Ghost Fluid Method (GFM) [8].

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 that 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 (11), (12) become *boundary conditions* on the border of the liquid domain

$$(-p + 2\mu\,\mathbf{n}\cdot\mathbf{S}\cdot\mathbf{n})|_S = -p_{\text{air}} + \sigma\kappa \tag{13}$$

and

$$2\,\mu\;\mathbf{t}^{(k)}\cdot\mathbf{S}\cdot\mathbf{n}\big|_S = \mathbf{t}^{(k)}\cdot\nabla_S\sigma\,. \tag{14}$$

3. Numerical methods implemented in the code

3.1. Spatial discretization

We assume a regular cuboid grid, that 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 intersection

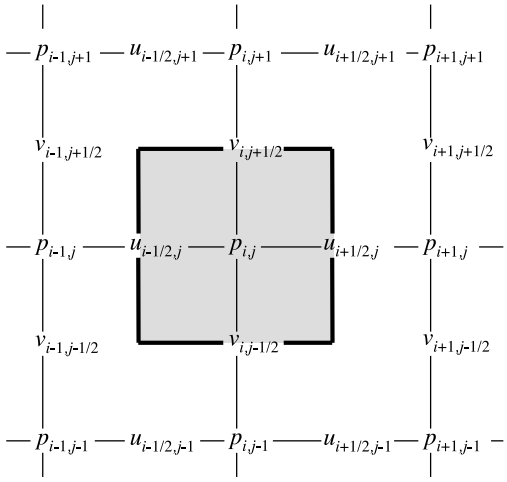

Fig. 1. Representation of the staggered spatial discretization. The pressure p is assumed to be known at the center of the control volume outlined by a thick solid line. The horizontal velocity component $u_1 = u$ is stored in the middle of the left and right edges of this control volume and the vertical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

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 Paris the Front Tracking method can indeed use stretched coordinates, while the routines implementing the VOF method are still limited to cubic grids.

We use a finite volume discretization of the momentum equation and consider staggered velocity and pressure grids. The staggered grid and control volumes for the pressure p are represented in Fig. 1. The corresponding control volumes of the velocity component u_m in direction m, $m = 1, 2, 3$, are shifted with respect to the control volume $\Omega_{i,j,k}$ surrounding the pressure p. The control volumes for the velocity components u_1 and u_2 in two dimensions, or for the corresponding momentum components, are shown on Fig. 2. 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 TSZ for a more detailed discussion.) This type of staggered representation is easily generalized to three dimensions.

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

$$\frac{u_{1;i+1/2,j,k} - u_{1;i-1/2,j,k}}{\Delta x} + \frac{u_{2;i,j+1/2,k} - u_{2;i,j-1/2,k}}{\Delta y} + \frac{u_{3;i,j,k+1/2} - u_{3;i,j,k-1/2}}{\Delta z} = 0. \tag{15}$$

In what follows, we shall use the notation $f = m\pm$, with the integer index $m = 1, 2, 3$, to denote the face of any control volume located in the positive or negative Cartesian direction m, and $\mathbf{n}_f$ for the normal vector to face f pointing outwards of the control volume. On a cubic grid the spatial step is $\Delta x = \Delta y = \Delta z = h$ and Eq. (15) becomes

$$\nabla^h\cdot\mathbf{u} = \sum_{m=1}^{3}(u_{m+} + u_{m-})/h = 0\,, \tag{16}$$

where $u_f = u_{m\pm} = \mathbf{u}\cdot\mathbf{n}_f$ is the velocity component normal to face f.

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, μ and C.

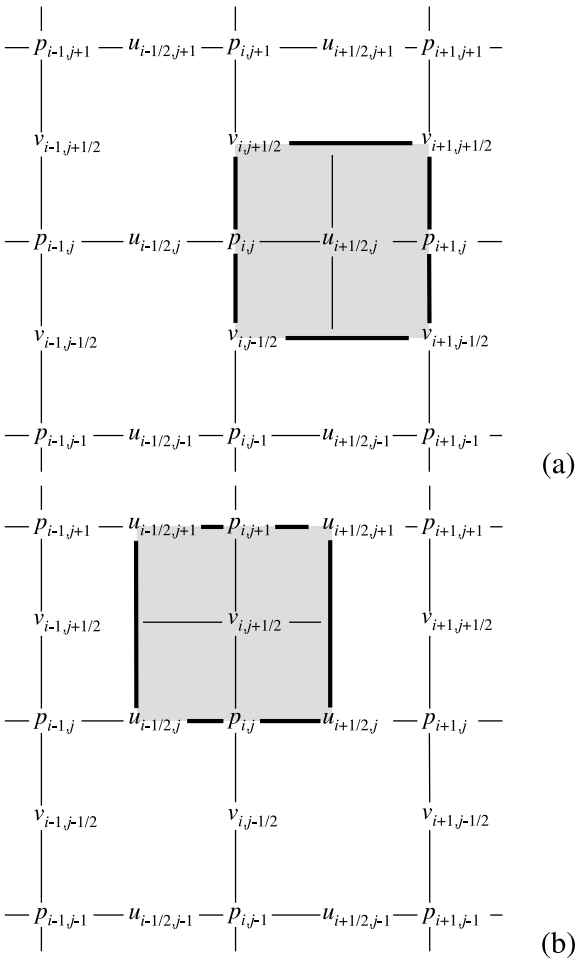

Fig. 2. The control volumes for the $u_1 = u$ and the $u_2 = v$ velocity components are displaced half a grid cell to the right (horizontal velocities) and to the top (vertical velocities). Here the indexes show the location of the stored quantities. Thus, half indexes indicate those variables stored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

3.2. Time marching

Time marching can be performed in a first-order or second-order manner using a small, possibly variable time step τ so that $t_{n+1} = t_n + \tau$. We start with the description of the first-order time stepping. The interface is first advanced in time as follows. In the Front-Tracking method, the front points $\mathbf{x}_k$ are moved as

$$\mathbf{x}_k^{n+1} = \mathbf{x}_k^n + \tau\, \mathbf{u}_i(\mathbf{x}_k^n, t_n) \tag{17}$$

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). A marker function I must then be constructed to compute the material properties in the grid points, as described in Section 3.3.2.

In the VOF method the volume fraction field is updated as

$$C^{n+1} = \mathcal{L}_{\mathrm{VOF}}(C^n, \mathbf{u}^n \tau / h)\,, \tag{18}$$

where $\mathcal{L}_{VOF}$ represents the operator that updates the C data given the velocity field $\mathbf{u}^n$ and is described in detail in Section 3.5. Once the volume fraction C (or the marker function I) is updated, the material properties are computed from (1), while the interface geometry, including its unit normal $\mathbf{n}$ and curvature κ, is estimated with the different methods described in the next sections. The velocity field is then updated with a predictor–corrector method. In the predictor step a provisional velocity field $\mathbf{u}^*$ is computed. Two different versions are used. One is the "non-consistent", non conservative version of Eqs. (9), (10)

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

the other is a "mass–momentum consistent", conservative version of Eqs. (5), (6)

$$\begin{aligned}\rho^{n+1}\mathbf{u}^* &= \rho^n \mathbf{u}^n + \tau \mathcal{L}^h_{\mathrm{cons}}(\rho^n, \mathbf{u}^n) \\ &\quad + \tau\left(\mathcal{L}^h_{\mathrm{diff}}(\mu^{n+1}, \mathbf{u}^n) + \mathcal{L}^h_{\mathrm{cap}}(C^{n+1}) + \mathcal{L}^h_{\mathrm{ext}}(C^{n+1})\right).\end{aligned} \tag{20}$$

Clearly the above operators depend on the discretization steps τ and h as well as the fluid properties. The "mass–momentum consistent" advection operator $\mathcal{L}^h_{\mathrm{cons}}$ is described in detail in [9]. In the second step the pressure gradient *corrects* the velocity

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\tau}{\rho^{n+1}} \nabla^h p\,. \tag{21}$$

This step 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

$$\nabla^h \cdot \mathbf{u}^{n+1} = 0\,, \tag{22}$$

which leads to an elliptic equation for the pressure

$$\nabla^h \cdot \frac{\tau}{\rho^{n+1}} \nabla^h p = \nabla^h \cdot \mathbf{u}^*\,. \tag{23}$$

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

$$(\mathbf{f}^{n+1}, \mathbf{u}^{n+1}) = \mathbf{L}(\mathbf{f}^n, \mathbf{u}^n) \tag{24}$$

where $\mathbf{f}^n$ is the interface data (either the front points $\mathbf{x}$ or the VOF fraction C) and the operator $\mathbf{L}$ consists in the steps described above and is applied to the data at time t_n. A second-order time scheme can be obtained by first computing a set of temporary variables $\mathbf{f}^{**}$ and $\mathbf{u}^{**}$

$$(\mathbf{f}^{**}, \mathbf{u}^{**}) = \mathbf{L}(\mathbf{f}^n, \mathbf{u}^n) \tag{25}$$

followed by the update of the variables at time t_{n+1} by the trapezoidal rule

$$(\mathbf{f}^{n+1}, \mathbf{u}^{n+1}) = \frac{1}{2}\left(\mathbf{L}(\mathbf{f}^{**}, \mathbf{u}^{**}) + \mathbf{L}(\mathbf{f}^n, \mathbf{u}^n)\right) \tag{26}$$

where the operator $\mathbf{L}$ on the right hand side effectively is applied to data at the intermediate time $t_{n+1/2}$. The PARIS code implements both the first-order and the second-order time schemes, controlled by the parameter ITIME_SCHEME.

3.2.1. Non-conservative momentum advection

The non-conservative momentum advection in Eqs. (9), (10) amounts to integrate over one time step the PDE

$$\partial_t u_m = \mathcal{L}_{\mathrm{adv},m}(\mathbf{u}) \tag{27}$$

where

$$\mathcal{L}_{\mathrm{adv},m} = -u_j \partial_j u_m \tag{28}$$

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

$$\partial_t \phi + \nabla \cdot (\phi \mathbf{u}) = 0 \tag{29}$$

Integrating the advection equation (29) in space, over a control volume centered on a node of the scalar ϕ, and in time one obtains

$$\phi^{n+1}_{i,j,k} - \phi^n_{i,j,k} = -\sum_{\text{faces f}} F_f^{(\phi)}. \tag{30}$$

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 the advecting velocity component u_f is obtained by simple averaging or centered interpolation. For example, we can consider the first component of velocity $\phi = u_1 = u$, whose control volume is centered at $i + 1/2, j, k$ (see Fig. 2). For the flux along the horizontal x direction, the right face is located at index $i + 1, j, k$. For the normal component of the velocity on this face, the interpolated value is $u_f = u_{1;i+1,j,k} = \frac{1}{2}(u_{1;i+1/2,j,k} + u_{1;i+3/2,j,k})$. On the other hand for the flux along the y direction, the top face is located at index $i + 1/2, j + 1/2, k$ and the advecting velocity component is now $u_f = u_{2;i+1/2,j+1/2,k} = \frac{1}{2}(u_{2;i,j+1/2,k} + u_{2;i+1,j+1/2,k})$.

