OpenFOAM® Journal

Volume 3 [Full Papers], Pages 200–224
ISSN: 2753-8168

TwoPhaseFlow: A Framework for Developing Two Phase Flow Solvers in OpenFOAM

Henning Scheufler[1] and Johan Roenby[2]

[1]DLR German Aerospace Center, Institute of Space Systems, 28359 Bremen, Germany
Email address: henning.scheufler@web.de

[2]IMFUFA, Department of Science and Environment, Roskilde University, Roskilde, Denmark
Email address: johan@ruc.dk

DOI: https://doi.org/10.51560/ofj.v3.80
Results with version(s): OpenFOAM® v2206
Repository: https://github.com/DLR-RY/twophaseflow

Abstract. We present a new OpenFOAM based open-source framework, TwoPhaseFlow, enabling fast implementation and testing of new phase change and surface tension force models for two-phase flows including interfacial heat and mass transfer. Capitalizing on the runtime-selection mechanism in OpenFOAM, the new models can easily be selected and benchmarked against analytical solutions and existing models. The framework currently includes the following three interface curvature calculation methods for surface tension: 1) the height function method, 2) the parabolic fit method and 3) the reconstructed distance function method. As for phase change, two models are available: 1) Interface heat resistance and 2) direct heat flux. These can be combined in three solvers: 1) InterFlow for isothermal, incompressible two-phase flow, 2) compressibleInterFlow for compressible, non-isothermal two-phase flow and 3) multiRegionPhaseChangeFlow for compressible, non-isothermal two-phase flow with conjugated heat transfer. By design, addition of new models and solvers is straightforward and users are encouraged to contribute their specific models, solvers, and validation cases to the library.

1. Introduction

In almost all engineering design processes involving fluid flows, simulations with Computational Fluid Dynamics (CFD) software play an ever bigger role as a complement, and sometimes even as a replacement, to empirical laws and physical experiments. A computationally very challenging subset of these problems is those involving multiple fluid phases with heat and mass transfer between them. Here models and numerical methods are still relatively immature, and so the use of CFD for design optimisation is still limited within this arena. In this paper, we present a numerical modeling framework based on OpenFOAM that simplifies implementation and verification and thus enables faster development of more accurate models.

The formulation of the heat and mass transfer or surface tension force models is highly influenced by the numerical representation of the fluid interface. With interface tracking methods the interface is represented directly as the mesh faces separating the two fluid regions. This makes implementation of the surface tension and mass transfer across the interface relatively straightforward. The main drawback of this approach is the difficulty in dealing with large topological changes of the fluid interface. With interface capturing methods, like Volume of Fluid (VOF) and Level-Set (LS), topology changes are handled automatically. However, the implementation of phase change and surface tension models is more challenging and depend on the numerical interface representation. With VOF methods the fluid interface is represented by a volume fraction field, also sometimes called the colour function. The volume fraction of a computational cell is simply the fraction of the cell's volume occupied by the reference fluid (typically chosen to be the heavier fluid). OpenFOAM's standard VOF method is a so-called algebraic VOF method, employing an artificial interface compression term to prevent numerical smearing of the fluid interface, and employing the flux limiter, MULES, to keep the volume fraction field bounded. For brevity, this is often referred to simply as the MULES method. The other VOF method available in OpenFOAM® v2206 is the geometric VOF method, isoAdvector, which employs a geometric interface

* Corresponding author

Received: 16 February 2022, Accepted: 9 August 2023, Published: 2 December 2023

reconstruction step in the updating of the volume fraction field [1, 2]. This ensures a sharp and well–defined fluid interface at all times. The width of the fluid interface is typically 3-5 cells with MULES and 1-3 cells with isoAdvector. On the other hand isoAdvector is restricted to Courant numbers less than 1, whereas MULES has no such restriction. MULES and isoAdvector will form the basis for our implemented phase change and surface tension models. The TwoPhaseFlow framework makes it easy to implement other interface capturing models and combine them with available surface tension models and phase change models.

The choice of interface representation is important for implementation of phase change models. Hardt and Wondra [3] presented an interface heat resistance model using an algebraic VOF method. It is based on the Schrage equation and applies the temperature source terms in all computational cells containing fluid interface. In order to avoid interface smearing and numerical pressure oscillation, the mass source terms were not active directly in interface cells, but only in the nearest neighbours to the interface cells. Nabil and Rattner [4] released an open-source framework for the simulation of incompressible phase change phenomena with the addition of new surface tension models. The framework offers the implementation of interface heat resistance models, which are coupled explicitly with the governing equations. Among other phenomena, Nabil and Rattner [4] successfully simulated film condensation. Another class of phase change models is direct heat flux (DHF) models, which Kunkelmann and Stephan [5] found to be more accurate at the same spatial resolution when compared to the model of Hardt and Wondra [3]. These types of models require a geometrical interface representation which was achieved by reconstructing the 0.5-isosurface of the volume fraction field provided by the VOF model. The model assumes that the reconstructed interface is always at saturation temperature and computes the mass flux from the temperature gradient at the interface. Pressure oscillations were avoided using the same mass source distribution method as in Hardt and Wondra [3]. Batzdorf [6] found that the explicit treatment of the source terms in the energy equation leads to a time step restriction and instability of the model. Batzdorf formulated the gradient calculation implicitly and successfully solved these time step and stability issues. Sato and Niceno [7] constructed the interface in the same way as Kunkelmann and Batzdorf, but modified the calculation of the distance between the cell centres and the interface position to achieve an implicit coupling.

The accuracy of the phase change models depends on the precision of the temperature field, which is influenced by the spurious velocities (sometimes also referred to as parasitic currents) caused by numerical errors in the surface tension model [7]. Therefore, accurate simulations of small-scale phase change phenomena require a precise prediction of the surface tension forces. The spurious velocities occur due to discretisation errors in the pressure jump conditions at the fluid interface. The main challenge is to convert a force acting on a surface to a volumetric force in line with the cell volume averaging of the finite volume method. The pressure jump at the interface equals the surface tension multiplied by the curvature. For a constant curvature e.g. a sphere, a well-balanced solution was found by Francois et al. [8]. Thus, an exact curvature model applied to a sphere would result in near-machine precision accuracy. However, computing a curvature that converges with mesh refinement has proven very difficult. Brackbill et al. [9] proposed what is probably the most widely used surface tension model, calculating the curvature from the gradient of the volume fraction. This unfortunately produces large spurious velocities and a curvature estimate that does not converge with mesh refinement. A more accurate prediction of the curvature can be achieved with geometric VOF by exploiting the interface reconstruction data. With a parabolic fit [10] or with Reconstructed Distance Function (RDF) [11] the accuracy of the curvature calculation can be improved by more than an order of magnitude, however, still without convergence with mesh refinement. Currently, the most accurate approach to calculate the interface curvature is the height function method [10, 11], which is usually paired with geometric VOF. It achieves second-order convergence in reconstruction of a static sphere or disc. Yet, for advection of a sphere or disc in a prescribed, constant velocity field, it does not converge with mesh refinement [10, 12]. The spurious velocities are well-known for surface tension force, but can also arise from gravity forces as demonstrated by Wroniszewski et al. [13], where the choice of discretisation of the gravity term significantly affects the results. Here it is convenient to incorporate the hydrostatic potential into the pressure because this leaves us with a gravity source term which is only active on the fluid interface. The modified pressure (sometimes referred to as the dynamic pressure) then has a jump at the interface similar to the pressure jump caused by surface tension when the interface curves.

This paper presents a novel coding framework incorporating well-established VOF methods and new implementations of surface tension and phase change models. The framework facilitates the implementation of new user-defined models by simplifying the implementation and provides a benchmark suite of well-established analytical cases for easy comparison. One of the library's objectives is to encourage

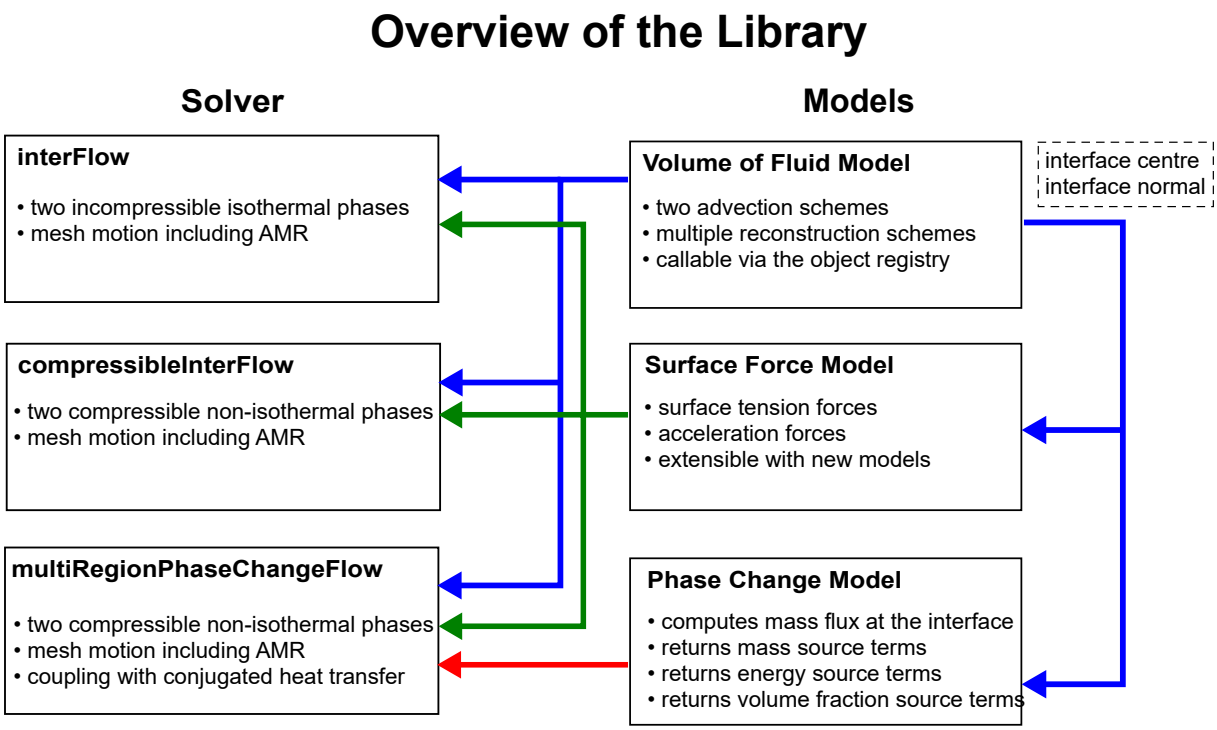


Figure 1. Schematic overview of the TwoPhaseFlow library.

the OpenFOAM community to contribute their code, boosting the reuse of already implemented models. The library extends the current capability of OpenFOAM® v2206 to simulate flows with surface tension and phase change phenomena. The accuracy of the models is compared with standard phase change and surface tension benchmarks.

2. Library Overview and governing equations

The library consists of three main modules and three solvers depicted in Fig. 1. The core of the library is the Volume of Fluid module, which advects the volume fraction field, α, and also reconstructs the interface inside each interface cell by calculating the polygon separating liquid and vapor inside the cell. The surface force and phase change modules utilize this data to compute e.g. curvature or phase change mass. This data is then provided to the top-level solvers shown in Fig. 1. The simplest solver is an incompressible solver without heat and mass transfer called `interFlow`. This is essentially identical with the standard `interFoam` solver of OpenFOAM® v2206, albeit with an option to choose between isoAdvector and MULES for the interface advection. The `compressibleInterFlow` solver extends `interFlow` with heat transfer and a compressible formulation. This solver is identical to the `compressibleInterFoam` solver available in OpenFOAM® v2206, with the option added to choose between isoAdvector and MULES. The most complex solver, `multiRegionPhaseChangeFlow`, is then built from `compressibleInterFlow` by adding mass transfer and conjugated heat transfer. All solvers are capable of utilizing automatic mesh refinement and mesh motion. The governing equation for the most general solver, `multiRegionPhaseChangeFlow` are now given.

The transport equation for the volume fraction field, α, is given by[1] [14–17]

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{u}\alpha) - \dot{\alpha}_{pc} = \alpha \nabla \cdot \mathbf{u} + \alpha(1-\alpha)\left(\frac{\psi^v}{\rho^v} - \frac{\psi^l}{\rho^l}\right)\frac{Dp}{Dt}. \tag{1}$$

Here $\mathbf{u}$ is the velocity field, p is the pressure, ψ is the compressibility and, ρ is the mass density. The superscript, l, denotes the liquid phase, and v denotes vapor phase. The two terms on the right-hand side account for the effects of volumetric changes due to heating or compression and can therefore be neglected in case of incompressible flow. The phase change is considered in the explicit source term, $\dot{\alpha}_{pc}$, as explained in Section 3.2. The first and second terms on the left hand side together represent the

[1]Strictly speaking, the volume fraction is a cell averaged quantity, not a field in the mathematical sense, and so its usage in a partial differential equation is not properly defined. In finite volume literature it is, however, common practice to ignore this, and write the partial differential equation as short-hand notation for their cell volume integrated counterpart.

passive advection of the volume fraction field. The discretisation of these two terms can be performed using either MULES or isoAdvector, as discussed in [1] and [2].

