Journal of Computational Physics 270 (2014) 587–612

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

# A computational framework for conservative, three-dimensional, unsplit, geometric transport with application to the volume-of-fluid (VOF) method

Mark Owkes *, Olivier Desjardins

*Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA*

ARTICLE INFO

*Article history:*
Received 17 May 2013
Received in revised form 3 April 2014
Accepted 7 April 2014
Available online 15 April 2014

*Keywords:*
Multiphase flow
Geometric transport
Volume-of-fluid
Incompressible flow
Mass conservation
Semi-Lagrangian flux

ABSTRACT

In this work, a novel computational framework for calculating convection fluxes is developed and employed in the context of the piecewise linear interface calculation (PLIC) volume-of-fluid (VOF) method. The scheme is three-dimensional, unsplit, discretely conservative, and bounded. The scheme leverages the idea of semi-Lagrangian transport to estimate the amount of liquid that is fluxed through each face during a time-step.

The present work can be seen as an extension of the two-dimensional EMFPA method of López et al. (2004) [16] to three dimensions and an improvement of the three-dimensional FMFPA-3D method of Hernández et al. (2008) [17] with the addition of discrete conservation. In FMFPA-3D, fluxes of liquid volume fraction are calculated by transporting a cell face back in time with a semi-Lagrangian method that uses cell face edge velocities to produce a flux hexahedron with flat faces. The flux hexahedron may overlap with neighboring fluxes hindering the conservation properties of the method. The proposed method computes the fluxes by transporting the cell face back in time using a semi-Lagrangian step based on the cell face corner velocities, which results in a three-dimensional, generalized flux hexahedron that does not typically have flat faces. However, the flux volumes do not overlap and discrete conservation can be achieved. The complex flux volume is partitioned into a collection of simplices and a simple sign convention allows the calculation of the flux to be reduced to a straightforward and systematic algorithm. The proposed VOF scheme is tested on multiple benchmark cases including Zalesak's disk, two- and three-dimensional deformation tests, and the evolution of a droplet in homogeneous isotropic turbulence.



## 1. Introduction

In simulations of multiphase flows, an accurate representation of interface motion is important for many interesting engineering applications with significant interface topology changes. Large discontinuities can exist at the interface, including large jumps in pressure, density, and viscosity. Therefore, various numerical schemes have been developed to track the interface location. These schemes can be broadly classified as interface tracking and interface capturing. Interface tracking schemes represent the interface explicitly using, for example, a mesh that deforms with the interface [1] or marker particles on the interface [2]. When the interface undergoes significant deformation and breaking or merging events, interface

* Corresponding author.
*E-mail address:* mfc86@cornell.edu (M. Owkes).

http://dx.doi.org/10.1016/j.jcp.2014.04.022
0021-9991/

tracking schemes suffer from the need to frequently perform re-meshing or re-seeding of marker particles. Interface capturing schemes implicitly represent the interface and include level set methods [3] and volume-of-fluid (VOF) schemes [4]. Level set methods can be very accurate but suffer from the lack of discrete mass conservation. The conservation properties were improved through the development of the conservative level set [5–8], but the method still lacks discrete conservation. VOF methods have the potential to provide discrete mass conservation and second-order accuracy, and are the basis for this work.

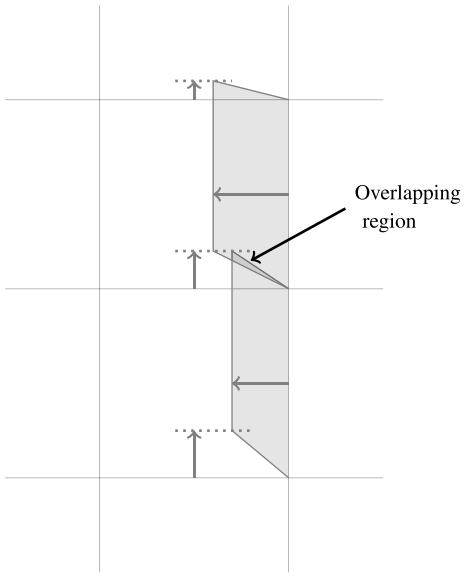
VOF methods track the interface by storing the ratio of liquid volume to cell volume for each computational cell, known as the liquid volume fraction. VOF methods were introduced in the early 1970s when DeBar [9], Nichols and Hirt [10], and Noh and Woodward [11] all developed variants of VOF within a short period of time. Many advancements have improved VOF schemes since their inception through improved representations of the liquid within the domain and improved advection schemes. Tryggvason et al. [12] provide a detailed history of VOF methods with descriptions of many of the significant contributions. In piecewise linear interface calculation (PLIC) methods, the interface is approximated within each cell using a straight line (2D) or plane (3D) [9,13,14]. PLIC methods mainly differ in the algorithm used to orient the linear function through the calculation of the interface normal vector. In this work we use the PLIC interface representation with an interface normal computed using the ELVIRA method [15], although the methodology can readily be used with other interface normal calculation strategies.

VOF schemes also differ in how the liquid volume fraction is transported. Early VOF schemes used flux splitting, wherein the multidimensional transport step is replaced by successive one-dimensional transport steps [4]. This approach, which is straightforward to implement, suffers from errors due to the flux splitting step. Alternatively, unsplit schemes that avoid this source of error have been constructed. Pilliod and Puckett [15] developed a second-order unsplit transport algorithm by computing fluxes by integrating over volumes formed by characteristics in space–time. The method is shown to produce superior results to split methods, however the extension to three dimensions is not provided.