Contrary to the estimates of u_f above, the estimation of the advected quantity 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, we assume that the variable ϕ, akin to the velocity component u_1, takes values $\phi_{i+1/2}$ at half-integer grid point indexes. We need to estimate the flux at integer index points x_i, thus we need to predict $\phi_i = \phi_f$. To this aim, an interpolation function is defined that computes this value as a function of the value of ϕ at the four nearest points, and in an upwind manner based on the sign of $u_i = u_f$, that is given by the previously-defined centered interpolation. The face value ϕ_i is then approximately given by

$$\phi_i = f(\phi_{i-3/2}, \phi_{i-1/2}, \phi_{i+1/2}, \phi_{i+3/2}, u_i) \tag{31}$$

where the function f is both of sufficiently high order and such as to limit 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 with the non conservative formulation (19), ϕ in Eq. (30) is taken equal to one of the velocity components u_m. This method is available whether one uses the VOF method or the Front-Tracking method. With the VOF method we have extensively used a combination of QUICK, away from the interface, and a first-order upwind near the interface, and a combination of Superbee slope limiter, away from the interface, and its modified version near the interface. These combinations have been found numerically more robust than others, and several simulation results are presented in [9].

With the Front Tracking method we use typically ENO or QUICK, as they offer a good combination of accuracy and stability.

3.2.2. Mass–momentum 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 both the VOF volume fraction C and the velocity $\mathbf{u}$. In other words, when there is a density jump on the interface, the discontinuity of the momentum density $\rho\mathbf{u}$ is advected exactly at the same velocity as the discontinuity of the mass density ρ. 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 when ρ is obtained from the VOF-advection of C using Eqs. (1) and (18) 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 non-consistent advection of the previous section was often unstable at large density ratios, while consistent methods are more stable [10–19].

The Paris code implements a modification of the classical momentum-preserving scheme proposed by [20] for the case of a staggered grid and Volume of Fluid (VOF) method. The scheme needs to be modified from the one in the previous section only near the interface. In particular, away from the interface, the density is constant and the scheme in (30) is already conservative. Near the interface, with $\phi = \rho u_m$, the momentum advection can be written, see [9] for details, as

$$\left(\rho u_m\right)^{n+1}_{i,j,k} - \left(\rho u_m\right)^{n}_{i,j,k} = -\sum_{\text{faces f}} u_{m,f}\, F_f^{(\rho)} + \sum_{l=1}^{3} \widetilde{u}_m\, C_l^{(\rho)}, \tag{32}$$

where $u_{m,f}$ is the one-dimensional interpolation of the velocity component u_m on face f and $F_f^{(\rho)}$ is the mass flux through the same face. The last sum represents the compressional term and it is related to the fact that each term in Eq. (16) can be different from zero even if the flow has zero divergence. In particular this sum is equal to zero for the mass-conserving VOF method of [21].

The one-dimensional interpolation schemes are different if they are applied away from or near the interface. The combinations that are used with the mass–momentum consistent advection have been already discussed in the previous section.

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. From (6), (7) we have

$$\mathcal{L}_{\text{diff,j}}(\mu, \mathbf{u}) = (\nabla\mu) \cdot \left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right) + \mu\nabla^2\mathbf{u}. \tag{33}$$

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

$$\mathbf{u}^* = \mathbf{u}^n + \tau\, \mu^{n+1}\nabla^2\mathbf{u}^*. \tag{34}$$

Then the discrete operator is defined as

$$\mathcal{L}^h_{\text{diff,j}}(\mu^{n+1}, \mathbf{u}^n) = (\nabla\mu^{n+1}) \cdot \left(\nabla\mathbf{u}^n + (\nabla\mathbf{u}^n)^T\right) + \left(\mathbf{u}^* - \mathbf{u}^n\right)/\tau \tag{35}$$

where $\mathbf{u}^*$ is the solution of the linear problem (34). The implicitation of the viscous terms is optional and controlled by a code parameter.

3.3. Interface advection: Front-tracking method

The interface advection in the "one-fluid" formulation is required to update the material properties, see Eq. (1), and the interface geometry in order to compute the surface tension forces. In the Paris code the interface can be moved either by Front Tracking (FT) or by the VOF method. We describe the former in this section. The FT method, 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

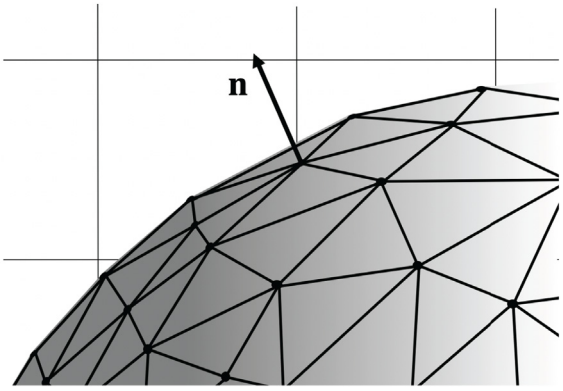

Fig. 3. An interface separating two immiscible fluids represented by marker points connected by triangular elements.

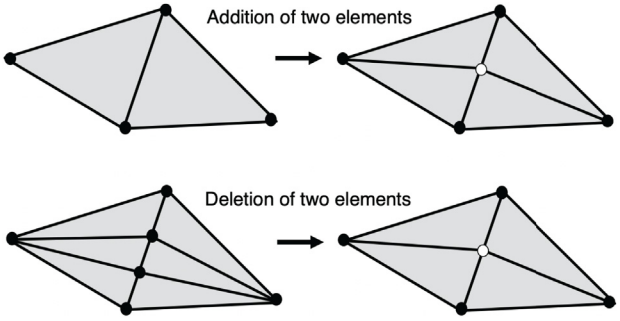

Fig. 4. Restructuring of a triangulated grid by adding and deleting points and elements.

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, as present in a simulation containing several bubbles and drops, can have different material properties, for example surface tension. Fig. 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 changes as points move away from, or closer to, each other. We keep the length of the edges of each element within about a quarter to a half of the spacings of the fixed 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 Fig. 4.

Topology changes in front tracking can be done in several slightly different ways, but in all cases additional code is required to: (a) detect points where coalescence/breakup should take place, and (b) restructure the front to account for the topology change. For a short description of a topology change algorithm and a few examples see [22] and for applications of a slightly modified version of the algorithm to more complex flows see [23] and [24]. A detailed description of a topology change algorithm is beyond the scope of the current manuscript.

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. The motion of the interface depends on the flow and the flow depends on where the interface is, therefore 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 on one side to interpolate data from the grid to the front and on the other side to spread, or "smooth", data from the front to the fixed grid. The first type of data transfer is required to move the front, where the velocity at the front points is interpolated from the velocity on the fixed grid. On a staggered grid each velocity component is interpolated separately. In general, we have

$$\phi_f^l = \sum_{ijk} w_{i,j,k}^l \, \phi_{i,j,k} \, . \tag{36}$$

Here, ϕ_f^l 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, $w_{i,j,k}^l$ 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 second type of data transfer is usually referred to as smoothing, since it replaces a quantity defined at a front point on the sharp interface, such as surface tension, 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

$$\phi_{i,j,k} = \sum_l \phi_f^l \, w_{i,j,k}^l \, \frac{\Delta A_l}{\Delta x \Delta y \Delta z}, \tag{37}$$

where ΔA 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 [25] which involves four grid points in each coordinate direction, or 64 points total. The weights are given by

$$w_{i,j,k}^l = d(r_x)\, d(r_y)\, d(r_z), \tag{38}$$

where r_x is the scaled distance (by the grid spacing) between x_f^l and the grid line located at x_i; r_y and r_z are defined in the same way. In our case, the weighting function $d(r)$ is given by

$$d(r) = \begin{cases} (1/4)(1+\cos(\pi r/2)), & |r| < 2, \\ 0, & |r| \geq 2. \end{cases} \tag{39}$$

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 I

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 I, 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. The grid value of the components of the gradient is given by Eq. (37) of the previous section

$$\left(\nabla I\right)_{i,j,k} = \sum_l \left(\Delta I\, \mathbf{n}\right)_l w^l_{i,j,k} \frac{\Delta A_l}{\Delta x \Delta y \Delta z}\,, \tag{40}$$

where ΔI is the jump in the value of the marker function across the interface (usually a given constant value) and $\mathbf{n}$ the unit normal to the element. Once the gradient field has been smoothed onto the fixed grid, we can integrate it to recover the marker function I. For example, on a staggered grid

$$I_{i,j,k} = I_{i-1,j,k} + \left(\frac{\partial I}{\partial x}\right)_{i-1/2,j,k} \Delta x\,. \tag{41}$$

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

$$\nabla^2 I = \nabla \cdot (\nabla I)_f\,, \tag{42}$$

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, as it is deduced from the position of the front through (41), this equation amounts to the Poisson equation

$$\nabla^2 I = \nabla \cdot \mathbf{G}_f \tag{43}$$

which must be solved to find I for a given $\mathbf{G}_f = \left(\nabla I\right)_f$. In the current version this equation is solved for the entire grid to keep the implementation simple, 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.

3.4. Surface tension: Front-Tracking method

In simulations of flows with sharp interfaces the front serves two main functions. The first is the advection of the marker function I, 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.

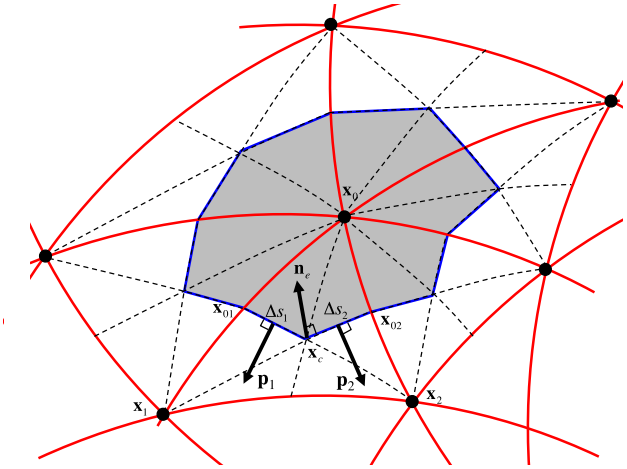

Fig. 5. Computation of the surface force on a triangulated grid by integrating over the edges of an element.

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 (37). 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

$$\mathbf{f}_\sigma = \int_{\Delta A} \sigma\, \kappa\, \mathbf{n}_e\, dA = \oint_C \sigma\, \mathbf{p}\, dl\,. \tag{44}$$

Here the Stokes theorem has been used to convert the area integral over ΔA into a line integral along its boundary C. The unit vector $\mathbf{p} = \mathbf{t} \times \mathbf{n}_e$ is tangent to the interface and perpendicular to the boundary of the interface element (see Fig. 5). 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 guaranteed to give the same result 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.