The Navier-Stokes equations are written in the form

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) - \nabla\cdot\left\{\mu_{\text{eff}}(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)\right\} = -\nabla p_{\text{rgh}} + (\mathbf{g}\cdot\mathbf{x})(\rho^l - \rho^v)\hat{\mathbf{n}}_s\delta_s + \mathbf{f}. \tag{2}$$

Here the density is calculated based on α as

$$\rho = \alpha\rho_l(T,p) + (1-\alpha)\rho_v(T,p), \tag{3}$$

where ρ_l and ρ_v are the densities of the liquid and vapour phase, each depending on the temperature, T, and pressure, p, via the user-specified equations of state (e.g. the ideal gas law). The effective turbulent viscosity, μ_{eff} is defined by

$$\mu_{\text{eff}} = \alpha\mu_l(T,p) + (1-\alpha)\mu_v(T,p) + \mu_t, \tag{4}$$

where μ_l and μ_v represent the dynamic viscosity in the liquid and vapour, respectively, each depending on temperature and pressure via user-specified viscosity models (e.g. Newtonian fluid). The turbulent eddy viscosity, μ_t, is calculated by the user-specified turbulence model ($\mu_t = 0$ for laminar flow). The auxiliary quantity, p_{rgh}, is defined as

$$p_{\text{rgh}} = p - (\mathbf{g}\cdot\mathbf{x})\rho, \tag{5}$$

where $\mathbf{g}$ is the gravity vector, $\mathbf{x}$ is the position vector. The vector, $\hat{\mathbf{n}}_s$ in Eqn. 2 represents the unit interface normal (pointing into the liquid), while δ_s is a 3D Dirac delta function used to mark the instantaneous position of the fluid interface. Numerically, working with p_{rgh} rather than p has the advantages that the specification of boundary conditions becomes simpler, and that the remaining gravity term with the Dirac delta function is only nonzero at the fluid interface. The last term, $\mathbf{f}$, on the right hand side of Eqn. (2) accounts for additional source terms such as surface tension. In the finite volume framework, the governing equations are averaged over cell volumes, and the fields ($\rho, \mathbf{u}, p$ etc.) are represented by their volume averaged values in the computational cells. In particular, for cells containing both liquid and vapor the fields represent a mixture of the local liquid and vapor field values in the cell.

The energy equation is formulated in terms of the temperature, T^i, in a two-field approach, where i is either v for vapor or l for liquid,

$$\frac{\partial\alpha^i\rho^i c_p^i T^i}{\partial t} + \nabla\cdot(\alpha^i\rho^i c_p^i T^i\mathbf{u}) = \nabla\cdot(\lambda^i\nabla T^i) + q^i_{\text{pc}} + q^i. \tag{6}$$

Here, c_p is the specific heat and λ is the thermal conductivity. The source term, q_{pc}, accounts for energy changes caused by phase change. The choice between explicit or implicit discretisation of Eqn. 6 depends on the selected phase change model, which is discussed in Section 3.2. Other effects, such as e.g. compressibility of the gas phase, are accounted for by the last term, q.

In the solid region, the heat transfer is calculated by

$$\frac{\partial\rho^s h^s}{\partial t} - \nabla\cdot\left(\frac{\lambda^s}{c_p^s}\nabla h^s\right) = 0, \tag{7}$$

where h^s is the enthalpy, ρ^s is the density, c_p^s is the heat capacity, and λ^s is the thermal conductivity of the solid.

3. Methods

In the following, we present the implementation details of the three modules forming the basis the three described solvers.

3.1. Volume of fluid module.

The Volume of Fluid module is the core part of the library as it provides the interface reconstruction data for the other models. It has already been released as open source by Scheufler and Roenby [2]. The library consists of two base classes: One for the interface reconstruction scheme and one for the interface advection scheme as illustrated in Fig. 2. The advection scheme base class facilitates the potential integration of interface advection methods beyond the VOF type, such as Level Set (LS) or phase field methods. These currently unavailable models are symbolized by "..." in Fig. 2 and must also be derived from advectionSchemes. Currently, the module includes the two previously mentioned VOF methods already available in OpenFOAM® v2206, namely MULES and isoAdvector. Here MULES does not provide the geometric interface data which we need for our phase change and surface tension models. Therefore in this work, when using MULES, we calculate the $\alpha = 0.5$ isosurface and derive the required interface position and orientation data based on that. This is a standard approach which is, however, limited to first-order accuracy. For the isoAdvector method, a more accurate interface reconstruction

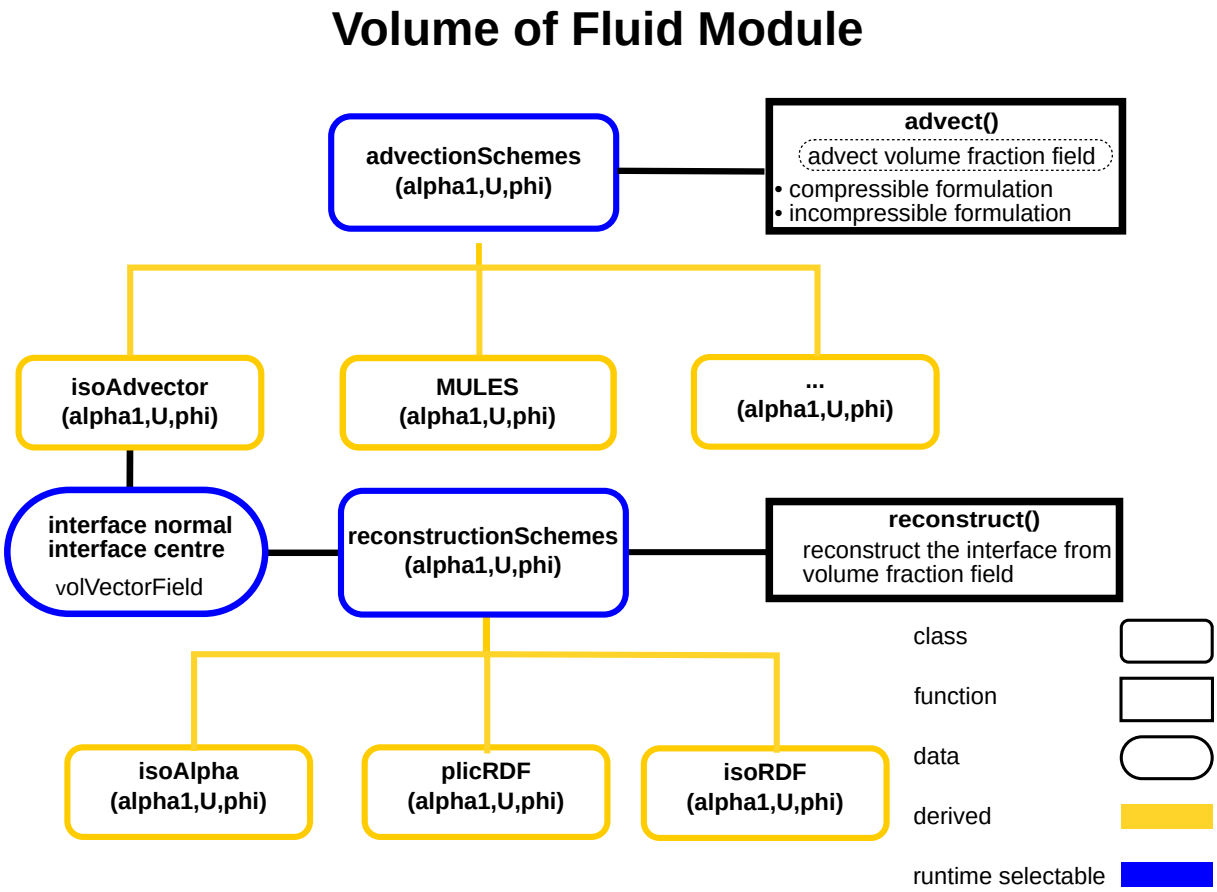


Figure 2. Schematic overview of the VOF module.

method is used which is described in [2]. It is based on a reconstruction scheme base class and two derived classes, `isoAlpha` and `plicRDF`. Here `isoAlpha` calculates the interface in a cell as an isosurface of the volume fraction as described in Roenby et al. [1]. The `plicRDF` method implements a PLIC (Piecewise Linear Interface Construction) scheme, where the interface orientation is computed as the gradient of a Reconstructed Distance Function (RDF) as described in [2]. The `isoAlpha` scheme is the fastest, but also the least accurate compared to `plicRDF`, which was demonstrated in [2] to be second-order convergent with respect to mesh refinement on hexahedral, tetrahedral and polyhedral meshes.

Both reconstruction methods ensure that the interface segment inside a cell cuts the cell into subcells with volumes in accordance with the cell's volume fraction value. With linear/planar interface segments, this means that the interface lacks C^0 continuity as illustrated in Fig. 3. The PLIC and isosurface-based reconstruction both compute the polygonal representation of the interface segment inside a cell. For each interface cell they store the interface centre point as well as the area vector of the interface segment. Each of these are stored in a `volVectorField`. For cells that are not intersected by the fluid interface, the interface centre and area vectors are set to the zero vector. These interface centre and area vector fields are required for the geometric VOF scheme, but are also used in other submodules. The global object registry manages a pointer to the reconstruction scheme, thus allowing easy access to the interface position and orientation data from other classes. The surface tension module uses this information to compute (among other things) the curvature of the surface which will then rely on the accuracy of the interface reconstruction scheme. As shown in Scheufler and Roenby [2], the interface position and orientation can be accurately predicted by the RDF method on arbitrary unstructured meshes for cases where the interface does not touch the domain boundaries. We now propose a method to extend the RDF method by enabling a prescribed contact angle, θ, on boundaries. Let $\tilde{\Psi}_{cc,\mathbf{x}_s}$ denote the distance from a cell centre, $\mathbf{x}_{cc}$, to the centre of an interface segment, $\mathbf{x}_s$, along the interface normal, $\hat{\mathbf{n}}_s$,

$$\tilde{\Psi}_{cc,\mathbf{x}_s} = \hat{\mathbf{n}}_s \cdot (\mathbf{x}_{cc} - \mathbf{x}_s), \tag{8}$$

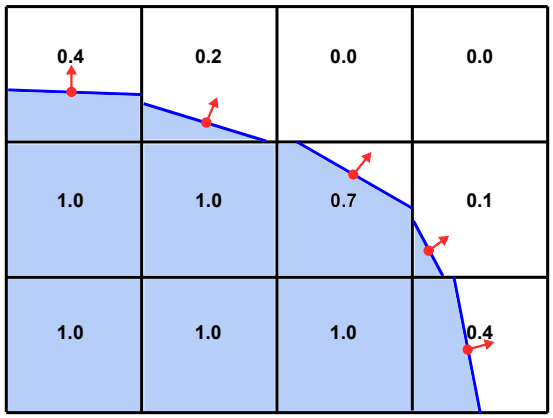


Figure 3. Example of a PLIC interface.

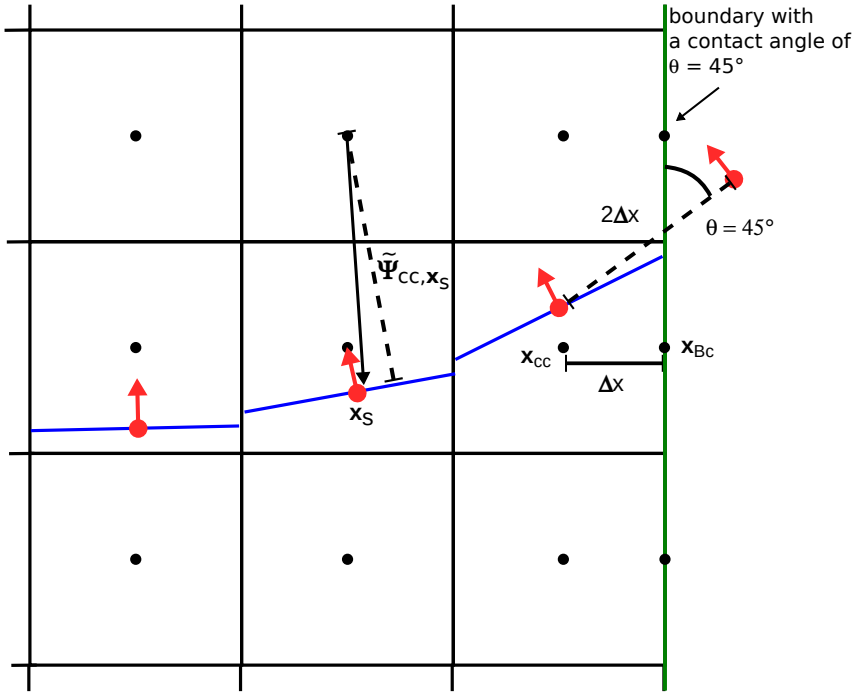


Figure 4. Idea behind boundary extrapolation of the fluid interface.