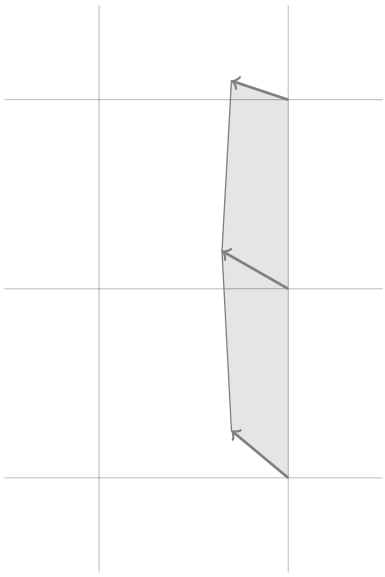
Geometric unsplit transport schemes provide a framework for constructing fluxes that are consistent with the characteristics of the problem and are used in this work. Rider and Kothe [14] proposed an unsplit geometric advection scheme that uses trapezoidal flux regions constructed using cell face velocities as shown in Fig. 1(a). In general, neighboring faces will have different velocities, which results in regions that overlap and are fluxed twice. López et al. [16] developed EMFPA as an improvement to Rider and Kothe's method. EMFPA uses cell vertex velocities to create the flux region and avoids any overlapping regions as shown in Fig. 1(b). The present work is an extension of EMFPA to three dimensions and the two schemes produce very similar results in two dimensions. Note that EMFPA is more straightforward to implement than the proposed scheme if the reader is strictly interested in two dimensional problems. Hernández et al. [17] developed a three-dimensional flux calculation method known as FMFPA-3D that is based on the velocity on each edge of the cell face as shown in Fig. 1(c). The resulting fluxes can have overlapping regions that hinder the conservation properties of the scheme. The proposed method constructs three-dimensional flux regions using cell corner velocities as shown in Fig. 1(d). The resulting flux volumes do not overlap and provide a framework for unsplit, conservative, bounded, three-dimensional transport. The approach is complicated due to the presence of non-flat faces on the flux region that are produced when the corner vertex velocity vectors do not lie in the same plane. Furthermore, the flux volumes can become crossed [18]. However, a straightforward, systematic approach is presented in this work to deal with the complex geometry.

The proposed VOF scheme shares similarities with other geometric VOF methods. The HyLEM method of Le Chenadec and Pitsch [19] is a semi-Lagrangian transport scheme that updates the liquid volume fraction by projecting the cell forward and backward in time. HyLEM is not flux-based and is not discretely conservative. The proposed scheme can be seen as a flux based version of HyLEM, wherein the fluxes have been modified to ensure discrete conservation. If the fluxes are not modified, the two schemes are equivalent [20]. The voFoam method of Marić et al. [21] is a three-dimensional, unsplit, conservative flux based VOF scheme that is similar to the proposed method. voFoam is shown to work well, however it is unclear how crossed fluxes, wherein the flux moves liquid into and out of a cell through the same face, are dealt with. In three dimensions it is not obvious how to deal with the complex geometries that arise in such cases. Furthermore, it is unclear how the flux volumes are rescaled to construct conservative fluxes without introducing overlapping regions between neighboring flux volumes. The EMFPA-3D of Ivey and Moin [22] is similar to voFoam. Details are not provided on how crossed flux volumes are handled. Unfortunately, their approach to form conservative flux volumes introduces overlapping regions. Furthermore, their study fails to provide verification or validation test cases for the method.

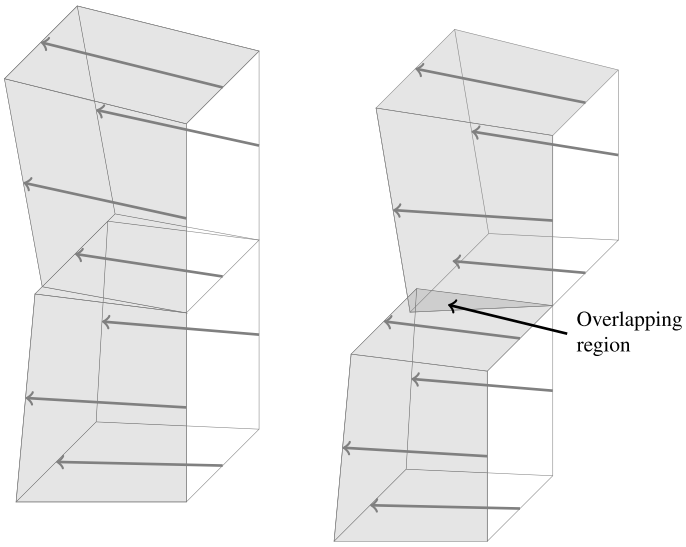
In the proposed scheme, purely geometrical operations are used to calculate the fluxes. The flux geometry is systematically partitioned into a collection of simplices (triangles or tetrahedra in two or three dimensions, respectively). Simplices are easier to work with computationally and, when coupled with an appropriate sign convention, lead to a straightforward scheme that relies on a small number of geometric routines. The approach naturally handles crossed flux volumes. The method uses a correction to the flux volumes to ensure discrete conservation that does not introduce overlapping regions between neighboring flux volumes. The correction is performed using an analytic expression that avoids the need for an iterative procedure. With this framework, conservative unsplit transport can be performed. In this paper, we apply this transport methodology to the VOF interface capturing strategy. The proposed method could be used to transport other quantities. For example, both momentum and the gas–liquid interface could be transported, forming a method similar to the scheme of Le Chenadec and Pitsch [20] but with the addition of discrete conservation of mass and momentum. This scheme will be considered in a future publication.



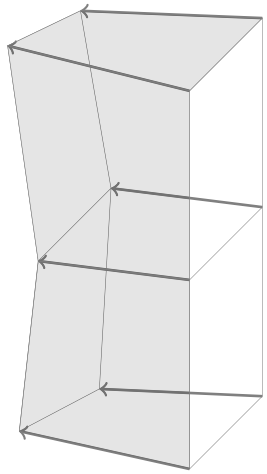

(a) 2D fluxes based on face velocities, Rider and Kothe [14]



(b) 2D fluxes based on vertex velocities, López et al. [16]




(c) 3D fluxes formed using planes based on edge velocities, Hernández et al. [17]



(d) 3D fluxes based on vertex velocities, proposed scheme

**Fig. 1.** Example of methods used to compute geometric fluxes. Velocity vectors used to construct fluxes are shown with arrows. Fluxes are shaded.