Fig. 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 parts by connecting the centroid $\mathbf{x}_c = (1/3)(\mathbf{x}_0+\mathbf{x}_1+\mathbf{x}_2)$ to the midpoints 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 midpoint rule

$$\Delta\mathbf{f}_\sigma \approx \sigma_1 \Delta s_1 \mathbf{p}_1 + \sigma_2 \Delta s_2 \mathbf{p}_2 \,. \tag{45}$$

Here, the pull on each element is the cross product of the outward unit normal, $\mathbf{n}_e$, and the tangent vector to the edge, i.e. $\Delta s_1 \mathbf{p}_1 = \mathbf{n}_e \times (\mathbf{x}_{01} - \mathbf{x}_c)$ 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 $\mathbf{n}_e$ is found by the normalized cross product of two of the tangent vectors to the edges of the element. After straightforward algebra, we have

$$\Delta\mathbf{f}_\sigma \approx \frac{\mathbf{n}_e}{3} \times \left(\sigma_1\big(\mathbf{x}_2 - \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_0)\big) - \sigma_2\big(\mathbf{x}_1 - \frac{1}{2}(\mathbf{x}_2 + \mathbf{x}_0)\big)\right) \tag{46}$$

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

$$\begin{aligned}\sigma_1 &= \frac{1}{2}\Big(\frac{1}{2}\big(\sigma(\mathbf{x}_0) + \sigma(\mathbf{x}_1)\big) + \frac{1}{3}\big(\sigma(\mathbf{x}_0) + \sigma(\mathbf{x}_1) + \sigma(\mathbf{x}_2)\big)\Big) \\ &= \frac{1}{12}\big(5\sigma(\mathbf{x}_0) + 5\sigma(\mathbf{x}_1) + 2\sigma(\mathbf{x}_2)\big)\end{aligned} \tag{47}$$

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 nodal point, that is then "smoothed" onto the fixed grid.

3.5. Interface advection: VOF method

When the interface location is captured by the VOF method, a variable $C_{i,j,k}$ is initialized. It is equal to the fraction of the cell $\Omega_{i,j,k}$ that is filled with the reference fluid 1. The implementation of the VOF method is limited for the time being to cubic cells of edge length h. Moreover, we use a rescaling of space and time variables so that the cell size and the time step are both 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.

3.5.1. Normal vector determination

The VOF method proceeds by a sequence of interface reconstructions and advections of C. In the reconstruction step, one attempts to recover the interface geometry from the VOF 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 solves the problem of finding a plane perpendicular to $\mathbf{n}$ which cuts the cube with 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) [26], described also in TSZ. However, when a quick determination of the normal is needed but not very accurate, the finite difference method $\mathbf{n} = \nabla^h C$ (the Youngs scheme [27]) can also be used.

3.5.2. Plane constant determination

Once the interface normal vector $\mathbf{n}$ is determined, a new, collinear 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

$$\mathbf{m} \cdot \mathbf{x} = \alpha \tag{48}$$

cuts exactly a volume V of the cube. 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 are using rescaled units of space, so that $0 \le V \le 1$ and $0 \le \alpha \le 1$. Then α is determined by the resolution of a cubic equation [28]. This resolution, and similar ones often used in the VOF method and derived in [28], are implemented in a kind of small library contained in the single file vof_functions.f90.

3.5.3. Volume initialization

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 either 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" initialization, the mesh cells are subdivided into n_l^3 subcells (where n_l 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 in each subcell is $c_l = \chi\big(\phi(\mathbf{x}_c)\big)$, where χ is the Heaviside function and $\mathbf{x}_c$ the subcell center. The cell value $C_{i,j,k}$ is then given by the sum of the c_l values divided by the subcells number n_l^3. In tests it was found that $n_l = 8$ was sufficient. However $n_l \ge 8$ 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 [29] and [30]. Then the initialization is performed using highly-accurate numerical integration to compute the fluid volumes in each grid cell, implicitly defined by the equation $\phi(\mathbf{x}) < 0$. The code is linked to the Vofi library with the shell variable HAVE_VOFI set before compilation.

3.5.4. General split-direction advection

The reconstruction at time t_n provides the approximate position of the interface, which is used to compute the reference phase fluxes across the cell boundary in order to update the volume fractions $C_{i,j,k}$ at time t_{n+1}. The Paris code contains two methods for split-direction advection, which can be selected by the user: Lagrangian Explicit (LE) advection, with the keyword VOF_ADVECT = LE, and 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), however 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 [31] and classified as the "LE" method in [32]. 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 [21] for WY. The reader may also refer to [9] for a condensed description of both methods.

An important operation in some simulations is the "clipping" procedure, and a small parameter ϵ_c is defined to this purpose. After advection, all cells that have $C_{i,j,k} < \epsilon_c$ are set to 0 and all cells that have $C_{i,j,k} > 1 - \epsilon_c$ are set to 1. This removes some, but not all, of the wisps, floatsam and jetsam that are generated. In the current version of the code the default value is $\epsilon_c = 10^{-8}$. This is a rather high value used mainly with WY in simulations at very high density contrast. The code has been observed to run well also with the smaller threshold value $\epsilon_c = 10^{-12}$ when the ratio of physical parameters is less extreme, or with no threshold at all with the LE advection. Other VOF schemes, and in particular unsplit schemes, are not reported to require clipping.

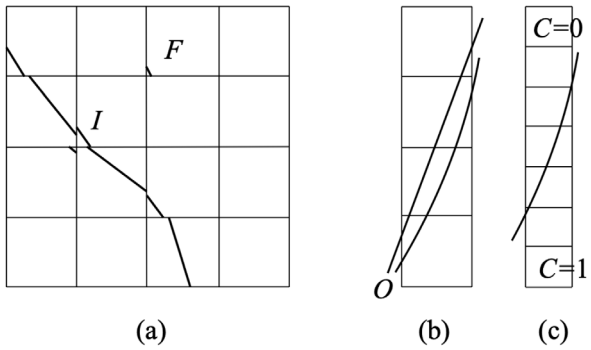

Fig. 6. (a) A small floatsam F in a cell away from the interface is negligible when height functions are used to determine the location of the interface. On the other hand requiring the interface to pass near each centroid C_i has a large effect. A small inconsistency such as near point I is also ignored by the height function. (b) Two cases where the height function expression (50) 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 the HF calculation one needs to have one full cell ($C = 1$) below and one empty cell ($C = 0$) above, or the converse.

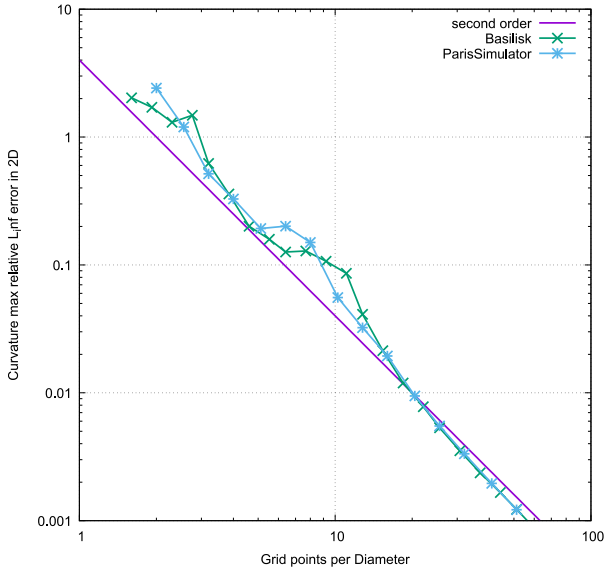

Fig. 7. Maximum L_∞ error norm in two dimensions for the curvature estimated for a cylinder using the height function method in Paris simulator and Basilisk. The mixed-height option is set in both codes.

3.6. Surface tension: VOF method

3.6.1. CSF method

For simplicity, we consider only the case where σ is constant. In the Continuous Surface Force (CSF) method [7] the force $\sigma\kappa\mathbf{n}\delta_S$ in the capillary term (8) is written as

$$-\sigma\kappa\nabla\chi = -\sigma\kappa^h\nabla^h C. \tag{49}$$

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 Ref. [33]).

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

3.6.2. Height functions

We give some details about height functions since it is a relatively novel aspect of the code. Height functions were introduced in [34] and further discussed, tested and improved in several papers [35,36]. A height function is a function on the discrete grid that gives the elevation of the interface with respect to a reference plane. 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 the interface in the given cell it would create a large error on the interface location (see Fig. 6a). With high resolution, height functions can also yield the position of the interface to fourth-order accuracy [35]. We consider for simplicity the case of an approximately horizontal interface, then the reference plane is aligned with the x, y plane. A local height H, rescaled by the grid size h, may be defined as

$$H = \sum_{p=k_0}^{p=k_0+n_c} C_{i,j,p}. \tag{50}$$

and is illustrated on Fig. 6b. The expression (50) defines a "vertical" height with the reference plane passing through point O and the base of the bottom cell on Fig. 6b. More general cases, with arbitrary orientation of the interface, are considered in Appendix.

The computation of the curvature may then be performed by reconstructing a polynomial approximation to the height function. To illustrate this, we take again the case of an approximately horizontal interface. Then we fit the local heights by the function

$$H(x, y) = \frac{a_1}{2}x^2 + \frac{a_2}{2}y^2 + a_3xy + a_4x + a_5y + a_6 \tag{51}$$

where the coefficients a_i are computed from the heights data using finite differences (see Eq. (A.3)). The curvature is then

$$\kappa = \epsilon\frac{a_1(1 + a_5^2) + a_2(1 + a_4^2) - 2a_3a_4a_5}{(1 + a_4^2 + a_5^2)^{3/2}} \tag{52}$$

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

The performance of our method for the computation of curvature is shown in Figs. 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 these two 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.