From these distances, the RDF in the centre of cell cc is calculated as

$$\Psi_{cc} = \frac{\sum_{nei} w_{nei} \tilde{\Psi}_{cc,\mathbf{x}_s}}{\sum_{nei} w_{nei}}, \tag{9}$$

where the sum is over all point neighbours of (i.e. cells that share a vertex with) cell cc that are interface cells, and the weighting factor is chosen to be

$$w_{nei} = \frac{|\hat{\mathbf{n}}_s \cdot (\mathbf{x}_{cc} - \mathbf{x}_s)|^2}{|\mathbf{x}_{cc} - \mathbf{x}_s|^2}. \tag{10}$$

The interface normal is then approximated with a least square fit as:

$$\hat{\mathbf{n}}_s = \nabla \Psi. \tag{11}$$

Since the accuracy of the normal calculation only depends on the estimation of the RDF function, special treatment must be applied at boundary cells, where the user may want to specify a contact angle. For a boundary cell with a calculated interface centre we propose to define a ghost interface point on the other side of the cell's boundary face as illustrated in Fig. 4. This ghost point is defined by first drawing a line through the interface centre, intersecting the boundary face at the user defined angle (dashed line on the left of Fig. 4). The ghost interface point is then chosen as the point along this line which is $2\Delta x$ from the

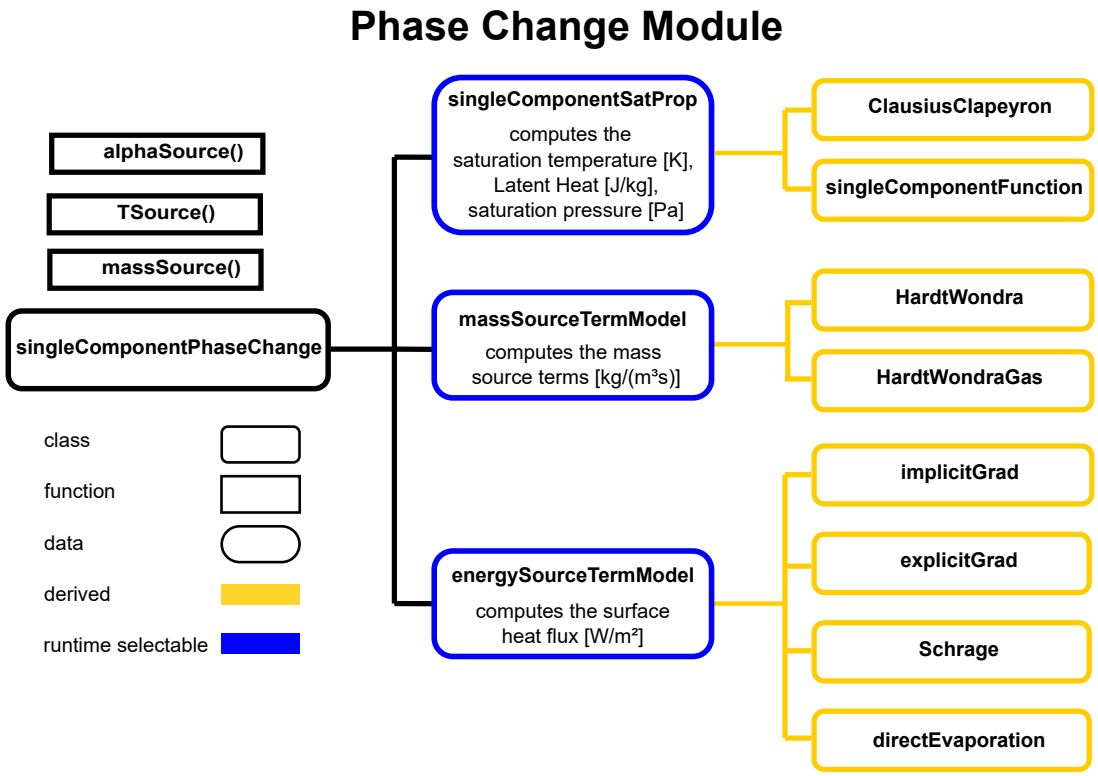


Figure 5. Schematic overview of the phase change module.

interface centre, where Δx is the distance from the boundary cell centre to the boundary face centre. The orientation of the ghost interface *normal* is chosen so that its angle with the boundary face normal equals the user-specified contact angle. These choices of ghost interface point and orientation are illustrated in Fig. 4. The ghost cell interface points and normals satisfying the contact angle requirement on the boundary are used in the weights in Eqn. (10) defining the value of the RDF function in the cell centres as determined by Eqn. (9). This method for coping with contact angles is far from optimal but is our best current approach based on extensive numerical experimentation with various approaches. Generally, the literature is very sparse with regard to interface reconstruction with the inclusion of boundary handling.

3.2. **Phase change module.**

The phase change module computes the mass transfer at the liquid/gas interface in a one-species system. The schematic overview of the module is shown in Fig. 5 and is utilised in the custom solver `multiRegionPhaseChangeFlow`. The module consists of the wrapper class `singleComponentPhaseChange` that provides source terms for the Volume of Fluid, energy and continuity equation. It wraps three runtime selectable classes implementing the specific models that are described in the following section.

The `massSourceTermModel` computes the source terms for the VOF and mass equation while the `energySourceTermModel` computes the energy source terms. The material properties relevant for saturation are managed by a class called `singleComponentSatProp`.

3.2.1. *Saturation properties.*

The `singleComponentSatProp` class provides the saturation temperature, saturation pressure and heat of evaporation. Currently, two options to specify the properties are available:

(1) Clausius–Clapeyron relation :

$$\ln(p/p_1) = -\frac{L}{R}\left(\frac{1}{T_1} - \frac{1}{T}\right) \tag{12}$$

(2) Functions of temperature and pressure:

$$\begin{aligned} T_{Sat} &= f(p), \\ p_{Sat} &= g(T), \\ L &= h(p) \end{aligned} \tag{13}$$

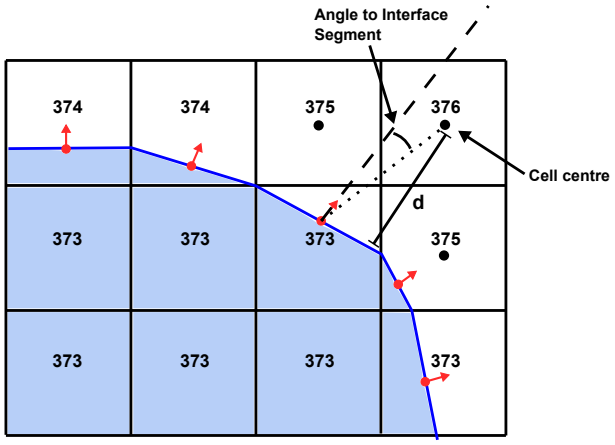


Figure 6. Gradient estimation of the `explicitGrad` model.

In the Clausius–Clapeyron relation, one must specify the latent heat, L, the specific gas constant, R, as well as a point, (T_1, p_1) on the saturation curve in order to describe the saturation temperature and pressure. For a large pressure range, the assumption of constant latent heat can be erroneous. Specifying the saturation properties as a polynomial function of pressure or temperature or interpolating it from a table is a more accurate approach, which is, therefore, an option in the implemented code.

3.2.2. *Energy source terms.*
The class, `energySourceTermModel`, is the core of the phase change library. It computes the energy source terms and provides it to the energy equation of the solver, Eqn. (6), by returning a matrix. Additionally, these source terms are also used in the mass source term model that (currently) smears and scales the provided source terms. In the literature, two types of energy source models are described [5]: 1) gradient-based models and 2) interface heat resistance, both of which are implemented in the framework. The gradient-based models compute the volume-specific power associated with phase change as

$$q_{pc} = q^l_{pc} + q^v_{pc} = \lambda^l \nabla T^l \cdot \hat{\mathbf{n}}_s + \lambda^v \nabla T^v \cdot (-\hat{\mathbf{n}}_s). \tag{14}$$

The interface heat resistance models calculate the power as

$$q_{pc} = q^l_{pc} + q^v_{pc} = \frac{T^l - T_{Sat}}{R_{int}} + \frac{T^v - T_{Sat}}{R_{int}}, \tag{15}$$

where T_{Sat} is the saturation temperature. The models are implemented as derived classes of `energySourceTermModel` and compute heat transfer due to phase change. The computed heat flow at the interface is used to model the phase change mass flow and the resulting velocity jump, which is described in more detail in Section 3.3.

explicitGrad
This model is a simple implementation of a gradient-based model and is similar to the model described in Kunkelmann [5]. With the assumptions that the interface is on saturation temperature and that energy can only be transported by diffusion over the interface, the heat flux q_{pc} can be computed by the Fourier law. The challenging part is the discretisation of the temperature gradient at the interface depicted in Fig. 6. For an interface cell with interface centre $\mathbf{x}_s$ and interface normal $\hat{\mathbf{n}}_s$, we chose to calculate the one-sided temperature gradient on the liquid side based on the temperature T_{nei} in the neighbour cell towards which $\hat{\mathbf{n}}_s$ is pointing. That is, if a neighbour cell has centre $\mathbf{x}_{\text{nei}}$, then we choose the neighbour where $\mathbf{x}_{\text{nei}} - \mathbf{x}_s$ makes the smallest angle with $\hat{\mathbf{n}}_s$. From this choice, we then approximate the temperature normal derivative as

$$\hat{\mathbf{n}}_s \cdot \nabla T^l \approx \frac{T_{\text{nei}} - T_{Sat}}{\hat{\mathbf{n}}_s \cdot (\mathbf{x}_s - \mathbf{x}_{\text{nei}})}. \tag{16}$$

The one-sided normal temperature gradient in the vapor phase, $\hat{\mathbf{n}}_s \cdot \nabla T^v$ is calculated in the same way, but with the chosen neighbour cell in the vapor phase, i.e. the cell pointed at by $-\hat{\mathbf{n}}_s$ due to the convention that $\hat{\mathbf{n}}_s$ points out of the vapor region.

With the normal gradients computed both on the vapor and liquid side of the fluid interface, we can calculate the heat flux using Eqn. (14). The computed heat flux is multiplied by the interface area within a cell (provided by the Volume of Fluid module) and applied as explicit source term in Eqn. (6). Due to the explicit nature of the source term, a stability criterion enforces a time step limitation on the solver [6].

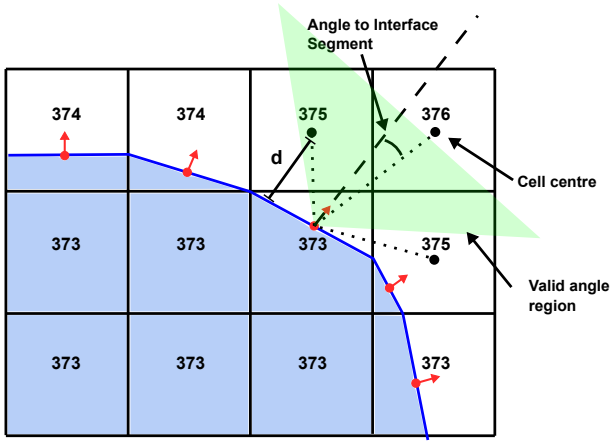


Figure 7. Gradient estimation of the `implicitGrad` model.

implicitGrad

To circumvent the time step constraint and stability problems that may arise with the explicit variant of the gradient-based models, Batzdorf [6] proposed an implicit formulation that forms the basis for the `implicitGrad` model. The basic idea of this approach is to include part of the gradient on the diagonal of the matrix in the discretised energy equation,

$$q^i_{\text{pc}} = \sum_{\text{nei}} \frac{w_{\text{nei}} \lambda^i}{d_{\text{nei}}} T_{Sat} - \sum_{\text{nei}} \frac{w_{\text{nei}} \lambda^i}{d_{\text{nei}}} T_{\text{nei}} \text{ with } w_{\text{nei}} = \left(\frac{\cos\theta_{\text{nei}}}{\sum_m \cos\theta_m} \right)^4. \tag{17}$$

The angle $\theta_{\text{nei}} = \hat{\mathbf{n}}_s \cdot (\mathbf{x}_{\text{nei}} - \mathbf{x}_s)$, and for a given interface cell the sums are over all neighbour cells, where this angle is less than 70 degrees. Figure 7 illustrates this choice of neighbour cells. In contrast to the explicit variant, the source term is not applied directly in the interface cells but in the neighbouring cells. The second sum in Eqn. (17) is treated implicitly and hence adds to the diagonal of the matrix, while the first sum is added to the source term of the matrix equation. With this approach, no explicit time step criterion exists, which significantly improves the stability of the solver.