This paper begins with a detailed mathematical derivation of the method in Section 2. Section 3 provides a description of the computational building blocks that are used in the method. The section includes details on the reconstruction of the interface from the liquid volume fraction, the construction and discrete representation of the flux volumes, and the calculation of the fluxes. Canonical test cases including Zalesak's disk, two- and three-dimensional deformation tests, and the time evolution of a drop in synthetic homogeneous isotropic turbulence are presented in Section 4. Finally, conclusions are drawn in Section 5.

## 2. Mathematical formulation

In this section, a detailed derivation of the conservation laws applied to a fixed control volume is presented. The derivation recasts a scalar advection partial differential equation (PDE) into a relation for the evolution of a scalar in a fixed control volume due to geometric fluxes. While the resulting equations, Eq. (13) for a general advected function and Eq. (18) for the liquid volume fraction, are similar to previously published unsplit geometric transport advection equations, see for

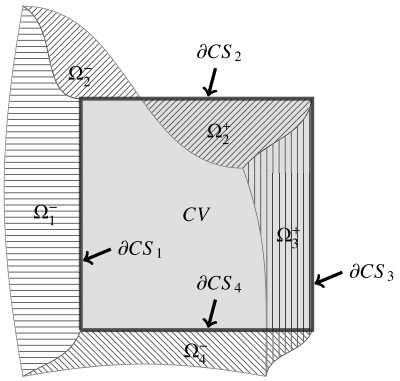



**Fig. 2.** Example geometry used to construct the flux volumes. The control volume $CV$ is shown as the shaded region. The bounding surface of $CV$ has been partitioned into four sub-surfaces $\partial CS_i$ with $i = 1, \ldots, 4$. Associated with each sub-surface is the flux volume $\Omega_i$. Each flux volume is indicated with a different pattern of lines.

example [14,16,17,21,23,24], this derivation is useful in that it shows these are exact relations. The derivation highlights that exact fluxes to advect a function from time $t^n$ to time $t^{n+1}$ can be computed by constructing streak-tubes emitted from each face of the control volume and evaluating the advected function within the streak-tube at time $t^n$.

### 2.1. Problem setup and notations

The material evolution of a conserved scalar $f(\boldsymbol{x}, t)$ in a solenoidal velocity field is described by

$$\frac{\partial f}{\partial t} + \nabla \cdot (\boldsymbol{u} f) = 0, \tag{1}$$

where $\boldsymbol{x}$ is the spatial coordinate, $t$ is time, and $\boldsymbol{u}$ is the velocity field that is assumed to be known. Integrating this equation over a discrete time-step $\Delta t = t^{n+1} - t^n$ and fixed control volume $CV$ (e.g., a computational cell) with bounding surface $CS$ and using Gauss' theorem on the second term allows us to write

$$\int_{CV} \left( f\left(\boldsymbol{x}, t^{n+1}\right) - f\left(\boldsymbol{x}, t^n\right) \right) \mathrm{d}V + \int_{t^n}^{t^{n+1}} \oint_{CS} f \boldsymbol{u} \cdot \boldsymbol{n}_{CV} \, \mathrm{d}S \, \mathrm{d}t = 0. \tag{2}$$

The term on the right is the flux through the surface of the control volume and depends on $f(\boldsymbol{x}, t)$ throughout the time-step, which is not usually known. Typically, a discrete representation of $f(\boldsymbol{x}, t^n)$ is known and an update equation is used to calculate $f(\boldsymbol{x}, t^{n+1})$. Therefore, we recast the flux so it depends solely on $f(\boldsymbol{x}, t^n)$. To do this we start by partitioning the surface of the control volume $CS$ into sub-surfaces $\partial CS_i$ (e.g., the faces of our computational cell) such that

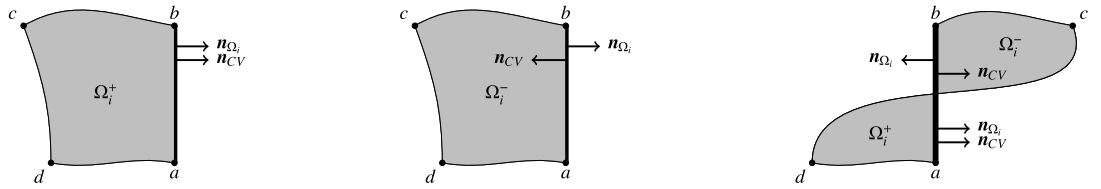
$$\begin{aligned} &CS = \bigcup_{i=1}^{N_S} \partial CS_i \quad \text{and} \\ &\partial CS_i \cap \partial CS_j = \emptyset \quad \text{for } i \text{ and } j \in \{1, \ldots, N_S\} \text{ and } i \neq j. \end{aligned} \tag{3}$$

To each sub-surface $\partial CS_i$ we associate the flux volume $\Omega_i(t)$ with bounding surface $\omega_i(t)$. $\Omega_i(t)$ is the signed volume that flows through the sub-surface $\partial CS_i$ between time $t$ and $t^{n+1}$. Hence, $\Omega_i(t^{n+1})$ is zero by definition and $\Omega_i(t^n)$ is the volume that flows through $\partial CS_i$ during $\Delta t$. Fig. 2 shows an example control volume with surface $CS$ that has been partitioned with four sub-surfaces $\partial CS_i$ for $i = 1, \ldots, 4$. The flux volume associated with each sub-surface is shown. The sign of the volume is defined to be negative if the volume moves through the sub-surface and into the $CV$ during the step and positive if the volume moves through the sub-surface and out of the $CV$. Formal definitions of the flux volumes and signs are provided in the derivation below.