3.7. Pressure solver

3.7.1. In-code Gauss–Seidel solver

The default Poisson solver used to invert the elliptic operators appearing in Eqs. (23) and (34) is a red–black Gauss–Seidel (GS) solver with overrelaxation [37]. Both equations can be discretized in the form

$$\begin{aligned} &A_{1,i,j,k}\,p_{i-1,j,k} + A_{2,i,j,k}\,p_{i+1,j,k} + A_{3,i,j,k}\,p_{i,j-1,k} + \\ &A_{4,i,j,k}\,p_{i,j+1,k} + A_{5,i,j,k}\,p_{i,j,k-1} + A_{6,i,j,k}\,p_{i,j,k+1} - \\ &A_{7,i,j,k}\,p_{i,j,k} = A_{8,i,j,k}\,, \end{aligned} \tag{53}$$

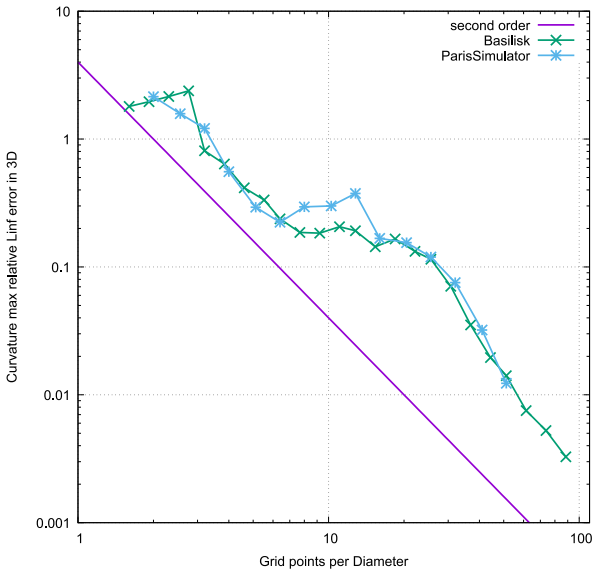

Fig. 8. Maximum L_∞ error norm in three dimensions for the curvature estimated for a sphere using the height function method in Paris and Basilisk. The mixed-height option is set in both codes.

where the coefficients verify the relations

$$A_{7,i,j,k} = \sum_{p=1}^{6} A_{p,i,j,k} \tag{54}$$

$$A_{2,i,j,k} = A_{1,i+1,j,k} \tag{55}$$

$$A_{4,i,j,k} = A_{3,i,j+1,k} \tag{56}$$

$$A_{6,i,j,k} = A_{5,i,j,k+1} \tag{57}$$

and are constructed by interpolations of $1/\rho$, for Eq. (23), or μ, for the implicit part (34) of the viscous term in the momentum equation. The GS solver iterates the assignment

$$\begin{aligned} p_{i,j,k} \leftarrow (1-\beta)\, p_{i,j,k} + \frac{\beta}{A_{7,i,j,k}} \Big(& A_{1,i,j,k}\, p_{i-1,j,k} \\ & + A_{2,i,j,k}\, p_{i+1,j,k} + A_{3,i,j,k}\, p_{i,j-1,k} + A_{4,i,j,k}\, p_{i,j+1,k} \\ & + A_{5,i,j,k}\, p_{i,j,k+1} + A_{6,i,j,k}\, p_{i,j,k-1} - A_{8,i,j,k} \Big) \end{aligned} \tag{58}$$

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

3.7.2. Library multigrid solver

The Hypre library, developed at 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, a few solvers of 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. In fact, the scaling performance of the PFMG solver is much better than the SMG solver, since the smoothing operations only involve 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, however if the iteration diverges or fails to converge within the maximum iteration number, 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 then 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_∞.

3.7.3. 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 coordinate direction. In particular the V-Cycle scheme is implemented and fully parallelized [37]. Relaxation operations are applied starting first from the finest to the coarsest grid, and then from the coarsest to the finest one, the number of relaxation operations 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 installed in the system, such as Hypre. 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.

3.7.4. 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 Eq. (58). 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 pressure matrix p, it also enables the leanest data transfer between CPUs and GPUs, which is a critical aspect of normal CUDA applications. A Gauss–Seidel red–black solver is in principle possible and could be 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 (58)
end for
for all $\Omega_{i,j,k}$ cells of "black" type **do**
  compute $p_{i,j,k}$ using (58)
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 this 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 coefficients matrix A and the pressure matrix 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.

3.7.5. Free-surface flow solver

A free-surface flow solver is implemented in Paris, which is designed to apply a free-surface condition as described in Section 2.3. The implementation of the boundary conditions (13), (14) is here limited to inviscid flows, hence only the jump condition (13) along the normal direction is considered. The free-surface solver uses the VOF method implemented in Paris to track the interface. The flow is then solved only in the phase with $\chi = 0$, using the same numerical methods described previously. For the purpose of description the solved phase will be called liquid and the unsolved phase will be gas. In the region occupied by the gas phase the conservation equations are not solved, instead a homogeneous pressure field is assumed, that can vary only in time. The time variation of the pressure is determined by a polytropic gas law

$$p_c = p_0 \left(\frac{V_0}{V_c} \right)^{\gamma} , \tag{59}$$

where V_c is the total volume of the gas pocket at pressure p_c, p_0 and V_0 are respectively the reference pressure and volume of the gas phase, and finally 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 Eq. (13).

The method used to apply this pressure boundary condition is inspired by the idea of Fedkiw and Kang [8,38], often referred to as the Ghost Fluid Method (GFM). Let the time-varying pressure in the unsolved phase be p_c. Special care is required in the discretization of the elliptic equation (23) 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 (we recall that 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 gray area represents a gas-filled volume. Cells that contain a filled circle are included in the pressure solution, while cells without a marker in the center 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 neighboring cell falls inside the gas phase. The pressure in this cell must be substituted by a surface pressure p_s. We apply a finite-difference gradient approach as Chan [39]. As an example, the approximation for the pressure gradient components for the cell with indexes i and j in Fig. 10 is written

$$\nabla_x^h p_{i+1/2,j} = \frac{p_{s,i+1,j} - p_{i,j}}{\delta_{i+1/2,j}} \; ; \quad \nabla_y^h p_{i,j-1/2} = \frac{p_{i,j} - p_{s,i,j-1}}{\delta_{i,j-1/2}} , \tag{60}$$

where δ is the distance between the pressure node under consideration and the intersection with the interface. The surface pressure p_s on the liquid side of the interface is found by adding to the gas pressure p_c the Laplace pressure jump. The pressure p_c inside each cavity is known from the polytropic law (59). The interface pressure p_s in the first expression of (60) will then be

$$p_{s,i+1,j} = p_{c,i+1,j} + \sigma \frac{\kappa_{i,j} + \kappa_{i+1,j}}{2} . \tag{61}$$

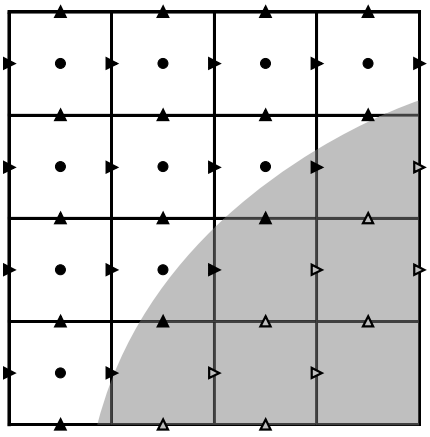

Fig. 9. A 2D section of the numerical grid, showing part of a gas bubble in gray. Circles represent computational cell nodes where pressure is calculated. Triangles indicate scalar velocity components on the computational cell faces. Filled triangles indicate values which are found by solving the governing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

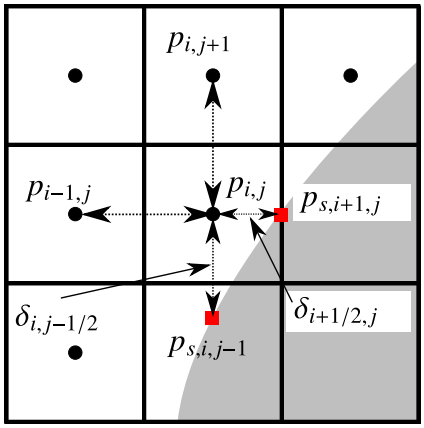

Fig. 10. Discretization of the pressure equation near the interface.

From Eqs. (60) and (61) it is clear that an accurate computation of the 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 boundary 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 neighbors 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 neighbors in the gas phase, we extrapolate the resolved velocities similarly to Popinet [40].

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 neighbors using a linear least-square fit. Let us assume that the velocity field can be described as a linear combination

$$\mathbf{u}(\mathbf{x}) = \mathbf{A} \cdot (\mathbf{x} - \mathbf{x}_0) + \mathbf{u}_0 \tag{62}$$

where the components of the tensor $\mathbf{A}$ and of the vector $\mathbf{u}_0$ are the unknowns. In two dimensions we take a 5×5 stencil around

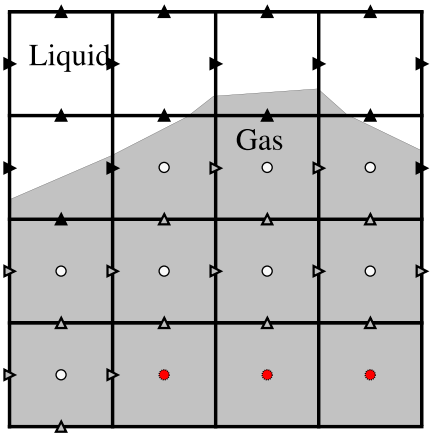

Fig. 11. 2D example of the problem to correct the extrapolated velocities (unfilled triangles). A Poisson's problem is solved in the cells marked with an unfilled circle.

the unknown gas velocity at location $\mathbf{x}_0$ and find the extrapolated velocity $\mathbf{u}_0$ by minimizing the functional

$$\mathcal{L} = \sum_{k=1}^{N} \left| \mathbf{A} \cdot (\mathbf{x} - \mathbf{x}_0) + \mathbf{u}_0 - \mathbf{u}_k \right|^2 \tag{63}$$

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

Ensuring volume conservation. 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 [41] 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" pressure is obtained in these cells by solving a Poisson's equation

$$\nabla^h \cdot \left(\nabla^h \hat{P} \right) = \nabla^h \cdot \tilde{\mathbf{u}} \,, \tag{64}$$

where $\hat{P}$ is the "phantom" pressure and $\tilde{\mathbf{u}}$ is the extrapolated velocity on the faces of the first two gas neighbors. $\hat{P}$ is only calculated in the cells represented by unfilled circles 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, a fixed pressure is prescribed in the cells (red filled circles) outside the region where the "phantom" pressure is computed. Only the extrapolated velocities (unfilled triangles) are then corrected by the solved pressure gradient $\nabla \hat{P}$

$$\tilde{\mathbf{u}}^{n+1} = \tilde{\mathbf{u}} - \nabla^h \hat{P} \tag{65}$$

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

For more details on the numerical method and its application in idealized *microspalling*, see [42] and [43]. This process is found in metals that under shock loading melt and where, with the reflection of the shock wave from the material free surface, cavities can nucleate, grow and merge.

3.8. 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 $i + 1/2, j, k$ for the horizontal velocity component u). The following pseudo code is executed at initialization relatively to the x direction, with $i = 1, 2, \ldots, n_x$

for all j, k **do**
  $s^x_{1/2,j,k} \leftarrow s_{1,j,k}$
end for
for all i, j, k **do**
  $s^x_{i+1/2,j,k} \leftarrow s_{i,j,k}$
end for

and similarly for the y and z directions. 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**
  $u_{i+1/2,j,k} \leftarrow s^x_{i+1/2,j,k}\, u_{i+1/2,j,k}$
end for

and similarly for the velocity components v and w. The pressure correction should not change the velocity on the solid boundaries, 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**
  $A_{1,i,j,k} \leftarrow s^x_{i-1/2,j,k} A_{1,i,j,k}$
  $A_{2,i,j,k} \leftarrow s^x_{i+1/2,j,k} A_{2,i,j,k}$
  (similarly 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 (23) 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. To avoid this type of problem, there is a small utility program "`rockread.c`", distributed with the code, that checks that the fluid "percolates".