Schrage

The Schrage model is implemented as an interface heat resistance model. In this type of model the heat flux is computed from the temperature difference between the cell temperature, T, and the saturation temperature, T_{Sat}, as follows,

$$q_{pc} = q^l_{pc} + q^v_{pc} = \frac{T^l - T_{Sat}}{R_{int}} + \frac{T^v - T_{Sat}}{R_{int}}, \tag{18}$$

where the coefficient, R_{int}, is defined by

$$R_{int} = \frac{2 - C_{acc}}{2C_{acc}} \frac{T^{3/2}_{Sat} \sqrt{2\pi R_{gas}}}{\rho^v L^2}. \tag{19}$$

Here C_{acc} is the accommodation factor, R_{gas} is the specific gas constant and L is the latent heat. As in the implicit gradient model, the first term of Eqn. (18) is added to the source of the matrix, while the second term adds to the diagonal of the matrix. This implicit treatment increases the stability of the solver. The difference between our implementation and the model proposed by Hardt and Wondra [3] is in the calculation of the interface area where they use $|\nabla\alpha|$ while we use the geometrically calculated interface area provided by the VOF module. Furthermore, our implementation employs a temperature field for each equation rather than a single field.

Direct evaporation

The direct evaporation model is a combination of the interface heat resistance model and the gradient-based model. This "engineering model" requires the specification of the superheated temperature and an interface heat resistance coefficient. If the temperature exceeds the superheated temperature the liquid is evaporated but with the assumption that the resulting volume increase instantly moves to the interface.

3.3. **Mass source terms.**

Phase change causes a velocity jump at the interface, which is modeled by applying source terms in the pressure Poisson equation. The magnitude of the velocity jump is proportional to the volume-specific mass flux, $\dot{\rho}_{sharp}$, at the interface, which we calculate as

$$\dot{\rho}_{sharp} = \frac{q_{pc}\,|\mathbf{n}_s|}{LV}. \tag{20}$$

Here q_{pc} is the heat flux from the `energySourceTermModel`, $|\mathbf{n}_s|$ is the surface area in the cell obtained from the VOF model. L and V are the latent heat and the cell volume, respectively. This results in a sharp source term distribution at the interface, especially for the kind of geometrically reconstructed interface used here. Applying the resulting source term field directly at the interface results in pressure and velocity oscillation as well as a smearing of the interface [3]. The implemented mass source term models circumvent the problem by smearing the source terms and by only applying them in the neighbourhood of the interface and not directly at the interface. The `massSourceTermModel` provides the source terms for the continuity equation and for the VOF equation.

3.3.1. *HardtWondra.*

The Hardt and Wondra [3] model avoids the pressure oscillation and a smearing of the interface by smoothing the sharp source term distribution provided by the `energySourceTermModel`. A detailed description of the implementation can be found in Kunkelmann [5] and Batzdorf [6]. In the first step, the sharp source term distribution is smeared with a Laplacian function,

$$\dot{\rho}_{smeared} - \nabla^2(D\dot{\rho}_{smeared}) = \dot{\rho}_{sharp}, \tag{21}$$

with the numerical diffusion coefficient defined by $D = (C\Delta x)^2$, Δx being the cell size. The coefficient, C, is roughly the number of cells over which the interface is smeared (the default value is set to $C = 3$). The second step is to set all source terms to zero in the interface region defined by: $\text{cutOff} < \alpha < 1 - \text{cutOff}$ with the default value $\text{cutOff} = 1 \cdot 10^{-3}$. Laplacian smoothing keeps the volume integral of the source terms constant and is therefore a conservative operation. Obviously, setting the source terms to zero inside the interface region violates this conservation. Therefore, in the next step the source terms of the liquid and gas part are scaled in such a way that the volume integral matches the initial volume integral in the gas and liquid phase:

$$\begin{aligned} \int \dot{\rho}_{sharp}\,\mathrm{d}V &= N_v \int \alpha_v \mathcal{H}(\alpha_{cutOff} - \alpha_l)\dot{\rho}_{smeared}\,\mathrm{d}V \\ &= N_l \int \alpha_l \mathcal{H}(\alpha_{cutOff} - \alpha_v)\dot{\rho}_{smeared}\,\mathrm{d}V \end{aligned} \tag{22}$$

The last step is to switch the sign of the smeared source terms in the liquid part to account for the mass loss. We end up with two source term distributions with different signs on the two sides of the interface where the volume integral of the mass source term distribution is zero. With this operation, we subtract mass from the liquid side and add it to the gas side. Hence, no source terms are needed for the VOF equation because changes in the liquid content are handled by the continuity equation.

The advantage of this approach is that it is easy to implement and that it works for arbitrary cell shapes. The disadvantage is that the smearing may cause the non-physical removal of liquid and requires a fine resolution near the interface.

3.3.2. *HardtWondraGas.*

The `HardtWondraGas` model is identical to the Hardt and Wondra model just described, with the exception that the liquid loss is accounted for in the VOF equation. Hence, there are no source terms in the pressure Poisson equation in the liquid. Instead, a sharp source based on $\dot{\rho}_{sharp}$ from Eqn. 20 is used in the VOF equation:

$$\dot{\alpha}_{pc} = \dot{\rho}_{sharp}/\rho_l. \tag{23}$$

With this approach, velocities caused by the mass source terms in the liquid are no longer present. On the other hand, the shrinkage in a thin liquid film can be represented more accurately by the sharp source term, because it ensures removal or addition of liquid in the correct locations. An additional benefit is that in some scenarios, the stability of the solver can be improved since inaccuracies in the mass conservation can lead to the creation of cells with a tiny amount of liquid ($\alpha \approx 10^{-6}$). In the case of the geometric VOF scheme, a small interface segment would be found in that cell which may cause a significant amount of phase change in that region. With the Hardt and Wondra model [3] these tiny liquid volumes cannot evaporate and may accumulate during the simulation.

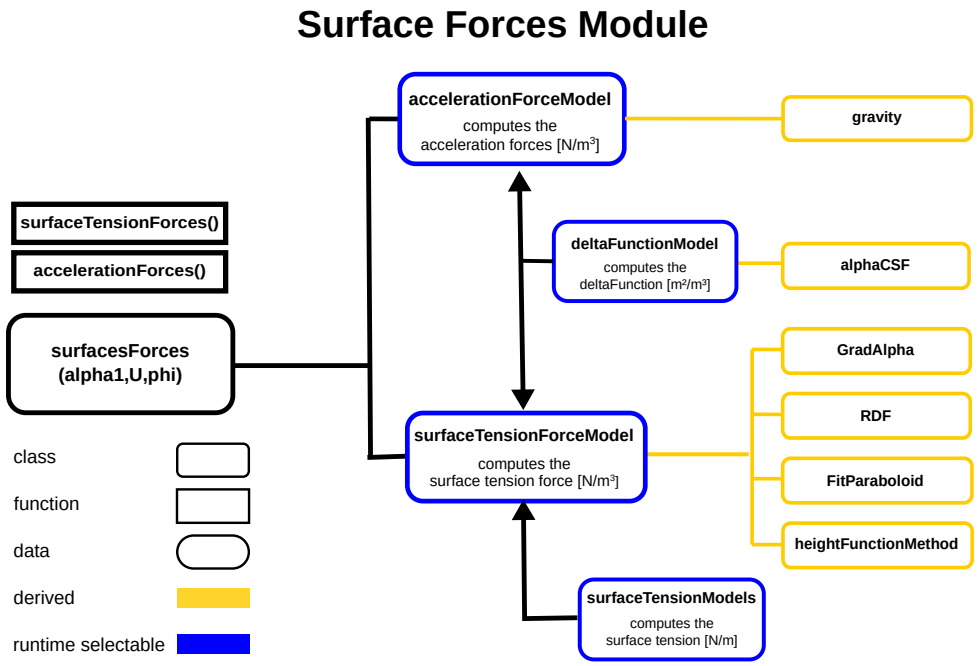


Figure 8. Schematic overview of the surface forces module.

3.4. **Surface force module.**

One challenge in the simulation of multiphase flows is the reduction of spurious velocities. These spurious velocities are induced around the interface by an imperfect discretisation of the pressure jump condition. This effect is well known for surface tension-driven flow but can also arise from external accelerations [13, 18].

Both the surface tension and acceleration forces induce a pressure jump over the interface. In the case of surface tension, this is the expected behaviour. All the OpenFOAM interfacial flow solvers, `interFoam`, `interIsoFoam` and `compressibleInterFoam` employ a pressure where the hydrostatic potential is subtracted, $p_{\mathrm{rgh}} = p - \rho \mathbf{g} \cdot \mathbf{x}$. This formulation reduces the spurious velocities [13] and simplifies the definition of the hydrostatic pressure boundary condition [19]. The acceleration force can be written in the form,

$$\mathbf{F}_{a,f} = (\rho^l - \rho^v)(\mathbf{a} \cdot \mathbf{x})\hat{\mathbf{n}}_s \delta_s, \tag{24}$$

where $\mathbf{a}$ is the acceleration vector. The surface tension takes a similar form,

$$\mathbf{F}_{\mathrm{st,f}} = \sigma \kappa \hat{\mathbf{n}}_s \delta_s, \tag{25}$$

with the surface tension σ and the curvature κ. Both forces share the same mathematical background, which is explained in Popinet [19] and Ghidaglia [20] in more detail. Both the acceleration force and surface tension force must be calculated on the face centre to achieve a well-balanced formulation with the pressure equation.

This formulation forms the basis of the surface forces module and is depicted in Fig. 8. It shows an overview of the implemented framework with runtime selectable classes and the currently available models, which will be described in the following subsections. The class, `surfaceForces`, handles the interface to the solver used in Eqn. 2. The `accelerationModel` reflects the factor $\mathbf{a} \cdot \mathbf{x}$, the `deltaFunctionModel` the factor $\hat{\mathbf{n}}_s \delta_s$, the `curvatureModel` the factor κ and `surfaceTensionModels` the factor σ. This design allows body force implementations based on other interface representations than VOF, e.g. LS or front-tracking, to be integrated. Marangoni convection could be modelled with the `surfaceTensionModels` as it allows for the definition of temperature dependent surface tension, σ. However, throughout this paper, we assume that the surface tension is constant.

3.4.1. *Delta function model.*

The surface forces act on a microscopic region at the interface that is typically significantly smaller than the cell size. But the discretisation of the function in the finite volume framework requires a region of at least one cell size to convert it into a volumetric force. There are multiple approaches found in literature [19] to discretise $\hat{\mathbf{n}}_s \delta_s$. Currently, only the Continuous Surface Force (CSF) method proposed

by Brackbill et al. [9] is available in our framework,

$$\hat{\mathbf{n}}_s \delta_s = \nabla \alpha, \tag{26}$$

where the gradient discretisation scheme is chosen in `fvSchemes`.

3.4.2. *Curvature model.*
The prediction of the curvature is essential for an accurate simulation of the surface tension-driven flow. If the exact curvature is known, the spurious velocities will drop to zero as the well-balanced formulation of the force is implemented here as in the standard solver in OpenFOAM. The computation of the curvature is extremely challenging and numerous models have been proposed to tackle that issue. This library provides multiple implementations of curvature models that can be selected in `transportProperties` for the `interFlow` solver and in `thermophysicalProperties` for `compressibleInterFlow` and `multiRegionPhaseChangeFlow`.

gradAlpha
The `gradAlpha` method is identical to the standard OpenFOAM formulation in the multiphase solvers. The model is based on the paper of Brackbill et al. [9] and computes the curvature directly from the volume fraction field, α. The computation consists of two steps: 1) the computation of the normal with the default gradient operator:

$$\hat{\mathbf{n}}_\alpha = \frac{\nabla \alpha}{|\nabla \alpha|} \tag{27}$$

2) Subsequently, the normals are interpolated to the faces, normalised and the curvature is computed as the divergence of the normals on the faces,

$$\kappa = \frac{1}{V} \int_V \nabla \cdot \hat{\mathbf{n}}_\alpha dV \approx \frac{1}{V} \sum_f \hat{\mathbf{n}}_{\alpha,f} \cdot \mathbf{S}_f, \tag{28}$$

where the integral is over the cell volume, V, and the sum is over all faces of the cell (assuming here the face area vectors, $\mathbf{S}_f$, to point out of the cell). The contact angle in this implementation is treated by setting the interface normal on the boundary face to the prescribed contact angle. The implementation of this model is straightforward and is probably the most frequently used model for curvature computation in combination with VOF based multiphase solvers. The accuracy of the model can be increased drastically by using a least square based gradient method, `pointCellLeastSquares`, for the volume fraction field in `fvSchemes`.

Reconstructed Distance Function
The Reconstructed Distance Function method (RDF) is based on an implementation of the model proposed by Cummins et al. [11]. The RDF model shares a lot of similarities with coupled LS-VOF models. The most significant difference is that in the RDF model the signed distance function, Ψ, is not found by solving a PDE as in the LS method, but is instead constructed geometrically based on the volume fraction field.