Integrating Eq. (1) over the flux volume $\Omega_i$ and using Gauss' theorem on the second term allows us to write

$$\int_{\Omega_i(t)} \frac{\partial f}{\partial t} \, \mathrm{d}V + \oint_{\omega_i(t)} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S = 0. \tag{4}$$

In this equation, $\boldsymbol{n}_{\Omega_i}$ is the outward-pointing normal to $\Omega_i(t)$. We define $\omega_i$ such that part of it coincides with $\partial CS_i$, which is fixed in time. The rest of $\omega_i$ is defined to have a zero flux of $f$ and thus must be a material surface that moves with the flow. Therefore, $\omega_i$ can be partitioned into two sub-regions, namely $\omega_{i,F} = \omega_i \cap \partial CS_i = \partial CS_i$ that is fixed and $\omega_{i,M} = \omega_i \setminus \partial CS_i$ that is a material surface. Integrating the previous equation over $\Delta t$ and using this partition, we can write




(a) Simple example with positive flux volume, $\Omega_i^- = \emptyset$.

(b) Simple example with negative flux volume, $\Omega_i^+ = \emptyset$.

(c) Example with crossed flux volume with both positive and negative regions.

**Fig. 3.** Example of the geometry used to define the flux volume $\Omega_i$ (shaded region) with outward-facing normal $\boldsymbol{n}_{\Omega_i}$. The flux volume is associated with the sub-surface $\partial CS_i$ shown with the thick line with vertices $a$ and $b$. $\boldsymbol{n}_{CV}$ the outward-facing normal to $CV$ is also shown.

$$\int_{t^n}^{t^{n+1}} \int_{\Omega_i(t)} \frac{\partial f}{\partial t} \, \mathrm{d}V \, \mathrm{d}t + \int_{t^n}^{t^{n+1}} \int_{\omega_{i,M}(t)} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S \, \mathrm{d}t + \int_{t^n}^{t^{n+1}} \int_{\partial CS_i} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S \, \mathrm{d}t = 0. \tag{5}$$

The last term is very similar to the flux term in Eq. (2) and will provide the connection between the two equations. However, the first two terms in this equation are difficult to deal with in their current form but can be elucidated using Leibniz's rule which states

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega_i(t)} f \, \mathrm{d}V = \int_{\Omega_i(t)} \frac{\partial f}{\partial t} \, \mathrm{d}V + \int_{\omega_{i,M}(t)} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S, \tag{6}$$

where we have used $\omega_i = \omega_{i,M} \cup \omega_{i,F}$, the property $\boldsymbol{u}_s = 0$ on $\omega_{i,F}$, and defined $\boldsymbol{u}_s = \boldsymbol{u}$ on $\omega_{i,M}$, which makes $\omega_{i,M}$ a material surface. At this point, it should be clear that $\Omega_i(t)$ is a streak-tube emitted backward in time from the surface $\partial CS_i$ during the time period $t$ to $t^{n+1}$. Integrating the previous equation over the time-step results in

$$\int_{\Omega_i(t^{n+1})} f\left(\boldsymbol{x}, t^{n+1}\right) \mathrm{d}V - \int_{\Omega_i(t^n)} f\left(\boldsymbol{x}, t^n\right) \mathrm{d}V = \int_{t^n}^{t^{n+1}} \int_{\Omega_i(t)} \frac{\partial f}{\partial t} \, \mathrm{d}V \, \mathrm{d}t + \int_{t^n}^{t^{n+1}} \int_{\omega_{i,M}(t)} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S \, \mathrm{d}t. \tag{7}$$

By definition, $\Omega_i(t^{n+1})$ has a zero volume and the first term in the previous equation is zero. Henceforth, we will call $\Omega_i(t^n)$ the flux volume and adopt the notation $\Omega_i = \Omega_i(t^n)$. Subtracting Eq. (7) from Eq. (5) and using this notation leads to

$$\int_{t^n}^{t^{n+1}} \int_{\partial CS_i} f \boldsymbol{u} \cdot \boldsymbol{n}_{\Omega_i} \, \mathrm{d}S \, \mathrm{d}t = \int_{\Omega_i} f\left(\boldsymbol{x}, t^n\right) \mathrm{d}V, \tag{8}$$

which provides a simple relationship between the flux through the sub-surface $\partial CS_i$ and the volume integral over $\Omega_i$. The previous equation states that the flux of $f$ through $\partial CS_i$ during the time-step is equivalent to the integral of $f$ in the flux volume at the beginning of the time-step.

The previous equation can almost be combined with Eq. (2), leading to a useful time advancement equation. However, the flux term in Eq. (8) contains $\boldsymbol{n}_{\Omega_i}$, the outward-pointing normal to $\Omega_i$, while the normal in Eq. (2) is $\boldsymbol{n}_{CV}$, which is outward-pointing with respect to $CV$. Because the normal vectors are defined using the same surface $\partial CS_i$, they are either identical (i.e., $\boldsymbol{n}_{CV} \cdot \boldsymbol{n}_{\Omega_i} = +1$) or point in opposite directions (i.e., $\boldsymbol{n}_{CV} \cdot \boldsymbol{n}_{\Omega_i} = -1$). As a result, we partition the sub-surfaces $\partial CS_i$ into two regions $\partial CS_i^+$ and $\partial CS_i^-$ such that

$$\begin{aligned} &\partial CS_i = \partial CS_i^+ \cup \partial CS_i^- \quad \text{and} \\ &\partial CS_i^+ \cap \partial CS_i^- = \emptyset. \end{aligned} \tag{9}$$

The sub-surfaces are defined using

$$\partial CS_i^+ = \{\boldsymbol{x} \in \partial CS_i \mid \boldsymbol{n}_{CV}(\boldsymbol{x}) \cdot \boldsymbol{n}_{\Omega_i}(\boldsymbol{x}) = +1\} \tag{10}$$

and

$$\partial CS_i^- = \{\boldsymbol{x} \in \partial CS_i \mid \boldsymbol{n}_{CV}(\boldsymbol{x}) \cdot \boldsymbol{n}_{\Omega_i}(\boldsymbol{x}) = -1\}. \tag{11}$$

Furthermore, we associate with the sub-surfaces $\partial CS_i^+$ and $\partial CS_i^-$ the flux volumes $\Omega_i^+$ and $\Omega_i^-$, respectively. The flux volumes are defined using the same methodology as described previously, and can therefore be thought of as streak-tubes. Fig. 3 shows three example fluxes with positive, negative, and positive and negative flux volumes, respectively.