3.9. Lagrangian point-particle model

A Lagrangian point-particle (LPP) model has been implemented in PARIS, that is fully coupled to the VOF method to provide a multi-scale modeling strategy to simulate liquid atomization: the large-scale interfaces on the bulk liquid are resolved by VOF, while the small droplets are represented by the LPP model. The details of the model can be found in [44] and only a brief summary is given here.

3.9.1. Overview

The combined VOF-LPP algorithm is shown in Fig. 12. The Navier–Stokes equations and the advection equation for the volume fraction C are solved for the resolved flow. Then the C field is tagged to identify different liquid structures. The tagging approach of [45] is used, and cells that are attached to each other, with respect to the liquid phase, will have the same tag number. The size, aspect ratio, and distance from the bulk flow of each liquid structure are computed in the tagging process. Small resolved droplets (RD) that satisfy the LPP criteria will be submitted to the RD-to-LPP conversion routine. The motion equation is then solved for each LPP droplet to update its velocity and position.

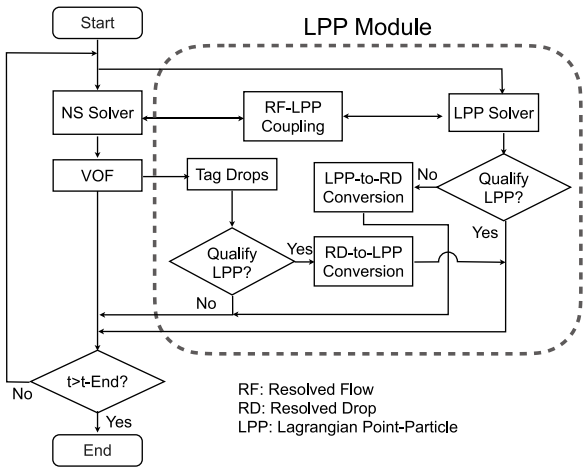

Fig. 12. Flow chart of the combined VOF-LPP algorithm.

Afterwards, the droplet is examined to decide whether or not it should be converted back to a resolved droplet.

LPP droplets and the resolved flow (RF) are two-way coupled. The RF properties, such as fluid velocity, are needed to calculate the forces acting on each LPP droplet. On the other hand, these forces need to be subtracted from the momentum equation of the flow, thus appearing as an additional source term.

3.9.2. Two-way coupling between LPP and RF

The non-conservative momentum equation (9), (10) is now written as

$$\rho\left(\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \nabla\cdot\mathbf{D} + \sigma\,\kappa\,\delta_S\,\mathbf{n} - \mathbf{f}_p \tag{66}$$

where $\mathbf{f}_p$ is the closure term that accounts for the backward effect of the LPP droplets on the resolved flow. The LPP model approximates each small droplet as a point mass, hence the droplet-scale flow is not resolved. To accurately track the LPP droplets in the Lagrangian framework, the force exerted on each droplet is calculated in terms of the undisturbed flow field properties. The closure is typically given by the force model or the so-called equation of motion (EOM) [46]

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}_p\,, \tag{67}$$

$$\frac{d\mathbf{u}_p}{dt} = \frac{\mathbf{u}-\mathbf{u}_p}{\tau_p}\phi + \frac{\rho}{\rho_p}\frac{D\mathbf{u}}{Dt} + \frac{C_m\rho}{\rho_p}\left(\frac{D\mathbf{u}}{Dt} - \frac{d\mathbf{u}_p}{dt}\right) + \mathbf{g}\,, \tag{68}$$

where ρ and $\mathbf{u}$ are the density and velocity of the undisturbed flow, and $\mathbf{x}_p$, $\mathbf{u}_p$, $\tau_p = \rho_p\, d_p^2/(18\,\mu)$, ρ_p, and C_m the position, velocity, response time, density, and added-mass coefficient of the LPP droplet. A tri-linear interpolation is used to interpolate the fluid velocity from the grid to the position of the LPP droplets. The modified gravity acceleration is denoted by $\mathbf{g} = (1-\rho/\rho_p)\mathbf{g}'$, where $\mathbf{g}'$ is the gravity acceleration. For the time integration of Eqs. (67) and (68) a second-order predictor–corrector method is used, which is consistent with the algorithms used for the resolved flow.

The force terms on the right hand side of Eq. (68) represent the quasi-steady force, the pressure-gradient force, the added-mass force, and the gravity force, respectively. The Basset-history force has been ignored for simplicity, while the Faxén and lift forces have been neglected because the LPP model is here applied to small droplets, and the effect of inhomogeneous ambient flows on the droplet force is expected to be small. The quasi-steady force is expressed as the Stokes drag multiplied by a correction term ϕ, which is a function of the Reynolds number $\mathrm{Re}_p = \rho\, d_p\, |\mathbf{u}-\mathbf{u}_p|/\mu$ [47],

$$\phi(\mathrm{Re}_p) = 1 + 0.15\,\mathrm{Re}_p^{0.687} + 0.0175\,\mathrm{Re}_p\left(1 + \frac{42500}{\mathrm{Re}_p^{1.16}}\right)^{-1}. \tag{69}$$

In atomization, the inter-droplet interaction can be ignored, as the small droplets are quickly dispersed away from the bulk liquid and the related volume fraction is relatively small.

As a consequence of Newton's third law, the force $\mathbf{f}_p$ exerted on the LPP droplets needs to be subtracted from the resolved flow and is referred to as backward coupling. This force in the momentum equation (66) is calculated as

$$\mathbf{f}_p = \sum_{i=1}^{N_p} \mathbf{F}_{fp,i}\, G(\mathbf{x}-\mathbf{x}_{p,i})\,, \tag{70}$$

where N_p is the total number of LPP droplets. The Gaussian function $G(\mathbf{x}-\mathbf{x}_{p,i})$ is a numerical representation of the LPP droplet coupling force [48]

$$G(\mathbf{x}) = (2\pi\,\mathcal{L}^2)^{-3/2}\exp(-|\mathbf{x}|^2/2\mathcal{L}^2)\,, \tag{71}$$

where $\mathcal{L}$ controls the size of the region where the force should be distributed. Note that only the force due to fluid-LPP-droplet interaction needs to be subtracted from the momentum equation

$$\mathbf{F}_{fp,i} = m_{p,i}\left(\frac{d\mathbf{u}_{p,i}}{dt} - \mathbf{g}\right). \tag{72}$$

3.9.3. Two-way conversion between LPP and resolved droplets

The droplets that are generated in the atomization process are converted to LPP droplets when they satisfy the following criteria.

LPP criteria. The criteria to determine whether a resolved droplet (RD) should be represented as a LPP droplet are based on its volume V_p, aspect ratio γ_p, and position $\mathbf{x}_p$. V_p is required to be smaller than a critical value $V_{crit} \simeq (4\,h)^3$, and γ_p close to one. Furthermore, since the current LPP model does not include either droplet formation or droplet-interface interaction, only droplets that are at a given distance away from the liquid jet interface will be converted. The distance is typically chosen to be the droplet diameter d_p. The overall conversion criteria can then be expressed as

$$\{\text{LPP Qualified}\} = \{V_p < V_{crit}\}\ \&\&\ \{|\gamma_p - 1| < \epsilon_\gamma\}\ \&\&\ \{\mathbf{x}_p \in \mathcal{R}_{ai}\}\,, \tag{73}$$

where ϵ_γ is the tolerance for the aspect ratio, and $\mathcal{R}_{ai}$ is the region away from the interface.

RD-to-LPP conversion. If a RD satisfies the LPP criteria, the volume fraction C in the corresponding cells is set to zero. A new LPP droplet is created and added to the LPP array, together with its volume and velocity. Furthermore, the flow field in the same region will be replaced by the undisturbed flow, so that the LPP droplet sees the proper undisturbed velocity field. This is done by interpolating each component of the fluid velocity along the corresponding coordinate from the surface into the interior of a cubic region centered at the droplet location. The reconstructed velocity field is globally divergence-free if the velocity on the surface of the interpolation region is divergence-free. Unless the particle Reynolds number is very small, a cubic region with an edge size twice the droplet diameter is sufficient to reconstruct the undisturbed flow field in a satisfactory way.

LPP-to-RD conversion. When a LPP droplet does not satisfy the LPP criteria anymore, for example when it is too close to the liquid-jet interface, it will be converted back to a resolved droplet. First, a spherical droplet is rebuilt around $\mathbf{x}_p$, by specifying the volume fraction C in the cells that will be occupied by the resolved droplet. The velocity field needs to be updated in the same cells. To account for the momentum of both the droplet and the virtual fluid moving with the droplet, the velocity in the cells occupied by the resolved droplet should be changed to $\mathbf{u}_p'$, which is calculated as

$$\mathbf{u}_p' - \mathbf{u} = \left(1 + \alpha \frac{\rho}{\rho_p}\right)(\mathbf{u}_p - \mathbf{u}), \tag{74}$$

where α is the ratio between the volumes of the virtual fluid and the droplet. For the added-mass effect, it is considered that the ratio between the volumes of the virtual fluid to be accelerated through the inviscid mechanism and the particle is the added-mass coefficient C_M, which is equal to 0.5 for a sphere in incompressible flows. Due to finite viscous diffusion time scale, the history force usually needs to be expressed in integral form. Nevertheless, it has been shown in [49] that, if the particle and ambient fluid acceleration time scales are much larger than the viscous diffusion time scale, the history force can also be expressed as a non-integral form like the added-mass force, and a viscous-unsteady coefficient C_{vu} similar to the added-mass coefficient C_M can be derived as

$$C_{vu} \approx 8.51 \left(\frac{0.75 + 0.105\mathrm{Re}_p}{\mathrm{Re}_p}\right). \tag{75}$$

The viscous-unsteady coefficient C_{vu}, can be considered as the ratio between the volumes of the virtual fluid to be accelerated through the viscous mechanism and the particle. The excess momentum added through $\mathbf{u}_p'$ is to mimic the effects of the added-mass and history forces on accelerating the surrounding fluid around the droplet. Therefore, α can be estimated as the sum of C_M and C_{vu} and thus depends on the droplet Reynolds number as well.

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 min on a laptop with an i7 processor and the long version takes approximately 25 min. 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 "mass–momentum consistent" option for velocity advection, has been tested extensively in [9] although several test cases of the method are included in the test suite and will be described below. In order to avoid the introduction of more equations we assume a constant surface tension coefficient σ in all the testing section.

In many of the tests the solution computed during the test run is compared to a reference solution. In some cases the reference solution was computed by the authors at a different resolution and is included with the code distribution. In other cases the "reference" is a near-identical numerical solution performed by the authors and included in the code distribution. In that case the reference should be identical to the solution except in some cases where tiny changes in the code or the implementation create moderately large differences. This is the case for the Raindrop test below.