The first step of the algorithm is the reconstruction of the signed distance function, Ψ, in a narrow band around the interface. The value of Ψ in every cell centre of the narrow band is computed as

$$\Psi_{cc} = \hat{\mathbf{n}}_s \cdot (\mathbf{x}_{cc} - \mathbf{x}_s). \tag{29}$$

After that, we calculate the gradient of the RDF:

$$\hat{\mathbf{n}}_\Psi = \frac{\nabla \Psi}{|\nabla \Psi|} \tag{30}$$

By definition, the gradient of a signed distance function has a length equal to one, but due to discretisation errors, this is not necessarily the case after the numerical calculation. Therefore, we normalise the calculated vector by dividing it by its length.

The curvature is then computed by interpolating the normal, $\hat{\mathbf{n}}_\Psi$, from cell centres to faces and applying the Gauss-Green gradient method,

$$\kappa = \nabla \cdot \hat{\mathbf{n}}_\Psi \rightarrow \kappa_{cc} \approx \frac{1}{V_{cc}} \sum_f \hat{\mathbf{n}}_{\Psi,f} \cdot \mathbf{S}_f, \tag{31}$$

where V_{cc} is the cell volume, the sum is over all the cell's faces and $\mathbf{S}_f$ is the face area vector pointing out of the cell. This method is accurate on structured grids but causes inaccuracies on unstructured grids due to interpolation errors. These can be reduced by computing the curvature as

$$\kappa = tr(\nabla \hat{\mathbf{n}}_\Psi). \tag{32}$$

While the two formulations are mathematically identical, the latter allows the usage of the numerical least square gradient method, which is more accurate on unstructured grids because the Gauss-Green gradient method is only zeroth order accurate on unstructured grids [21]. By setting the keyword `curvFromTr`, the user can switch between the two formulations. After the computation of Eqn. (32), we have the curvature at the cell centres, which would limit the scheme to first-order accuracy. The accuracy can be increased by interpolating the curvature to the interface centres using OpenFOAM's `cellPointInterpolation` class, which interpolates cell-centred data to any point in the computational domain.

To improve the accuracy of the normal calculation near boundary faces, they are included in the stencil. The value of the signed distance function, Ψ on the boundary is computed with the extrapolated interface information described in Section 3.1. After the computation of the normal with the gradient operator, the boundary values of the normals are set to the prescribed contact angle as in `gradAlpha` model.

The accuracy of the method can be significantly influenced by the accuracy of the gradient operator specified in `fvSchemes`, which makes OpenFOAM's `pointCellLeastSquares` the recommended choice.

fitParaboloid

The `fitParaboloid` method estimates the curvature by fitting a local parabola (2D) or paraboloid (3D) to the interface centres in a cell and its neighbours as provided by the interface reconstruction scheme. The local function is approximated by:

$$\begin{aligned} f(x,z) &= C_0 x + C_1 x^2 + z && \text{2D} \\ f(x,y,z) &= C_0 x + C_1 x^2 + C_2 y + C_3 y^2 + C_4 xy + z && \text{3D} \end{aligned} \tag{33}$$

Here x, y and z refer to a local rotated coordinate system with z pointing along the interface normal. The coefficients of the equation are found with a least square minimisation procedure, where the resulting linear system is efficiently solved using LU factorisation. After obtaining the coefficients, the derivatives of the Eqn. 33 can be calculated analytically and the curvature is computed as

$$\kappa = \begin{cases} \frac{f_{xx}}{(1+f_x^2)^{3/2}} & \text{2D} \\ \frac{f_{xx}(1+f_x)+f_{yy}(1+f_y)-2f_x f_y f_{xy}}{(1+f_x^2+f_y^2)^{3/2}} & \text{3D} \end{cases} \tag{34}$$

The handling of prescribed boundary conditions is straightforward and achieved by including the extrapolated centre value in the stencil (see Section 3.1).

Height Function Method

The Height Function Method (HFM) is a simple and second-order accurate method for the computation of curvature [10]. The height function method estimates the curvature directly from the volume fraction field in a way similar to the method proposed by Brackbill [9]. However, it does not compute the curvature based on the gradient of the volume fraction field, but instead with column heights. These columns represent the interface in the form $H(x, y, f(x, y))$ and the derivatives of this function H are used to compute the curvature given in Eqn. (34). The derivatives are computed with a central second-order accurate finite difference operator. The implementation is straightforward on structured grids and is successfully used in the basilisk flow solver [22]. Attempts to implement this method on unstructured grids result in considerably more complex implementation, but so far without achieving the second-order accuracy [23] [24]. The following variant of the height function method is only utilised in structured parts of the mesh, where the cells are cubes. The open source meshing tools cfMesh [25] or snappyHexMesh [26] generate a mesh with cubic cells in the interior of the domain and body-fitting polyhedra at the boundary as depicted in Fig. 11. This allows usage of the accurate height function method in the interior (the majority) of the mesh while falling back to less accurate methods near the boundaries. The general outline of such a hybrid approach is given in Algorithm 1.

First, we have to classify the cubic cells in the grid. A cell is defined as cubic, if 1) it has six faces and 2) all vectors from the cell centre to the neighbour cell centres form an orthogonal base (with an angle tolerance of 0.001 degrees. An example of such a classification is depicted in Fig. 11 where cells marked red are considered cubes. If an interface cell is not a cube, or the height function method fails, a paraboloid is fitted in the neighbouring interface centres as described in Section 3.4.2.

The height function, $H(x, y, f(x, y))$, is the accumulated volume fraction field in the coordinate directions, x, y or z, of the structured grid. Figure 9 shows computed column heights for a structured grid in the x-direction. The computation of the heights for a given cell is considered successful if all columns at a given column index only contain liquid on one side and only gas on the other side. This is the case in Fig. 9 if we look two cells to the right and two cells to the left from the central column. So the average

Algorithm 1: Outline of the height function method

```
1  Classify cuboid cells
2  Calculate interfaceCells: A list of all the interface cell indices, i.e. i ∈ interfaceCells if ϵ < α_i < 1 − ϵ.
3  for celli in interfaceCells do
4      if isCuboid then
5          vector n = surfaceNormal[celli].normalise()
6          for dir in sortAbsoluteComponents(n) do
               // function computeHeight is described in Algorithm 2
7              curv, foundHeight = computeHeight(dir)
8              curvature[celli] = curv
9              if foundHeight then
10                 break
11             end
12         end
13         if not foundHeight then
14             curvature[celli] = fitParaboloid()
15         end
16     else
17         curvature[celli] = fitParaboloid()
18     end
19 end
```

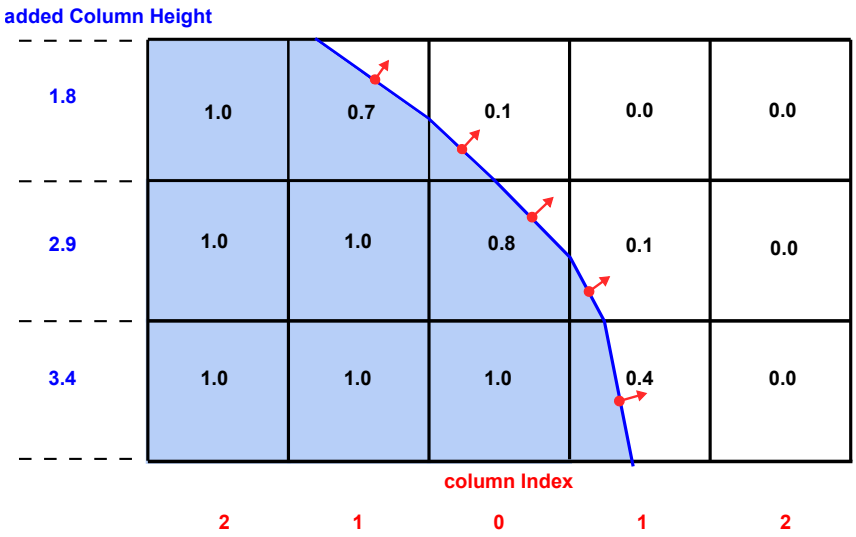


Figure 9. Volume fraction field with the computed column heights.

of the column to the left of the surface would be one and the average of the surface to the right of the interface would be zero.

Algorithm 2 describes the procedure of adding the heights in the given direction. The basic idea is to compute the height for the first column index (see Fig. 9) and then advance in the positive and negative directions and add the height in that location. After the accumulation of the height values, second-order accurate finite difference operators are used to compute the heights.

To advance in the given direction, the unstructured cell-point-cell stencil needs to be sorted to mimic structured stencil addressing as depicted in Fig. 10. The position in the stencil of 27 (in 3D) or 9 (in 2D) cells is calculated with structured coordinate directions i, j and k as position $= i + 3j + 9k$. With this addressing, the next cell label in the unstructured grid for the given positive and negative direction can easily be computed.

In parallelised cases, where the next cell in the current direction is on a neighbour processor, this neighbour processor will continue to accumulate heights. For this task, the neighbour processor needs the direction and status of the iteration as well as the cell index for which the height functions need to be computed. With this information, the neighbour processor can perform the height accumulation and send the results back to the original processor.

Algorithm 2: ComputeHeights.

```
def computeHeight(direction):
    // construct 2D direction-aligned stencil
    twoDimFDStencil cols(celli,dir)
    // compute column height for col index 0
    cols.addColumnHeight(celli)
    label dir_pos, dir_neg = cols.nextCellsInDirection()
    // advance the col index in pos and neg direction Fig. 9
    avgHeightValPos, avgHeightValNeg = 0.5
    for iter in [1 ... 7] do
        // is column empty avgHeight = 0 , full avgHeight = 1
        if foundHeight(avgHeightValPos) then
            avgHeightValPos = cols.addColumnHeight(dir_pos)
            label dir_pos = cols.nextCellsInDirection(pos=True)
        end
        if foundHeight(avgHeightValNeg) then
            avgHeightValNeg = cols.addColumnHeight(dir_neg)
            label dir_neg = cols.nextCellsInDirection(pos=False)
        end
    end
    // full and empty col found?
    bool foundHeight = foundHeight(avgHeightValPos,avgHeightValNeg)
    scalar curv = cols.calcCurvature()
    return curv, foundHeight;
```

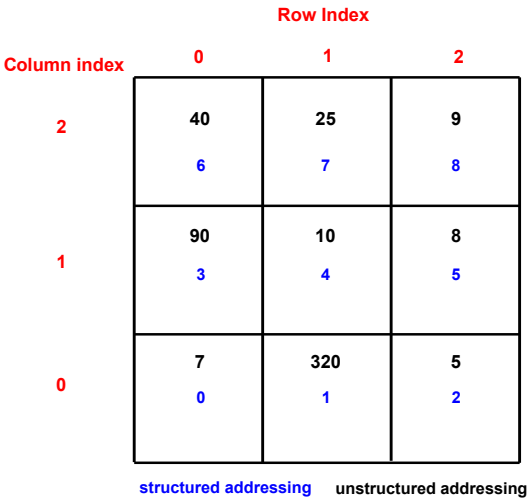


Figure 10. Height function stencil.

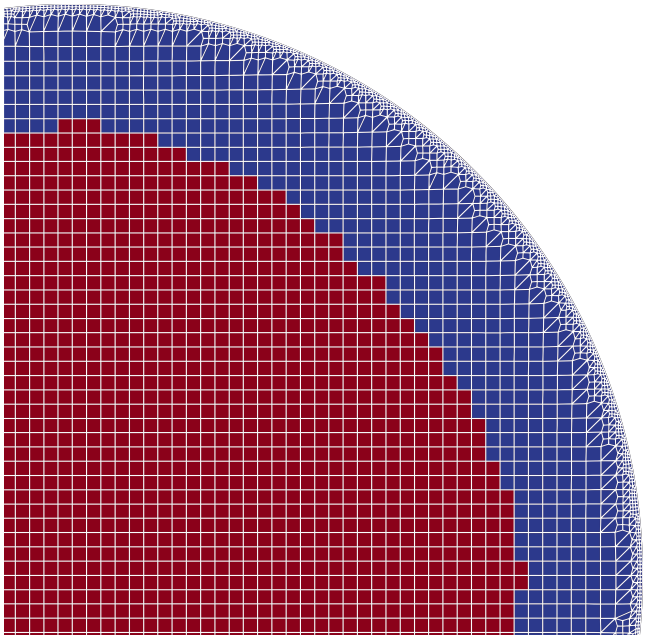


Figure 11. Cubic cells are marked in red.