With these definitions, Eq. (8) can be written as

$$\int_{t^n}^{t^{n+1}} \int_{\partial CS_i} f\boldsymbol{u}\cdot\boldsymbol{n}_{CV}\,\mathrm{d}S\,\mathrm{d}t = \int_{\Omega_i^+} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V - \int_{\Omega_i^-} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V. \tag{12}$$

Finally, combining Eq. (2) with Eq. (12) leads to

$$\int_{CV}\left(f\left(\boldsymbol{x},t^{n+1}\right)-f\left(\boldsymbol{x},t^n\right)\right)\mathrm{d}V + \sum_{i=1}^{N_S}\left[\int_{\Omega_i^+} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V - \int_{\Omega_i^-} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V\right] = 0. \tag{13}$$

The term on the left of this equation describes the change in $f$ within the control volume $CV$ during the time-step. The change is due to fluxes into or out of the control volume, as described by the term on the right. The flux term has been recast into a volume integral over the flux volumes, $\Omega_i^+$ and $\Omega_i^-$, that move out of and into the control volume, respectively.

### 2.2. Flux velocity

Until now, we have only considered one control volume. However, the control volumes represent the computational cells used in a simulation, so it is necessary to extend the notation to describe a collection of control volumes. Let the number of control volumes used in the simulation be $N_{CV}$ and the $p$th control volume be denoted $CV_p$ with bounding surface $CS_p$, which has been partitioned into sub-surfaces $\partial CS_{p,i}$ for $i=1,\ldots,N_S$. Similarly, the flux volume associated with the sub-surface $\partial CS_{p,i}$ is indicated as $\Omega_{p,i}$.

We introduce useful quantities associated with $\partial CS_{p,i}$ and $\Omega_{p,i}$. We define the area $\mathcal{A}_{p,i}=\int_{\partial CS_{p,i}}\mathrm{d}S$ and the signed volume $\mathcal{V}_{p,i}=\int_{\Omega_{p,i}^+}\mathrm{d}V-\int_{\Omega_{p,i}^-}\mathrm{d}V$. From the area, volume, and the time interval $\Delta t$, a mean normal fluxing velocity $\mathcal{U}_{p,i}$ can be defined as

$$\mathcal{U}_{p,i}=\frac{\mathcal{V}_{p,i}}{\Delta t\mathcal{A}_{p,i}}. \tag{14}$$

To ensure discrete conservation, the flux velocities must respect the solenoidal condition $\sum_{i=1}^{N_s}\mathcal{U}_{p,i}\mathcal{A}_{p,i}=0$.

### 2.3. Liquid volume fraction transport

To examine liquid volume fraction transport, we choose $f$ to be the liquid distribution function defined as

$$f(\boldsymbol{x},t)=\begin{cases}1, & \text{if } \boldsymbol{x} \text{ is in the liquid at time } t,\\ 0, & \text{if } \boldsymbol{x} \text{ is in the gas at time } t,\end{cases} \tag{15}$$

and introduce the shorthand notations

$$\alpha_p(t)=\frac{1}{\mathcal{V}_p}\int_{CV_p} f(\boldsymbol{x},t)\,\mathrm{d}V \tag{16}$$

where $\mathcal{V}_p=\int_{CV_p}\mathrm{d}V$ and

$$\alpha_{p,i}=\frac{1}{\mathcal{V}_{p,i}}\left(\int_{\Omega_{p,i}^+} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V - \int_{\Omega_{p,i}^-} f\left(\boldsymbol{x},t^n\right)\mathrm{d}V\right). \tag{17}$$

Note that $\alpha_p$ and $\alpha_{p,i}$ are volume averaged quantities, more precisely they are the zeroth order moments of $f$ within the control volume $CV_p$ and the signed flux volume $\Omega_{p,i}$, respectively. $\alpha_p(t)$ is commonly referred to as the liquid volume fraction within the $p$th control volume. $\alpha_{p,i}$ is the liquid volume fraction within the flux volume $\Omega_{p,i}$.

With these notations, Eq. (13) can be written as

$$\frac{\alpha_p(t^{n+1})-\alpha_p(t^n)}{\Delta t}+\frac{1}{\mathcal{V}_p}\sum_{i=1}^{N_S}(\alpha_{p,i}\mathcal{U}_{p,i}\mathcal{A}_{p,i})=0, \tag{18}$$

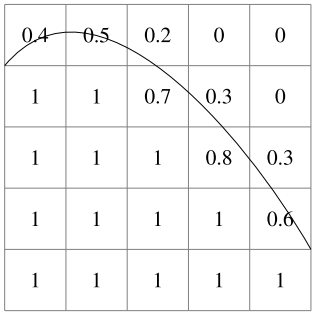



**Fig. 4.** Representation of interface (solid line) using a VOF scheme. Liquid volume fraction $\alpha$ shown with numbers.

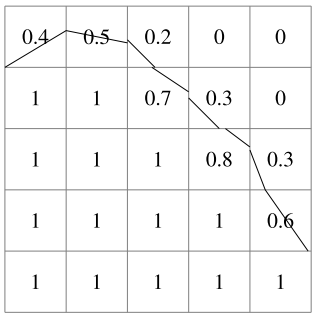



**Fig. 5.** Example of PLIC reconstruction of interface from liquid volume fraction.

which describes the change in liquid volume fraction within the $p$th computational cell due to fluxes, defined using streak-tubes, through the cell faces. No discretization choices have been made in the derivation of the previous equation and it is the equivalent of Eq. (1), where $f$ is chosen to be the liquid volume fraction, written for an arbitrary control volume.