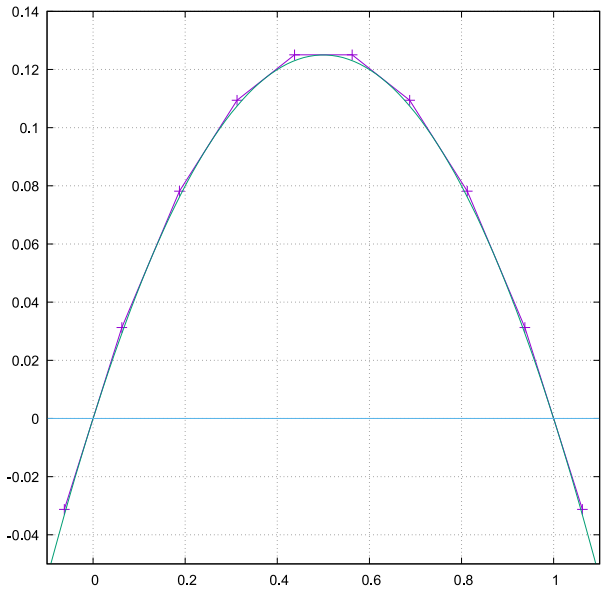

Fig. 13. Poiseuille flow test case.

4.1. Elementary tests

4.1.1. Poiseuille flow

An elementary Poiseuille flow [50] 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 set pressure on the left and right. The simulation is continued until the flow becomes stationary, 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 Fig. 13. 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 parabola amplitude is set by the quality of the approximation of the $u = 0$ boundary condition (solid wall). Since this condition is set at first order, there is a small $\mathcal{O}(h^2)$ 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 $u^{\mathrm{ghost}}_{i,-1/2} + u_{i,1/2} = 2u^{\mathrm{wall}}$. The boundary condition is thus implemented by writing in a "ghost layer" of the grid

$$u^{\mathrm{ghost}}_{i,-1/2} \leftarrow 2u^{\mathrm{wall}} - u_{i,1/2}. \tag{76}$$

The result is shown on Fig. 13. It is also possible to run the Poiseuille flow test in other manners, for example with set inflow velocity, or to run it with the implicit version of viscous diffusion, in which case the flow converges in a few time steps.

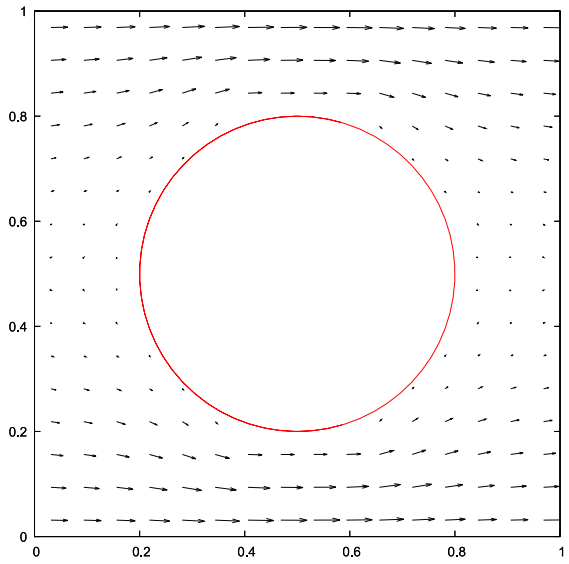

Fig. 14. Stokes flow around a disk test case.

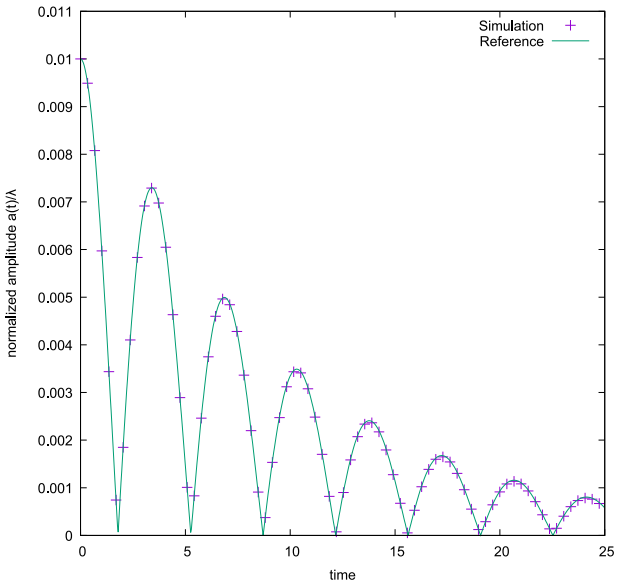

Fig. 15. Comparison of the temporal evolution of the normalized amplitude of the capillary wave obtained numerically and of the analytical initial-value solution by [51].

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 Fig. 14. There are two other similar tests beyond this first one. In the second one, located in the directory Disk, the flow is driven by a body force akin to gravity 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 volume fraction field C 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 undeformed droplet moving across the domain.

4.1.4. Cylinder advection

This is a more sophisticated test that probes the "mass-momentum consistent" option. The test is described in detail in [9]. 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. On a 2015 MacBook, a single processor run delivered a speed of $1.6\ 10^6$ cells per second. On an AMD EPYC 7351, a single processor run delivered a speed of $1.9\ 10^6$ cells per second and a parallel eight-processor run delivers a speed of $1.65\ 10^6$ cells per second and per processor (The parallel test is optional and can be run by the user by typing sh run-speed-test.sh N where $np = N^3$ is the number of processors desired). A significant drop of the code speed from this value would be a serious issue.

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 [34]. Results for Basilisk, have been obtained by the authors from the code published on the website http://basilisk.fr.

$N = \lambda/h$	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	$4.1\ 10^{-4}$

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 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 [51,52] for small-amplitude capillary waves in viscous fluids. In the AIVS the aspect ratio L_y/L_x should be sufficiently large (at least a value of 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 dependence of the remaining error on h. Fig. 15 compares the temporal evolution of the amplitude of the interface perturbation with the AIVS.

The relative L_2 error norm of the difference between the numerical solution of a few numerical codes and the AIVS has also been computed. The results, depicted in Table 1 and Fig. 16, 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 [34], a 0.005 amplitude had to be used to match the AIVS.

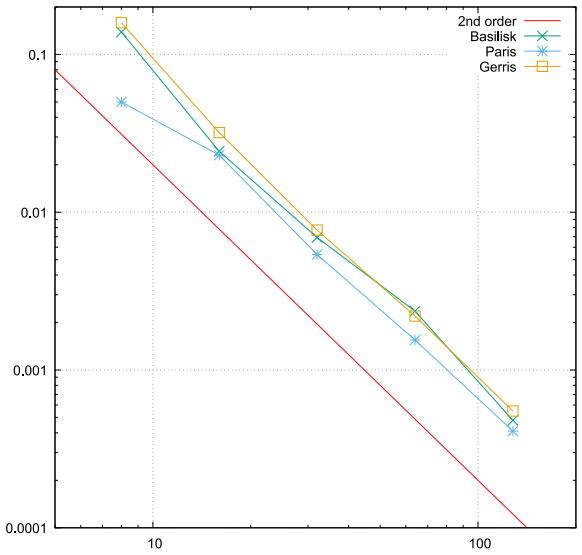

Fig. 16. Relative L_2 error of the numerical solution as a function of the number of grid points per wavelength $N = \lambda/h$ for the capillary wave test.

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

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

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

r	m	La
ρ_l/ρ_g	μ_l/μ_g	$\sigma\rho_l d/\mu_l^2$
833.3	58.82	218 400

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 Fig. 17. This test is not designed to assess the accuracy of the code, since the implemented numerical methods have already been used and tested elsewhere (see for example [53] for an oscillating droplet test with the VOF method and [54,55] for similar tests with Front Tracking. However, it should be noted that in 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 to assess whether air–water properties, which are often creating stability problems, are in fact in the stable regime of the code.

Fig. 18 shows the amplitude oscillations as a function of time for a droplet of $D/h = 19.2$ grid points per diameter and an initial excentricity of 0.75. The test is run without the mass–momentum consistent option, which is here seen to be unnecessary for the stability, and with the mixed-height option discussed in 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.

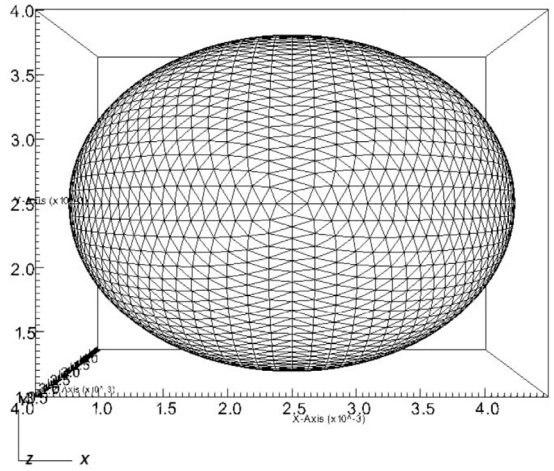

Fig. 17. Initial ellipsoidal shape of the droplet or bubble with the Front.

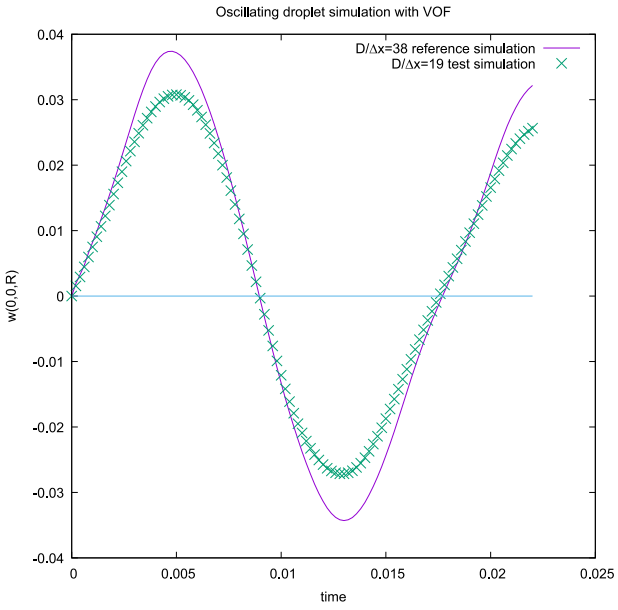

Fig. 18. Amplitude of capillary oscillations of the Droplet test case.

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 code by simulating the same droplet test. Results are shown on Fig. 19. The reference solution is once again the VOF simulation at larger resolution ($D/h = 38.4$). Satisfying agreement is found in this test as well.

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.

Fig. 20 shows for the Bubble test case the kinetic energy oscillations as a function of 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 code by simulating the same bubble test. Results are shown on Fig. 21. The

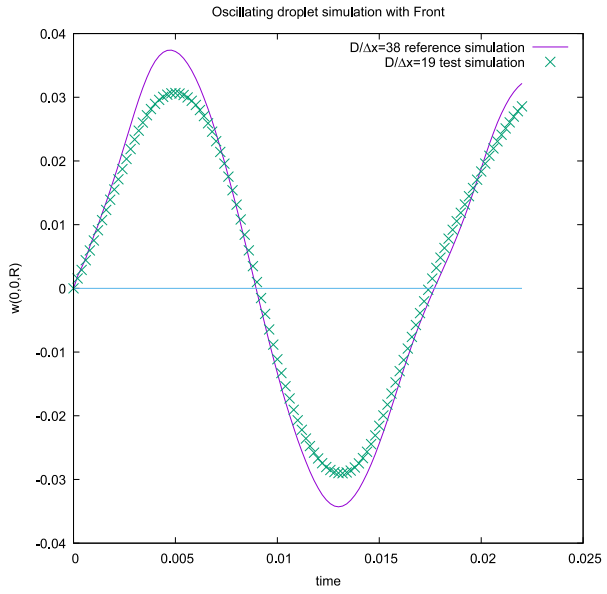

Fig. 19. Amplitude of capillary oscillations of the FrontDroplet test case.