3.5. **Acceleration model.**

It is well-documented in literature that spurious velocities can arise at the fluid interface for surface tension-dominated flows due to inaccurate curvature estimation. Perhaps less well–known are the spurious velocities arising from numerical errors introduced in the discretisation of the gravity term (second term on the right hand side of Eqn. (2)). There is a need for experimenting with different solutions to this problem. The framework offers the possibility to implement new gravity or acceleration models. Currently, only a single `gravity` model is available which is identical to the standard OpenFOAM implementation, i.e.,

$$\mathbf{F}_{a,f} = (\mathbf{g} \cdot \mathbf{x}_f)\hat{\mathbf{n}}_f \cdot \nabla\rho, \tag{35}$$

where $\mathbf{x}_f$ is the face centre and $\hat{\mathbf{n}}_f$ the unit normal to the face.

4. Validation

In the following, several benchmark test cases for the validation of the surface tension and phase change model are presented and compared to analytical solutions.

The framework allows easy implementation of new models and simplifies the testing by providing established numerical benchmarks. To simplify the parameter studies needed to verify the implementation and compare results with different models, we use the open-source library caseFoam [27]. It provides an easy way of generating a series of simulation test cases where various parameters are scanned within a specified range. The post processing data of the parameter studies can then easily be analysed and compared.

4.1. Phase change.

The easiest and most accurate way to test an implementation is to compare its results with known analytical solutions. For the phase change model, three analytical solutions are widely used: The Stefan problem, the sucking interface problem and the Scriven problem. The different models can be selected by changing the entries in the `phaseChangeProperties` dictionary:

```
// options: selectedGradExplicit implicitGrad Schrage
energySourceTermModel implicitGrad;

implicitGradCoeffs
{
}

// options: hardtWondra hardtWondraGasPhase
massSourceTermModel hardtWondra;
hardtWondraCoeffs
{
}

satProperties
{
    singleComponentSatProp function;
    Tmin 100;
    Tmax 500;
    pSat constant 1e5;
    TSat constant 373.15;
    L constant 2.26e6;
}
```

4.1.1. *Stefan problem.*

This one-dimensional validation test case describes the interface motion away from a superheated wall and is one of the most frequently used validation test cases [7] [5] [6] [3]. A gas column separates the superheated wall and the liquid. Due to energy transfer from the wall through the gas phase to the liquid interface, evaporation at the interface occurs, causing an interface displacement away from the wall. The analytical interface motion is given by

$$x(t) = 2\beta\sqrt{a_v t}. \tag{36}$$

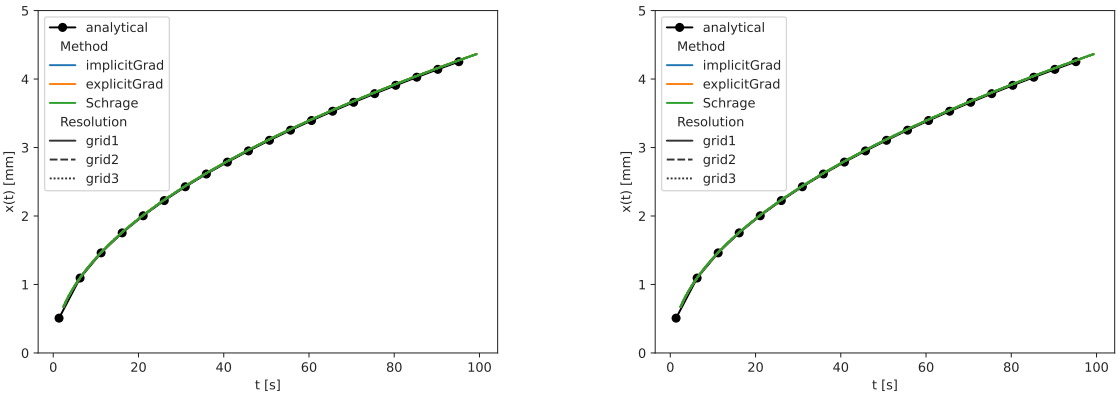
where a_v is the thermal diffusivity, and β is the solution to the transcendental equation (see [7])

$$\beta \cdot \exp(\beta^2) \cdot \mathrm{erf}(\beta) = \frac{c_p^v (T_{\mathrm{Wall}} - T_{\mathrm{Sat}})}{\sqrt{\pi} L}. \tag{37}$$

In our test case, the wall is superheated with $T_{\mathrm{Wall}} = T_{\mathrm{Sat}} + 5K$ and the thermodynamic properties of the fluids are given in Tab. 1. The grid is one-dimensional and has a length of 10 mm with a resolution of 50 (Grid1), 100 (Grid2) or 200 cells (Grid3). The solutions of the simulations are shown in Fig. 12a and 12b. All models deliver accurate results for all grid sizes and advection schemes.

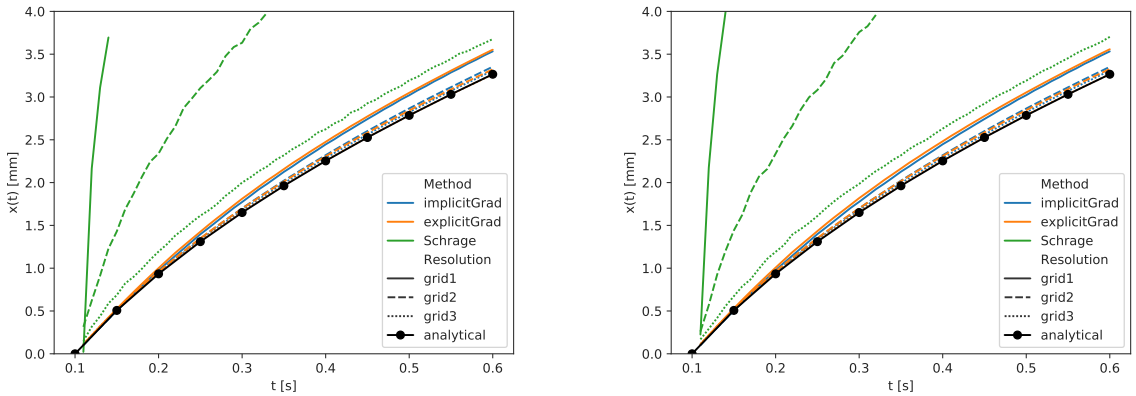
Table 1. Thermal properties of the mesh regions.

	R [J/(kgK)]	ρ [kg/m^3]	c_p [J/(kgK)]	λ [W/(mK)]	L [kJ/kg]
vapor	461.4	0.581	2030	0.025	2260
liquid	-	958.4	4216	0.671	2260



(a) Geometric VOF (isoAdector). **(b)** Algebraic VOF (MULES).

Figure 12. Stefan problem: Interface position as function of time. All calculated curves coincide with the analytical solution.

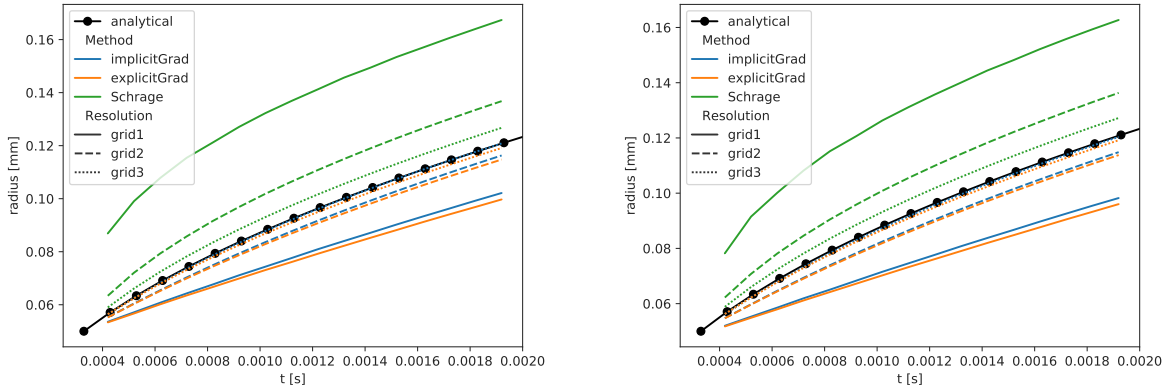


(a) Geometric VOF (isoAdector). **(b)** Algebraic VOF (MULES).

Figure 13. Sucking interface: Comparison of the interface position.

4.1.2. *Sucking interface.*
Another frequently encountered and slightly more complex one-dimensional test case is the sucking interface problem introduced by Welch and Wilson [28] which became one of the standard test cases [5] [7]. The model describes the interface movement for a one-dimensional superheated column. As in the Stefan problem, gas is located between a wall and the liquid but here the liquid is superheated and both gas and wall are at saturation temperature. The superheated liquid evaporates at the interface, creating volume and pushing the liquid away from the wall.

In our case setup, the values of the analytical solution are given with the thermophysical properties of Tab. 1 and with a superheated fluid temperature of 5 Kelvin. As in the previous simulation, the length of the domain is 10 mm, but we now use grids with cell counts of 100, 200 and 400. Figure 13a and 13b show the parameter study for the three grid sizes and the available models. The influence of the advection scheme is observed to be neglectable, whereas a strong dependency on the choice of phase change treatment can be observed. The most accurate is the `implicitGrad` model, followed by the `explictGrad` model, whereas the `Schrage` model is significantly less accurate.



(a) Geometric VOF (isoAdector). **(b)** Algebraic VOF (MULES).

Figure 14. Bubble in superheated liquid: Comparison of the interface position.

4.1.3. *Bubble in superheated liquid.*
The last phase change benchmark test case included here is the bubble in a superheated liquid. An analytical solution for this test case was provided by Scriven [29], which was adopted as a benchmark test case [5] [7] [6] for phase change models. The analytical evolution of the radius is similar to the Stefan Problem and is given by

$$x(t) = 2\beta_S\sqrt{\alpha_l t}, \tag{38}$$

where β_S is the solution of the transcendental equation,

$$\frac{\rho^l c_p^l (T_\infty - T_{Sat})}{\rho^v (L + (c_p^l - c_p^v)(T_\infty - T_{Sat}))} = 2\beta_S^2 \int_0^1 e^{-\beta_S^2\left((1-\xi)^{-2} - 2\left(1-\frac{\rho^v}{\rho^l}\right)\xi - 1\right)} d\xi. \tag{39}$$

The initial bubble radius, R, and radial temperature distribution must be specified. For $r \leq R$ we set the initial $T = T_{Sat}$. For $r > R$ we use

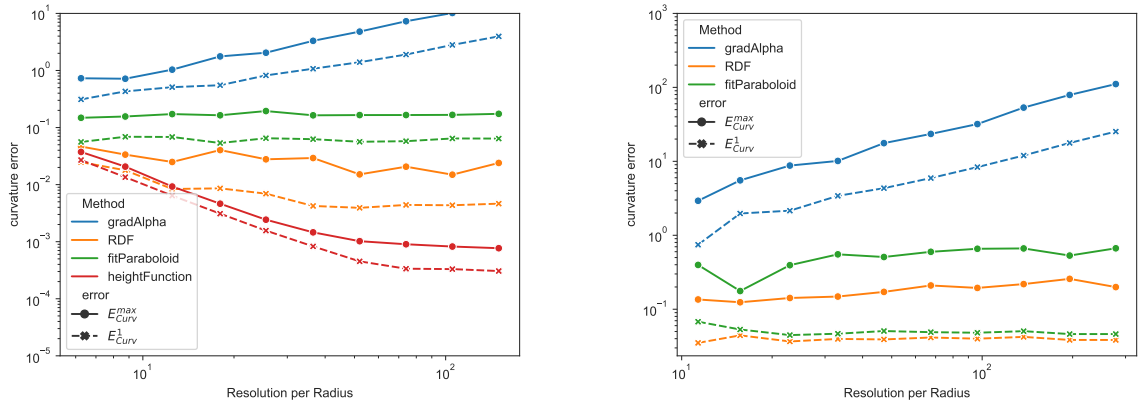
$$T = T_\infty - 2\beta_S^2 \frac{\rho^v (L + (c_p^l - c_p^v)(T_\infty - T_{Sat}))}{\rho^l c_p^l} \times \int_{1-R/r}^1 e^{-\beta_S^2\left((1-\xi)^{-2} - 2\left(1-\frac{\rho^v}{\rho^l}\right)\xi - 1\right)} d\xi \tag{40}$$

Results are shown in Fig. 14. As for the sucking interface problem, the gradient-based schemes deliver the most accurate result, followed by the Schrage results, which behave similarly to the sucking interface. In contrast to the previous two test cases, the combination of geometric VOF and phase change model is slightly more accurate compared to the MULES advection scheme.

4.2. **Surface tension models.**
For the validation of the surface tension models, multiple test cases, ranging from static reconstruction test cases to moving interface cases with non-constant curvature, are available. As in the previous examples, different curvature models can be selected by changing the entry in the `transportProperties` or `thermophysicalProperties` depending on the solver:

```
surfaceForces
{
    sigma 0.01; // surface tension in SI units
    // options: gradAlpha RDF heightFunction fitParaboloid
    curvatureModel gradAlpha;
    accelerationModel gravity;
    deltaFunctionModel alphaCSF;
}
```

The most challenging part in the simulation of surface tension is an accurate description of the pressure jump at the interface. The proposed models simulate the pressure jump by applying forces proportional to the local curvature at the mesh face centres. OpenFOAM uses the well-balanced surface tension formulation [30], resulting in an accuracy of machine precision if the correct curvature is specified. In other words, the accuracy of the curvature calculation is the main limiting factor for the accuracy of the



(a) Hexahedral grid. **(b)** Triangular prism grid.