## 3. Computational geometry toolbox

The framework presented heretofore requires calculating the volume and liquid volume fraction associated with the flux volumes $\Omega_{p,i}$ using a scheme that is accurate, efficient, and implementable in three dimensions. In this section, we present the computational tools needed to make the necessary calculations.

### 3.1. Interface reconstruction

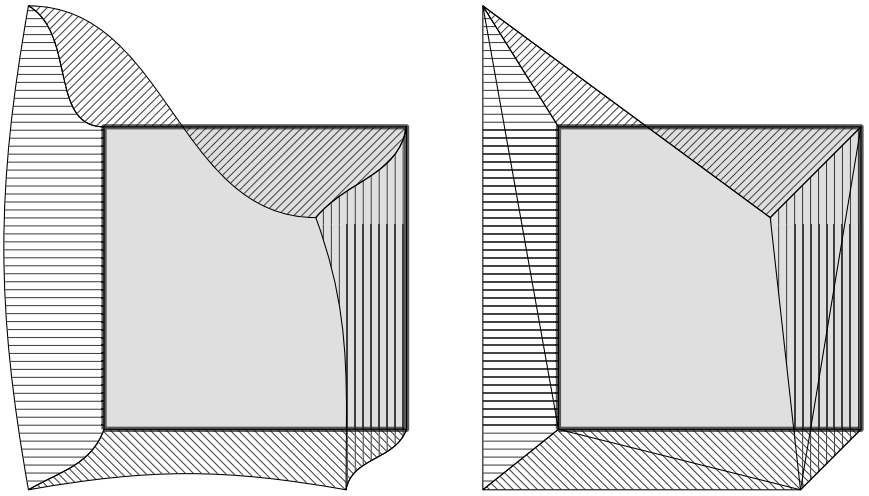
The interface reconstruction step corresponds to the process of calculating an approximation to the interface location from the liquid volume fraction. The interface location is needed to determine $\alpha_{p,i}$, the liquid volume fraction within the flux volume $\Omega_{p,i}$. A variety of methods have been developed in order to perform that reconstruction step [4,11,9,13]. We are using a three-dimensional extension of the piecewise linear interface calculation (PLIC) method [9,13], wherein the interface is represented using a piecewise planar reconstruction within each computational cell. Fig. 5 shows an example of how the interface in Fig. 4 may look when it is reconstructed.

Within each computational cell, the plane is uniquely defined with the interface normal vector and the liquid volume fraction. The normal vector provides the direction of the plane and the liquid volume fraction constrains the location of the plane such that the cell volume on the liquid side of the plane equals $\alpha_p \mathcal{V}_p$. In this work, the second-order interface normal calculation method known as ELVIRA [15] is used. Scardovelli and Zaleski [25] developed analytical relationships that permit the calculation of the PLIC reconstruction plane from the normal and liquid volume fraction directly. The combination of the analytical relations, the interface normal, and the liquid volume fraction provides a unique piecewise planar representation of the interface. Alternatively, an iterative approach such as Brent's method [26] could be used to form the PLIC reconstruction.

### 3.2. Discrete representation of the flux volume

#### 3.2.1. Tessellation

The flux volumes $\Omega_{p,i}$ are streak-tubes that generally do not have flat faces (even in two dimensions) and are non-convex with both positive and negative regions. In order to deal with these objects, we propose approximating the flux volumes



(a) Example of real geometry. (b) Discrete representation of geometry.

**Fig. 6.** Example geometry of the two-dimensional flux volume associated with a computational cell (shaded). (a) Shows an example of a realistic geometry where the four flux volumes (indicated with lines) do not have straight edges. (b) Shows how the real geometry can be approximated using two simplices per flux volume.

with a collection of simplices (triangles in two dimensions and tetrahedra in three dimensions), denoted $\Delta_{p,i,j}$ for $j = 1,\ldots,N_{\text{sims}}$ where $N_{\text{sims}}$ is the number of simplices such that

$$\Omega_{p,i} \approx \widetilde{\Omega}_{p,i} = \bigcup_{j=1}^{N_{\text{sims}}} \Delta_{p,i,j}. \tag{19}$$

In the previous equation, $\widetilde{\Omega}_{p,i}$ is the discrete representation of $\Omega_{p,i}$, which is simpler to manipulate computationally. Similarly, the discrete approximations of $\alpha_{p,i}$ and $\mathcal{V}_{p,i}$ are denoted with $\widetilde{\alpha}_{p,i}$ and $\widetilde{\mathcal{V}}_{p,i}$, respectively.

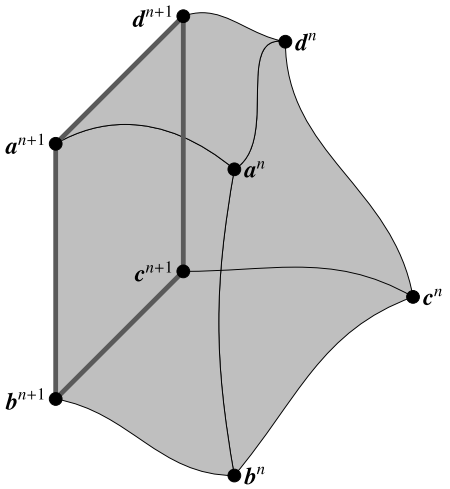
The number of simplices used in the tessellation depends on a balance between computational cost and accuracy, both of which increase with increasing number of simplices. In a second-order implementation, edges of the flux volume can be approximated with straightlines. In two dimensions, the resulting shape can therefore be represented using two simplices. This is shown in Fig. 6, wherein each of the flux volumes associated with a two-dimensional computational cell are represented using two simplices. A three-dimensional flux hexahedral volume with straight edges can be represented with five simplices. However, faces on opposite sides of the flux volumes will be cut along different diagonals, which can result in overlaps between the tessellated faces of neighboring flux volumes. Using six simplices avoids this source of error since the diagonals on opposite sides of the flux volume go in the same direction. Fig. 7 shows an example of the discretization of a three-dimensional flux volume using six simplices. Notice in the figure how the front face with vertices $\boldsymbol{a}^n\boldsymbol{a}^{n+1}\boldsymbol{b}^{n+1}\boldsymbol{b}^n$ is cut along the diagonal $\boldsymbol{a}^{n+1}\boldsymbol{b}^n$, which is in the same direction as the diagonal $\boldsymbol{d}^{n+1}\boldsymbol{c}^n$ that cuts the opposite face. In a five-simplex representation, the front face would still be cut by the $\boldsymbol{a}^{n+1}\boldsymbol{b}^n$ diagonal, but the opposite face would be cut along the $\boldsymbol{c}^{n+1}\boldsymbol{d}^n$ diagonal. Again, using more simplices improves the discrete representation of the flux volume but the computational cost also increases.