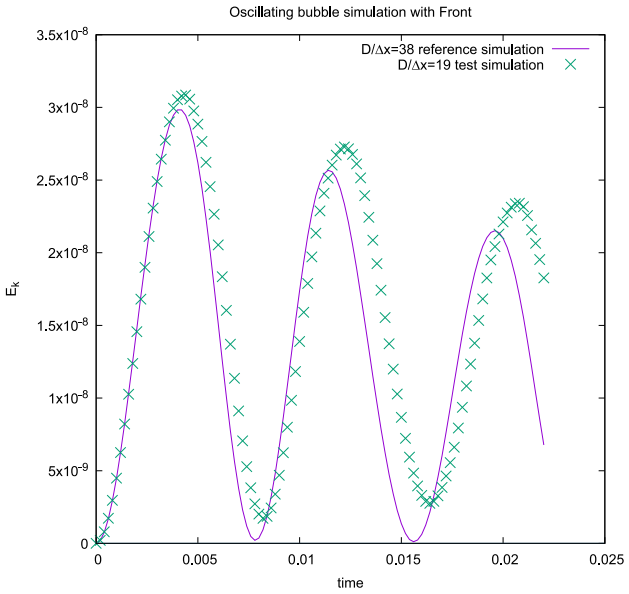

Fig. 21. Kinetic energy of capillary oscillations of the FrontBubble test case.

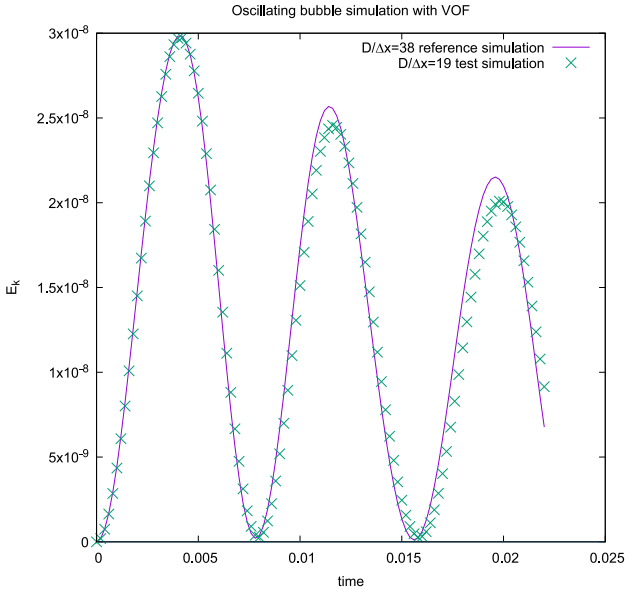

Fig. 20. Kinetic energy of capillary oscillations of the Bubble test case.

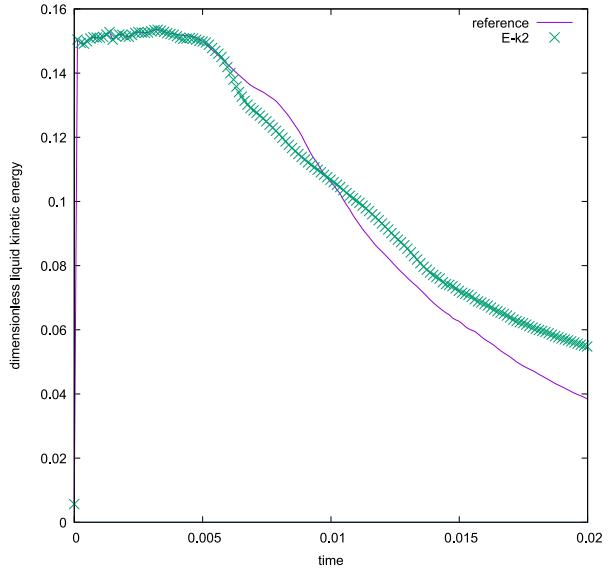

Fig. 22. Kinetic energy of a 3 mm falling raindrop at low resolution $D/\Delta x = 8$.

reference solution is again that of the previous case (VOF simulation with 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 $d = 3$ mm and allowed to fall in air under gravity. The droplet should remain approximately spherical, with at most a pancake or bun-like shape, but for many numerical codes this case is rather difficult and spurious atomization of the droplet is seen. For an example with the Basilisk code see [56]. For details about the setup of the test we refer the reader to [9]. The test is performed with the parameters of Table 4. Notice that these parameters differ slightly from those of [9] as we input here the values of the physical parameters for air and water at temperature $T = 20$ °C. The grid resolution is low $D/\Delta x = 8$, since the test is more demanding (i.e., it leads to a blowup of the code more easily) at low resolutions. Fig. 22 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, as explained in the introduction to this Section 4, since the flow is in a regime that is very sensitive to physical parameters and initial conditions and any small change in the code will create a deviation of that sort. The shape of the droplet is shown in Fig. 23.

5. Installation and usage

5.1. External libraries

When solving elliptic equations, we may apply the Hypre package SMG [57], a semi-coarsening multigrid solver with 3D

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

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

Fig. 23. Simulated shape of a 3 mm falling raindrop at low resolution $D/\Delta x = 8$.

plane smoothing on structured, cuboid meshes, as mentioned in Section 3.7.2.

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 [42]. In case of Hypre-related problems, fallback to version 2.10 is recommended for debugging.

The Vofi library [29,30] may be used to initialize the volume fraction field C in 2D and in 3D (see Section 3.5.3). 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 [42,43], and more rarely when initialization requires a considerable amount of computational time compared to Paris simulation time (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`.

Since Hypre and Vofi are indeed 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 they are not set the compilation proceeds without the libraries. The fallback for Hypre is the built-in Gauss–Seidel solver followed by the in-code multigrid solver, and the fallback for 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 at 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 many 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 point-particle module parameters (optional, implicitly requires two-phase flow, see [44]).
- inputvof (optional, VOF 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. 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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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, USA Grants CBET-1335913 and NSF DMS-1620158, by the ANR MODEMI project, France (ANR-11-MONU-0011) program and by grant SU-17-R-PER-26-MULTIBRANCH from Sorbonne Université, France. 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 at the Lawrence Livermore National Laboratory.

Appendix. Details of curvature computation from height functions (HF)

A.1. Height computation

The height computation is performed as described in Section 3.6.2. A more general definition than (50) 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 $\mathbf{e}_a$, $1 \leq a \leq 3$, where $\mathbf{e}_a$ is one of the Cartesian base vectors aligned 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 "liquid" $C = 1$ 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 one-dimensional symmetric "stencil" or "stack" S centered on $\mathbf{x}_{i,j,k}$ and oriented parallel to $\mathbf{e}_a$ and for the "positive" orientation ϵ. $L_{i,j,k} = \epsilon(\mathbf{x}_{i,j,k} - \mathbf{x}_0) \cdot \mathbf{e}_a$ is the distance from the base of the stack to the cell center $\mathbf{x}_{i,j,k}$. When the orientation is $\epsilon = -1$

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

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 Fig. 6(c). This height function can be computed whenever both the bottom cell and the top cell of the stack do not contain the interface, and the interface crosses only once the intermediate cells. This can be tested by requiring that there are cells with $C = 0$ and $C = 1$ at the top and bottom of the stack (see again Fig. 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 45° case. Thus, considering a cell crossed by the interface, one should explore one cell above and 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 three. However, even with 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 and 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 an accidental feature or our implementation 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 cells are exchanged between MPI processes, the sum in (50) is broken in two parts, one in each processor. Then the processes exchange two pieces of information, the "partial sums" just computed and the lengths $L_{i,j,k}$ in expression (A.1), allowing them to reconstruct the full sum.

A.2. First pass, first attempt: fully-aligned heights

In the first pass, a loop is performed over all cells $\Omega_{i,j,k}$ cut by the interface, hence having $0 < C_{i,j,k} < 1$. In the first pass two attempts are made. Let the current cell be Ω_0 with grid coordinates i_0, j_0, k_0. In the first attempt, the normal $\mathbf{n}$ is estimated by MYCS [26]. Then the grid direction $\mathbf{e}_a$ closest to the normal is determined (maximum of $|\mathbf{n} \cdot \mathbf{e}_a|$). Without loss of generality we consider the case $a = 3$ and the horizontal plane perpendicular to $\mathbf{e}_3$, that is the grid plane most closely aligned with the interface. A 3×3 planar block of cells Σ_0, aligned with this plane, is selected containing the nine cells Ω_{i,j,k_0}, with $i_0 - 1 \leq i \leq i_0 + 1$ and $j_0 - 1 \leq j \leq j_0 + 1$. 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 following coefficients of the polynomial (51) can be found using finite differences

$$\begin{aligned}
a_1 &= \partial^2_{xx} H \simeq H_{1,0} - 2H_{0,0} + H_{-1,0}\,,\\
a_2 &= \partial^2_{yy} H \simeq H_{0,1} - 2H_{0,0} + H_{0,-1}\,,\\
a_3 &= \partial^2_{xy} H \simeq (H_{1,1} - H_{-1,1} - H_{1,-1} + H_{-1,-1})/4\,,\\
a_4 &= \partial_x H \simeq (H_{1,0} - H_{-1,0})/2\,,\\
a_5 &= \partial_y H \simeq (H_{0,1} - H_{0,-1})/2\,.
\end{aligned} \tag{A.3}$$

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 for example 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 hard-coded 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 be fitted. In some cases, different directions $\mathbf{e}_a$ could yield two close points, say $\mathbf{x}^a_{i,j,k}$ and $\mathbf{x}^b_{l,m,n}$, in the same cell or in a neighboring cell. In this case, 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. Before the fitting is performed, an approximate normal is computed using the MYCS approach