Figure 15. Comparision of the curvature error for a circle for structured and unstructured grids.

surface force model. However, this is non-trivial as the curvature is proportional to the second derivative of a scalar field leading to small errors in the scalar field resulting in large errors in the curvature computation. We use two curvature error measures: The average error,

$$E^{1}_{\mathrm{Curv}} = \frac{1}{\kappa_{\mathrm{exact}}} \sum_{i}^{N_{\mathrm{ic}}} \frac{|\kappa_i - \kappa_{\mathrm{exact}}|}{N_{\mathrm{ic}}},$$

and the maximum error,

$$E^{\mathrm{max}}_{\mathrm{Curv}} = \frac{1}{\kappa_{\mathrm{exact}}} \max_i(|\kappa_i - \kappa_{\mathrm{exact}}|),$$

where the sum and max are over all the N_{ic} interface cells. There are four models available for curvature computation, but it is worth noting that HFM operates exclusively on structured grids.

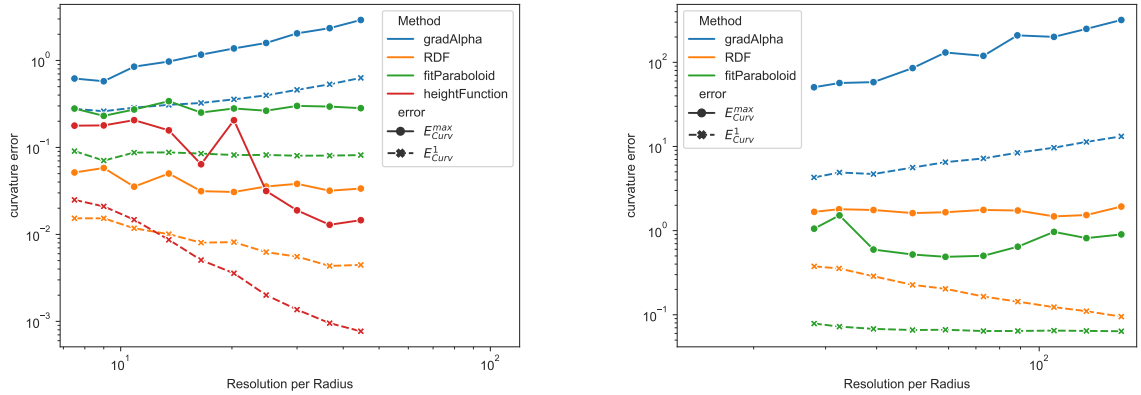
The first test case is the static reconstruction of a circle. For this, a domain with dimensions 2m×2m is created and a circular liquid region of radius 0.5 m is initialised. To test the proposed models, the cell type and resolution of the mesh are varied. The results are shown in Fig. 15. The difficulty in modeling surface tension is clearly seen by the five orders of magnitude in difference between the most accurate model, `HFM`, and the least accurate model, `gradAlpha`, in Fig. 15a. In this test, only the HFM model shows a converging behavior which flattens with refinement. This can be explained by the imperfect initialisation of the circle, as also mentioned by Coquerelle and Glockner [31]. The `fitParaboloid` and `RDF` models show zero-order convergence, which is an improvement in curvature computation of up to 1 to 2 orders of magnitude compared to the current standard model, `gradAlpha`. Unfortunately, applying the `HFM` on unstructured grids is not possible, which is why only three models can be compared on the triangular prism mesh shown in Fig. 15b. As on structured grids, the new models are able to achieve up to 1 to 2 orders of magnitude more accurate results.

The reconstruction of a sphere of radius 1m on a 2m×2m domain gives a similar picture as shown in Fig. 16. Again, the HFM is the most accurate method, followed by `RDF`, `fitParaboloid` and finally `gradAlpha`.

4.2.1. *Curvature of a disc for various contact angles.*
In this test case, the accuracy of the curvature computation with the presence of a boundary is simulated. To test the implementation, a 4m×2m domain is created and multiple circles cutting the domain boundary with angles 15, 30 , 45 ,60 and 75 degrees are initialised. As in the reconstruction test case, the resolution and grid type are varied in addition to the contact angle at the boundary.

This test case quantifies the accuracy of the contact angle implementation. The curvature error (see Figs. 17a and 17b) shows a diverging behaviour as in the previous test case with the maximum error being slightly larger for smaller contact angles. This increase in error for lower contact angles is most pronounced for the RDF and fitParaboloid model as depicted in Figs. 17c to 17f.

4.2.2. *Translating circle.*
So far, we have tested the capabilities of the different curvature models for a static interface configuration. The next step is to test them in combination with a flow solver, including the errors in pressure, velocity



(a) Hexahedral grid. **(b)** Tetrahedral grid.

Figure 16. Comparison of the curvature error for a sphere for structured and unstructured grids.

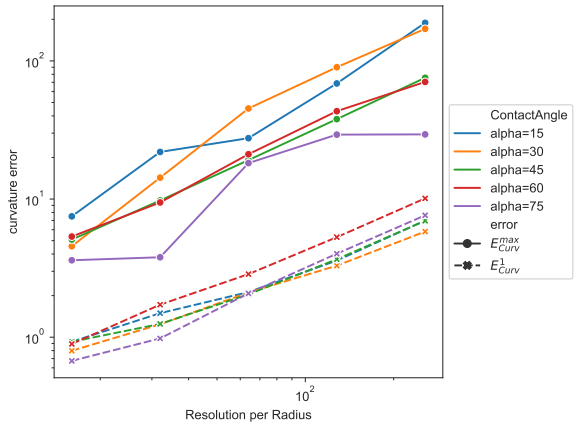
and advection. The most basic test case is the pure advection of a circle in a spatially and temporally constant flow. The circle moves with the same velocity as the surrounding gas, but the pressure inside the bubble is increased due to the Young-Laplace law. The channel has dimensions 4 m × 1 m with a background velocity of 1 m/s. The gas and liquid are assumed to have identical density and kinematic viscosity, 1000 kg/m^3 and $3.333 \cdot 10^{-5}$ m/s^2, respectively. We compare the time-averaged deviation of curvature and the maximal curvature error for hexahedral and triangular prism grids with different resolution using the proposed methods. Figure 18a shows the two curvature error measures as functions of grid resolution. As in the static reconstruction test case, the most accurate results are achieved by the HFM, but the maximum error does not seem to converge, which was also observed by Popinet [10]. As in the previous test case, the implemented models are significantly more accurate than the standard curvature model `gradAlpha`. On unstructured meshes, we see the same trend, that `fitParaboloid` and in particular `RDF` are significantly more accurate than the standard OpenFOAM model.

4.2.3. *Sine wave.*
The next step is the verification of the models with the analytical solution of Prosperetti [32]. He found an analytical solution for the movement of a cosine wave, including the effect of viscosity and surface tension. The main differences to the previous test case are that it includes the effect of the boundaries and that the curvature is not constant over the surface. Our domain and fluid properties are identical to the ones proposed by Popinet [10]. As in the previous benchmarks, the models are compared for different grid types, grid resolutions and interface advection methods. Figure 19a shows the evolution of the maximum height of the sine wave for the structured grids in combination with the geometric VOF method. The `RDF` and `fitParaboloid` methods are able to capture the amplitude, frequency and damping with good accuracy. Only the `gradAlpha` method deviates substantially from the analytical solution. The `HFM` method is not shown since it is not able to handle contact angles in our current implementation. In Fig. 19b the geometric VOF method is replaced by MULES. We observed that only the `RDF` method is capable of accurately representing the analytical function. The comparison of both advection methods with the available surface tension models reveals that the geometric VOF method is more accurate with the implemented surface tension methods.

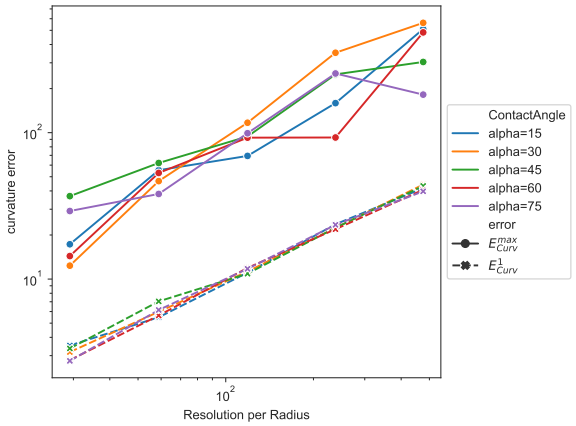
The combination of geometric VOF also shows good accuracy on unstructured grids, which is shown in Fig. 20. The `gradAlpha` and `fitParaboloid` methods show a significantly reduced accuracy compared to the structured grid. However, the `fitParaboloid` method at least qualitatively shows behaviour similar to the analytical solution in contrast to `gradAlpha`.

5. Conclusion

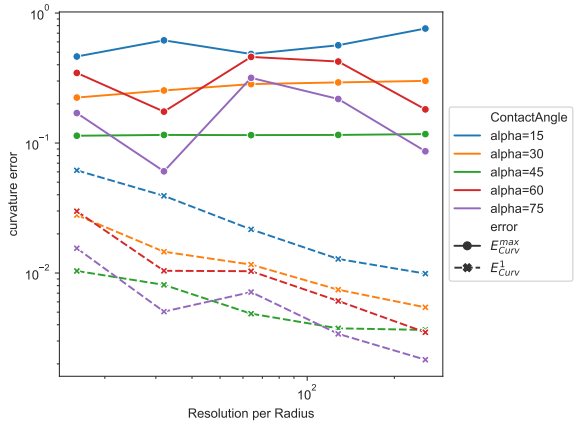
The OpenFOAM based code framework, TwoPhaseFlow, offers new phase change and surface tension models. The phase change module offers three phase change models denoted `explicitGrad`, `implicitGrad` and `Schrage`. The implementations have been validated using simple analytical benchmark cases from the literature. Currently, three surface tension models are available: The Height Function Method, the Reconstructed Distance Function method and the `fitParaboloid` method. These new models are



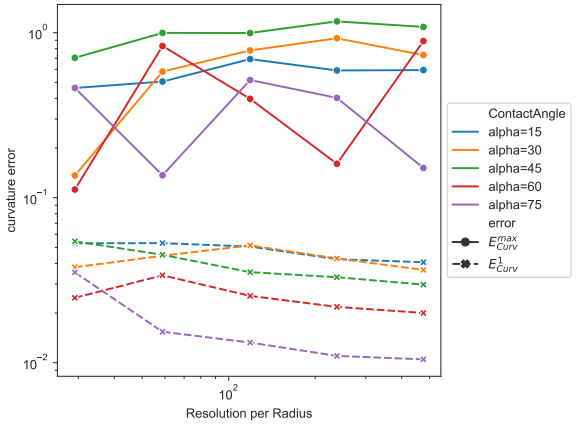
(a) gradAlpha - hexahedral grid.



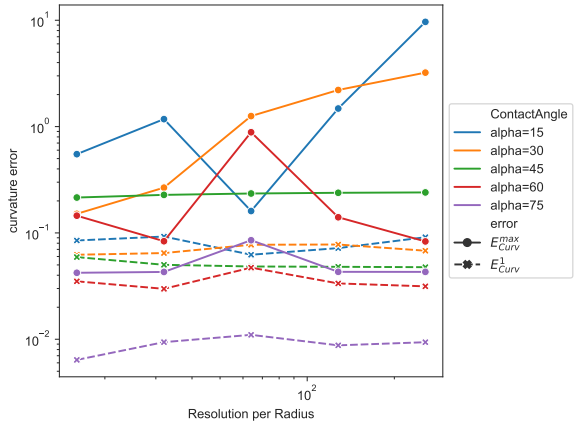
(b) gradAlpha - triangular prism grid.



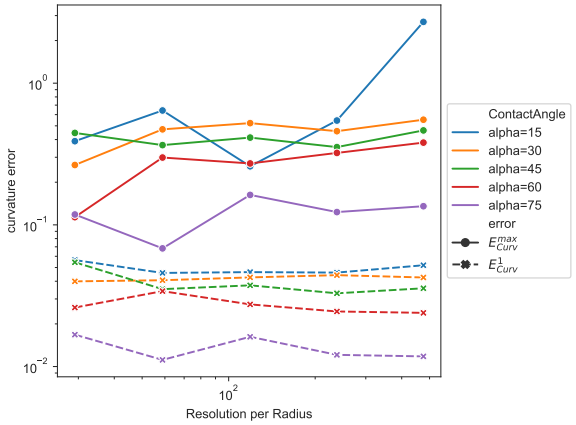
(c) RDF - hexahedral grid.