The simplices are created using a systematic approach that makes the implementation straightforward. Vertices that exist on the corners of the computational cell face are identified. Since the flux volume is a streak-tube, each vertex is transported back in time along its streak-line. For example, in Fig. 7 the vertices with superscript $n+1$ are on the corners of $\partial CS_{p,i}$. These vertices are transported back in time to their locations with superscript $n$. The simplices can then be constructed from the location of the original and transported vertices.

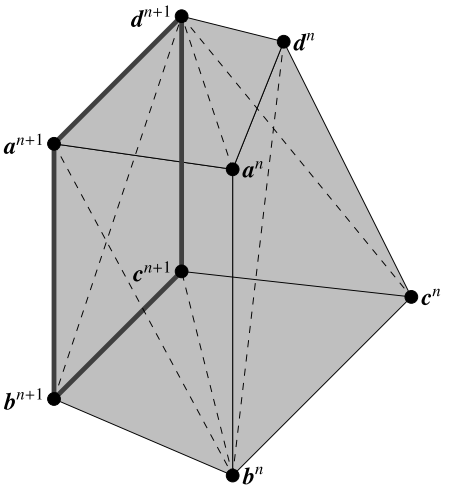
Vertex transport along a streak-line can be described using

$$\begin{aligned} &\frac{d\boldsymbol{x}_v(t)}{dt} = \boldsymbol{u}\big(\boldsymbol{x}_v(t),t\big) \quad \text{and} \\ &\boldsymbol{x}_v(t_0) = \boldsymbol{x}_{v,0}, \end{aligned} \tag{20}$$

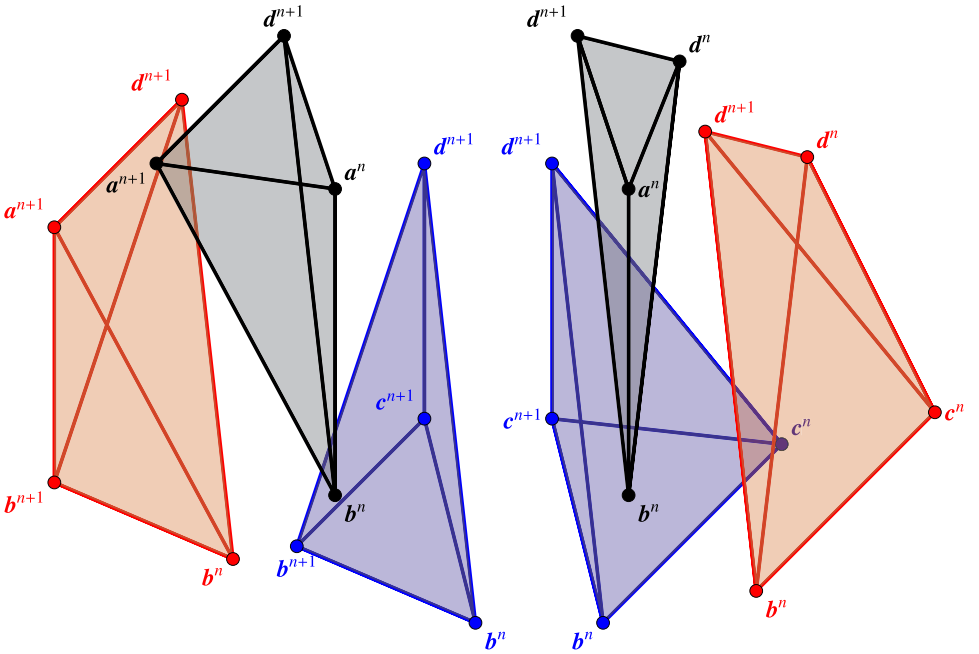
where $\boldsymbol{x}_v(t)$ is the position of the vertex at time $t$ and $\boldsymbol{x}_{v,0}$ is the starting location of the vertex on the computational cell face at time $t_0$. This equation is integrated backwards in time to find $\boldsymbol{x}_v(t^n)$. To simplify the process, we assume the velocity field is time-invariant. This is a reasonable (second-order) approximation since the velocity and interface transport steps are staggered in time in our implementation. Many time integration strategies can be used to solve Eq. (20), however higher-order methods will result in a vertex more closely following its streak-line. We have tested a range of Runge–Kutta schemes from first to sixth order and found little difference in the computed solutions. A second-order Runge–Kutta method is consistent with the rest of the scheme and is chosen due to the low cost compared with higher-order methods.




(a) Example of a flux volume in three dimensions.




(b) Discrete representation of a flux volume using six simplices.




(c) Exploded view of discrete representation of a flux volume using six simplices.

**Fig. 7.** Example of a three-dimensional flux volume associated with the computational cell face with vertices $(\boldsymbol{abcd})^{n+1}$. (a) Shows the real shape of the flux volume; (b) and (c) show how the flux volume is discretized using simplices.