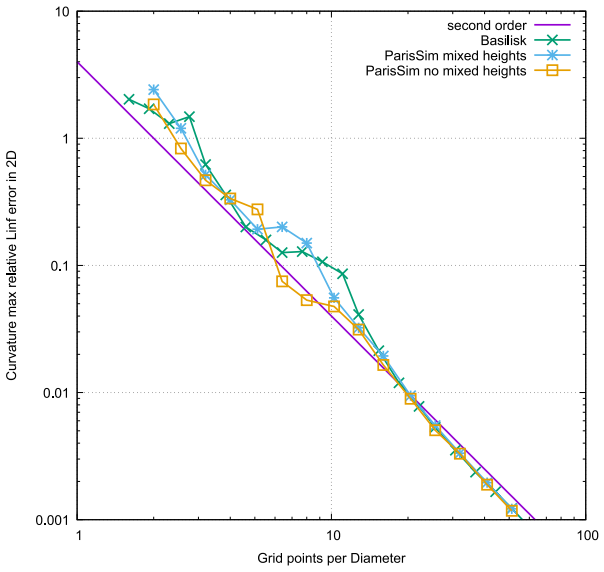

Fig. A.24. Maximum L_∞ error norm in two dimensions for the curvature estimated for a cylinder using the height function method in Paris and Basilisk. Two Paris results are shown, one with the mixed curvature option and one without. Averaging is used in both cases. Using the mixed-cell option yields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

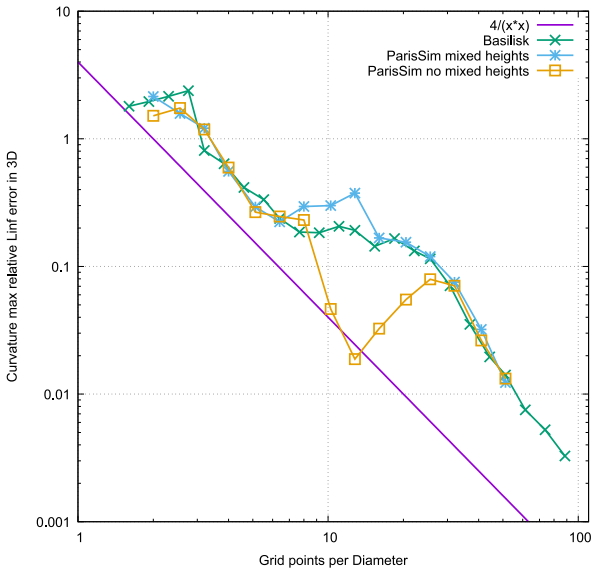

Fig. A.25. Maximum L_∞ error norm in three dimensions for the curvature estimated for a sphere using the height function method in Paris and Basilisk. Two Paris results are shown, one with the mixed curvature option and one without. Averaging is used in both cases. Using the mixed-cell option yields less accurate results than not using it.

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 have found that performing rotation has 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 mass–momentum consistent scheme is not used. The results in Fig. 7 are with MIXED_HEIGHTS='F' . With MIXED_HEIGHTS='T' one obtains the results of Fig. A.24. 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.

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 successful by either method in the first pass. For each cell $\Omega_{i,j,k}$, the cubic set of neighbors

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

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

A.5. Second pass: centroid fit

A final loop on cells $\Omega_{i,j,k}$ is performed. If all previous approaches have failed or are not set to be used, then one falls back to a fitting of centroids. In each cell of the cubic set $B_{i,j,k}$ containing an interface, the cell centroid is computed using the vof_functions microlibrary included in the code (implementing the method in [28]). 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 parameters in expression (51).

In some cases, the fit fails because the least-square linear operator is not invertible. The code then continues operating, increments a counter for statistical purposes and flags the cell as having an uncomputable curvature. A zero surface tension force is then added to the momentum.

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 Fig. A.25. 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.

References

[1] R. Scardovelli, S. Zaleski, 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] 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, 2012-2015.
[4] B. Lafaurie, C. Nardone, R. Scardovelli, S. Zaleski, G. Zanetti, J. Comput. Phys. 113 (1994) 134–147.
[5] S. Popinet, The gerris flow solver, 2001-2014, URL http://gfs.sf.net.
[6] S. Popinet, Basilisk, a Free-Software program for the solution of partial differential equations on adaptive cartesian meshes, 2018, URL http://basilisk.fr.
[7] J. Brackbill, D.B. Kothe, C. Zemach, J. Comput. Phys. 100 (1992) 335–354.
[8] R.P. Fedkiw, T. Aslam, B. Merriman, S. Osher, J. Comput. Phys. 152 (2) (1999) 457–492.
[9] T. Arrufat, M. Crialesi-Esposito, D. Fuster, Y. Ling, L. Malan, S. Pal, R. Scardovelli, G. Tryggvason, S. Zaleski, Comput. & Fluids 215 (2021) 104785.
[10] M. Bussmann, D.B. Kothe, J.M. Sicilian, ASME 2002 Joint US-european Fluids Engineering Division Conference, American Society of Mechanical Engineers, 2002, pp. 707–713.
[11] O. Desjardins, V. Moureau, Cent. Turbul. Res. Summer Program. 2010 (2010) 313–322.

[12] M. Raessi, H. Pitsch, Comput. & Fluids 63 (2012) 70–81.
[13] V. Le Chenadec, H. Pitsch, J. Comput. Phys. 249 (2013) 185–203.
[14] S. Ghods, M. Herrmann, Phys. Scr. 2013 (T155) (2013) 014050.
[15] G. Vaudor, T. Menard, W. Aniszewski, M. Doring, A. Berlemont, Comput. & Fluids 152 (2017) 204–216.
[16] J.K. Patel, G. Natarajan, J. Comput. Phys. 350 (2017) 207–236.
[17] N. Nangia, B.E. Griffith, N.A. Patankar, A.P.S. Bhalla, J. Comput. Phys. 390 (2019) 548–594.
[18] C.B. Ivey, P. Moin, J. Comput. Phys. 350 (2017) 387–419, http://dx.doi.org/10.1016/j.jcp.2017.08.054.
[19] M. Owkes, O. Desjardins, J. Comput. Phys. 332 (2017) 21–46, http://dx.doi.org/10.1016/j.jcp.2016.11.046.
[20] M. Rudman, Internat. J. Numer. Methods Fluids 28 (1998) 357–378.
[21] G.D. Weymouth, D.K.P. Yue, J. Comput. Phys. 229 (8) (2010) 2853–2865.
[22] M. Razizadeh, S. Mortazavi, H. Shahin, Acta Mech. 229 (2018) 1021–1043.
[23] J. Lu, G. Tryggvason, Phys. Rev. Fluids 3 (2018) 084401, (20 pages).
[24] J. Lu, G. Tryggvason, Phys. Rev. Fluids 4 (2019) 084301, http://dx.doi.org/10.1103/PhysRevFluids.4.084301.
[25] D.M. McQueen, C.S. Peskin, J. Comput. Phys. 82 (1989) 289–297.
[26] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, J. Comput. Phys. 225 (2007) 2301–2319.
[27] D.L. Youngs, An Interface Tracking Method for a 3D Eulerian Hydrodynamics Code, Technical Report 44/92/35, AWRE, 1984.
[28] R. Scardovelli, S. Zaleski, J. Comput. Phys. 164 (2000) 228–237.
[29] S. Bnà, S. Manservisi, R. Scardovelli, P. Yecko, S. Zaleski, Comput. & Fluids 113 (2015) 42–52.
[30] S. Bnà, S. Manservisi, R. Scardovelli, P. Yecko, S. Zaleski, Comput. Phys. Comm. 200 (2016) 291–299.
[31] J. Li, C. R. Acad. Sci., Paris II 320 (1995) 391–396.
[32] R. Scardovelli, S. Zaleski, Internat. J. Numer. Methods Fluids 41 (2003) 251–274.
[33] S. Popinet, Annu. Rev. Fluid Mech. 50 (2018) 49–75.
[34] S. Popinet, J. Comput. Phys. 228 (2009) 5838–5866.
[35] G. Bornia, A. Cervone, S. Manservisi, R. Scardovelli, S. Zaleski, J. Comput. Phys. 230 (2011) 851–862.
[36] M. Owkes, O. Desjardins, J. Comput. Phys. 281 (2015) 285–300, http://dx.doi.org/10.1016/j.jcp.2014.10.036, URL http://www.sciencedirect.com/science/article/pii/S0021999114007189.
[37] W.L. Briggs, A Multigrid Tutorial, SIAM Philadelphia, 1987.
[38] M. Kang, R.P. Fedkiw, X.-D. Liu, J. Sci. Comput. 15 (3) (2000) 323–360.
[39] R.K.C. Chan, R.L. Street, J. Comput. Phys. 6 (1970) 68–94.
[40] S. Popinet, S. Zaleski, J. Fluid Mech. 464 (2002) 137–163.
[41] M. Sussman, J. Comput. Phys. 187 (2003) 110–136.
[42] W. Aniszewski, S. Zaleski, A. Llor, L. Malan, Numerical simulations of pore isolation and competition in idealized micro-spall process, International Journal of Multiphase FlowDOI: https://doi.org/10.1016/j.ijmultiphaseflow.2018.10.013. URL http://www.sciencedirect.com/science/article/pii/S0301932218303082.
[43] L. Malan, Y. Ling, R. Scardovelli, A. Llor, S. Zaleski, Comput. & Fluids 189 (2019) 60–72, URL arXiv:1711.04561 [physics.flu-dyn].
[44] Y. Ling, S. Zaleski, R. Scardovelli, Int. J. Multiph. Flow. 76 (2015) 122–143, http://dx.doi.org/10.1016/j.ijmultiphaseflow.2015.07.002, URL http://www.sciencedirect.com/science/article/pii/S0301932215001524.
[45] M. Herrmann, J. Comput. Phys. 229 (3) (2010) 745–759.
[46] C.T. Crowe, M. Sommerfield, Y. Tsuji, Multiphase Flows with Droplets and Particles, CRC Press, 1998.
[47] R. Clift, W.H. Gauvin, Proc. Chemeca 1 (1970) 14–28.
[48] M.R. Maxey, B.K. Patel, E.J. Chang, L.P. Wang, Fluid Dyn. Res. 20 (1997) 143–156.
[49] Y. Ling, M. Parmar, S. Balachandar, Int. J. Multiph. Flow. 57 (2013) 102–114.
[50] P. Kundu, I. Cohen, D. Dowling, Fluid Mechanics, 891 pp, Elsevier, Amsterdam, 2012.
[51] A. Prosperetti, Phys. Fluids 24 (1981) 1217–1223.
[52] F. Denner, G. Paré, S. Zaleski, Eur. Phys. J. Spec. Top. 226 (6) (2017) 1229–1238.
[53] D. Fuster, G. Agbaglah, C. Josserand, S. Popinet, S. Zaleski, Fluid Dyn. Res. 41 (6) (2009) 065001.
[54] D.J. Torres, J.U. Brackbill, J. Comput. Phys. 165 (2) (2000) 620–644.
[55] U. Olgac, D. Izbassarov, M. Muradoglu, Comput. & Fluids 77 (C) (2013) 152–158.
[56] C. Pairetti, S. Popinet, S.M. Damián, N. Nigro, S. Zaleski, Bag mode breakup simulations of a single liquid droplet, 6th European Conference on Computational Mechanics (ECCM 6) and 7th European Conference on Computational Fluid Dynamics (ECFD 7) 1115 June 2018, Glasgow, UK.
[57] P. Sloot, C. Tan, J. Dongara, A. Hoekstra (Eds.), Computational Science - ICCS, Springer-Verlag, ICSS 2002, Berlin, 2002.