(d) RDF - triangular prism grid.



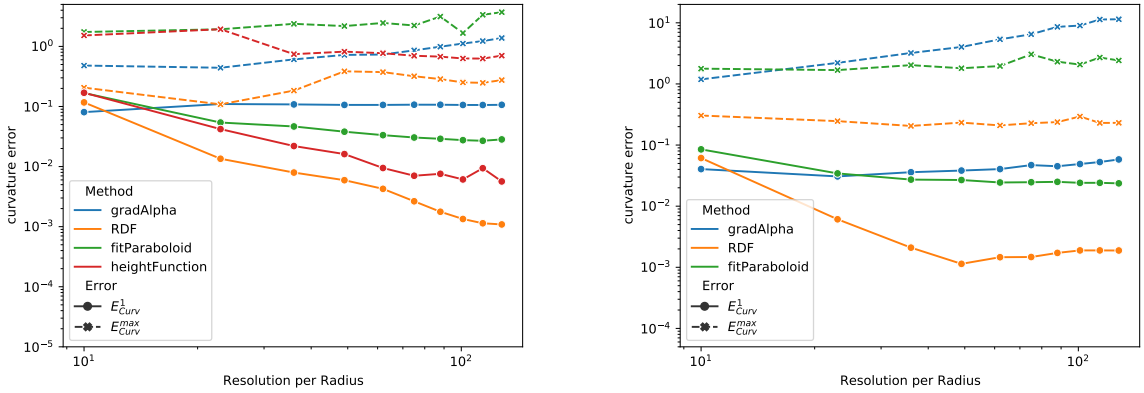
(e) fitParaboloid - hexahedral grid.



(f) fitParaboloid - triangular prism grid.

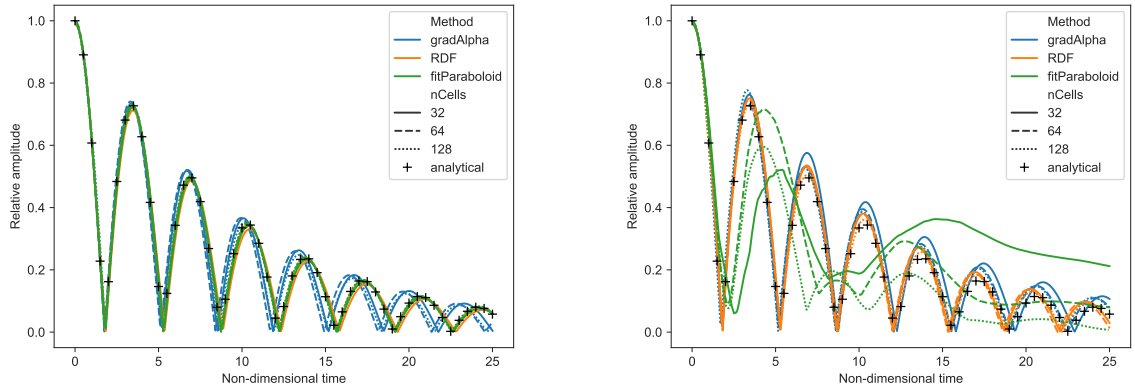
Figure 17. Comparison of the curvature error for different contact angles for structured and unstructured grids.

validated with surface tension benchmarks and show a reduction of the spurious velocities by more than an order of magnitude, depending on the model choice. The library offers three solvers of which the most general is able to simulate compressible two phase flow, including the effects of the phase change and surface tension. All models work with both the isoAdvector geometric VOF method as well as the MULES method for interface advection. The TwoPhaseFlow framework exploits the runtime selection mechanism of OpenFOAM, allowing easy implementation and verification of new models. The framework is released under the GPL v3 and the source code is publicly-available in a software repository [33].



(a) Hexahedral grids. **(b)** Tetrahedral grids.

Figure 18. Advected circle: Comparision of the curvature error for structured and unstructured grids.



(a) Geometric VOF (isoAdector). **(b)** Algebraic VOF (MULES).

Figure 19. Oscillating sine wave: Comparison of the interface position on hexahedral grids.

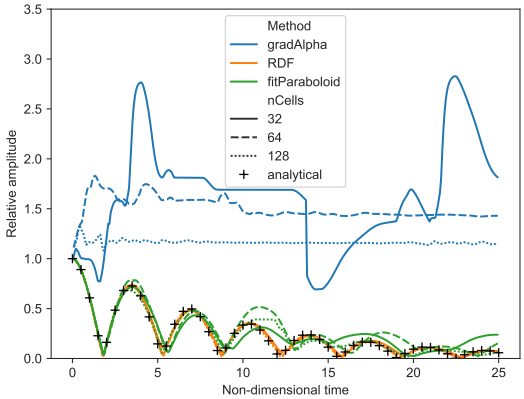


Figure 20. Oscillating sine wave: Comparison of the interface position on triangular prism grids.

Acknowledgement

This work was supported by German Aerospace Center - DLR. JR acknowledges support from Independent Research Fund Denmark (Grant-ID: 9063-00018B). The authors would like to thank Grega Belsak, Lionel Gamet and Christoph Wilms for testing the library and reporting numerous bugs.

Author Contributions: Methodology, H.S. and J.R.; software, H.S.; validation, H.S.; writing—original draft preparation, H.S. and J.R.; writing—review and editing, H.S. and J.R.; All authors have read and agreed to the published version of the manuscript.

Appendix A. Thermodynamic model

In this paper only phase change models caused by temperature gradients are implemented but for numerous engineering application concentration based phase change is crucial. In this framework we try to offer a possibility to simplify the implementation of the concentration based phase change. The solver `multiRegionPhaseChangeFlow` is able to account for mixture of different species in each phase by utilising a thermodynamic framework similar to the one found in `icoReactingMultiphaseInterFoam`. The activation of multiple components in phase is done by switching a keyword:

```
phases (liquid gas);
liquid
{
    type purePhaseModel;
}
gas
{
    type multiComponentPhaseModel;
    Sc              0.7;
    residualAlpha   1e-3;
}
```

The `multiComponentPhaseModel` is based on OpenFOAM's `rhoReactionThermo` module and therefore opens the possibility of including reactions in the future as well. In the current state of TwoPhaseFlow, the concentration fields only affect the phase density and do not accounted for phase change at the interface.

References

[1] J. Roenby, H. Bredmose, and H. Jasak, "A computational method for sharp interface advection," *Royal Society Open Science*, vol. 3, no. 11, p. 160405, 2016.

[2] H. Scheufler and J. Roenby, "Accurate and efficient surface reconstruction from volume fraction data on general meshes," *Journal of Computational Physics*, vol. 383, pp. 1–23, 2019.

[3] S. Hardt and F. Wondra, "Evaporation model for interfacial flows based on a continuum-field representation of the source terms," *Journal of Computational Physics*, vol. 227, no. 11, pp. 5871–5895, 2008.

[4] M. Nabil and A. S. Rattner, "interThermalPhaseChangeFoam—a framework for two-phase flow simulations with thermally driven phase change," *SoftwareX*, vol. 5, pp. 216–226, 2016.

[5] C. Kunkelmann, "Numerical modeling and investigation of boiling phenomena," Ph.D. dissertation, Technische Universität, Darmstadt, May 2011. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/2731/

[6] S. Batzdorf, "Heat transfer and evaporation during single drop impingement onto a superheated wall," Ph.D. dissertation, Technische Universität, Darmstadt, 2015. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/4542/

[7] Y. Sato and B. Ničeno, "A sharp-interface phase change model for a mass-conservative interface tracking method," *Journal of Computational Physics*, vol. 249, pp. 127–161, 2013.

[8] M. M. Francois, S. J. Cummins, E. D. Dendy, D. B. Kothe, J. M. Sicilian, and M. W. Williams, "A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework," *Journal of Computational Physics*, vol. 213, no. 1, pp. 141–173, 2006.

[9] Brackbill, Jeremiah U., Douglas B. Kothe, and Charles Zemach, "A continuum method for modeling surface tension," *Journal of Computational Physics*, vol. 100, no. 2, pp. 335–354, 1992.

[10] S. Popinet, "An accurate adaptive solver for surface-tension-driven interfacial flows," *Journal of Computational Physics*, vol. 228, no. 16, pp. 5838–5866, 2009.

[11] S. J. Cummins, M. M. Francois, and D. B. Kothe, "Estimating curvature from volume fractions," *Computers & Structures*, vol. 83, no. 6-7, pp. 425–434, 2005.

[12] T. Abadie, J. Aubin, and D. Legendre, "On the combined effects of surface tension force calculation and interface advection on spurious currents within volume of fluid and level set frameworks," *Journal of Computational Physics*, vol. 297, pp. 611–636, 2015.

[13] P. A. Wroniszewski, J. C. Verschaeve, and G. K. Pedersen, "Benchmarking of navier–stokes codes for free surface simulations by means of a solitary wave," *Coastal Engineering*, vol. 91, pp. 1–17, 2014.

[14] M. Jadidi, M. Tembely, S. Moghtadernejad, and A. Dolatabadi, "A coupled level set and volume of fluid method with application to compressible two-phase flow," in *Proceedings of the 22nd Annual Conference of the CFD Society of Canada, Toronto, ON, Canada*, 2014, pp. 1–4.

[15] S. Westermaier, W. Kowalczyk *et al.*, "Implementation of non-newtonian fluid properties for compressible multiphase flows in OpenFOAM," *Open Journal of Fluid Dynamics*, vol. 10, no. 02, p. 135, 2020.

[16] M. Koch, C. Lechner, F. Reuter, K. Köhler, R. Mettin, and W. Lauterborn, "Numerical modeling of laser generated cavitation bubbles with the finite volume and volume of fluid method, using OpenFOAM," *Computers & Fluids*, vol. 126, pp. 71–90, 2016.

[17] S. T. Miller, H. Jasak, D. A. Boger, E. G. Paterson, and A. Nedungadi, "A pressure-based, compressible, two-phase flow finite volume method for underwater explosions," *Computers & Fluids*, vol. 87, pp. 132–143, 2013.

[18] J. Roenby, H. Bredmose, and H. Jasak, "IsoAdvector: Geometric VOF on General Meshes," in *OpenFOAM® – Selected Papers of the 11th Workshop*, J. M. Nóbrega and H. Jasak, Eds. Springer International Publishing, 2019, pp. 281–296.

[19] S. Popinet, "Numerical models of surface tension," *Annual Review of Fluid Mechanics*, vol. 50, no. 1, 2017.

[20] J.-M. Ghidaglia, "Capillary forces: A volume formulation," *European Journal of Mechanics - B/Fluids*, vol. 59, pp. 86–89, 2016.

[21] A. Syrakos, S. Varchanis, Y. Dimakopoulos, A. Goulas, and J. Tsamopoulos, "A critical analysis of some popular methods for the discretisation of the gradient operator in finite volume methods," *Physics of Fluids*, vol. 29, no. 12, p. 127103, 2017.

[22] "Basilisk," http://basilisk.fr/, 2020.

[23] M. Owkes and O. Desjardins, "A mesh-decoupled height function method for computing interface curvature," *Journal of Computational Physics*, vol. 281, pp. 285–300, 2015.

[24] F. Evrard, F. Denner, and B. van Wachem, "Estimation of curvature from volume fractions using parabolic reconstruction on two-dimensional unstructured meshes," *Journal of Computational Physics*, vol. 351, pp. 271–294, 2017.

[25] C. Fields, "cfmesh," https://www.openfoam.com/releases/openfoam-v1712/pre-processing.php, 2017.

[26] OpenCFD, "Mesh generation with the snappyhexmesh utility," https://www.openfoam.com/documentation/user-guide/snappyHexMesh.php, 2020.

[27] H. Scheufler and J. Pearson, "casefoam: An OpenFOAM case manipulatior and creator," https://github.com/DLR-RY/caseFoam, 2019.

[28] S. W. Welch and J. Wilson, "A volume of fluid based method for fluid flows with phase change," *Journal of Computational Physics*, vol. 160, no. 2, pp. 662–682, 2000.

[29] L. E. Scriven, "On the dynamics of phase growth," *Chemical Engineering Science*, vol. 10, no. 1-2, pp. 1–13, 1959.

[30] S. S. Deshpande, L. Anumolu, and M. F. Trujillo, "Evaluating the performance of the two-phase flow solver interfoam," *Computational science & discovery*, vol. 5, no. 1, p. 014016, 2012.

[31] M. Coquerelle and S. Glockner, "A fourth-order accurate curvature computation in a level set framework for two-phase flows subjected to surface tension forces," *Journal of Computational Physics*, vol. 305, pp. 838–876, 2016.

[32] A. Prosperetti, "Motion of two superposed viscous fluids," *Physics of Fluids*, vol. 24, no. 7, p. 1217, 1981.

[33] "TwoPhaseFlow," https://github.com/DLR-RY/TwoPhaseFlow, 2019.