### 3.2.2. *Simplex construction and sign convention*

In two dimensions, it is possible to list all of the different shapes the flux volume can take and to systematically break the shapes into two simplices with associated signs that indicate whether the flux is into or out of $CV_p$. However, this approach is tedious due to the large number of different flux volume shapes that exist. In three dimensions, a similar approach is even more difficult to implement. Therefore, we propose a straightforward and systematic way of constructing the simplices and determining their sign.

We start by associating a sign to each of the simplices in the partition of $\widetilde{\Omega}_{p,i}$ based on the location and ordering of the simplices' vertices. The sign can easily be determined using

$$\text{sign}(\Delta) \equiv \text{sign}\big((\boldsymbol{b}-\boldsymbol{a}) \times (\boldsymbol{c}-\boldsymbol{a})\big) \tag{21}$$

for the two-dimensional simplex with vertices $\boldsymbol{abc}$ and

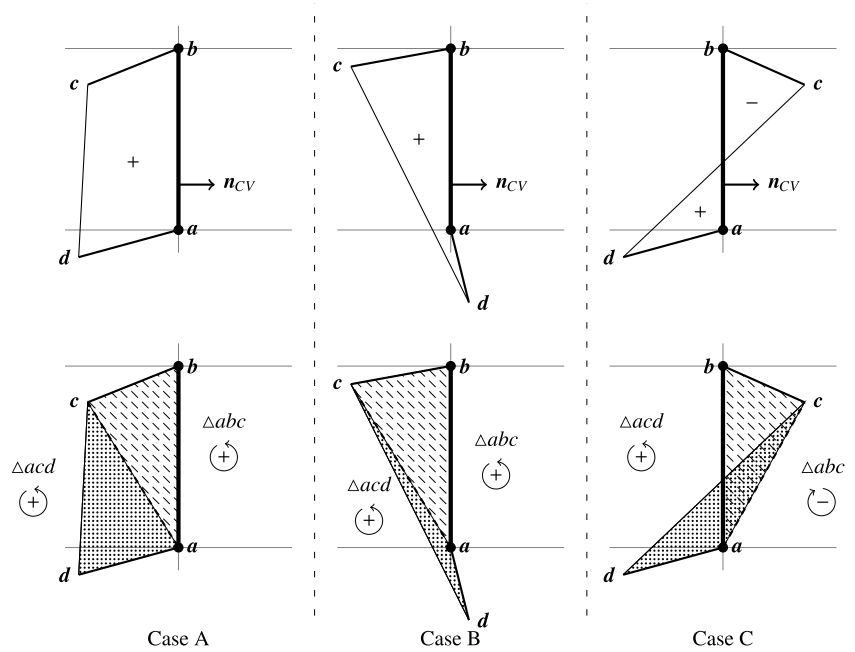



**Fig. 8.** Example of the discrete representation of two-dimensional flux volumes using signed simplices.

$$\text{sign}(\Delta) \equiv \text{sign}\left[\left((\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a})\right)\cdot\left(\boldsymbol{d}-\frac{1}{3}(\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c})\right)\right] \tag{22}$$

in three dimensions for the simplex with vertices ***abcd***. Clearly, the ordering of the vertices determines the sign and therefore it has to be chosen such that, when a simplex contributes a positive flux, it has a positive sign, and when the simplex contributes a negative flux, it has negative sign. The sign of the simplex determines whether it is part of $\widetilde{\Omega}^{+}_{p,i}$ or $\widetilde{\Omega}^{-}_{p,i}$ (the discretized counterparts to $\Omega^{+}_{p,i}$ and $\Omega^{-}_{p,i}$), i.e.,

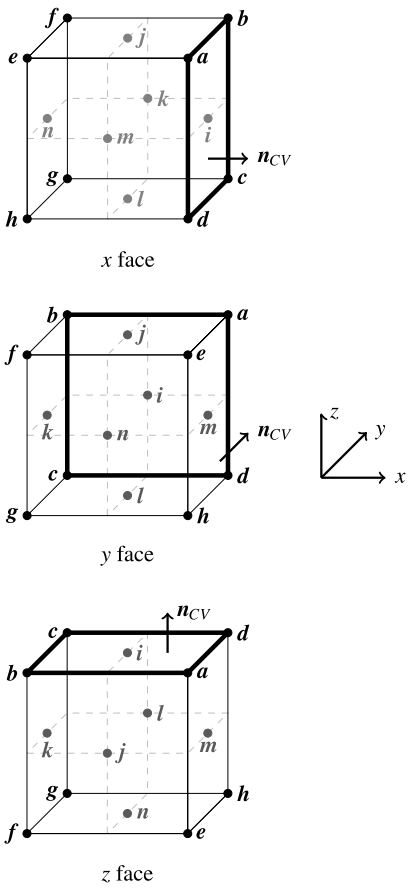
$$\widetilde{\Omega}^{+}_{p,i} = \bigcup_{\substack{j=1 \\ \text{sign}(\Delta_{p,i,j})=+1}}^{N_{\text{sims}}} \Delta_{p,i,j} \quad \text{and} \quad \widetilde{\Omega}^{-}_{p,i} = \bigcup_{\substack{j=1 \\ \text{sign}(\Delta_{p,i,j})=-1}}^{N_{\text{sims}}} \Delta_{p,i,j}. \tag{23}$$

Simplices created from the same ordering of vertices are used to represent all of the different flux volume shapes, which greatly simplifies the implementation. For example, Fig. 8 shows three two-dimensional flux volumes that represent the three main categories of shapes that are possible in two dimensions. In the figure, the cell face is the line ***ab***, ***d*** is the location of ***a*** at $t^n$ found by solving Eq. (20), and ***c*** is the location of ***b*** at $t^n$. In all of the cases the flux volumes are represented using the simplices $\Delta$***abc*** and $\Delta$***acd***. The sign of each simplex can be calculated using Eq. (21), and with these definitions it is found to be consistent with the sign of the flux that the simplex is representing.

Note that simplices may extend outside of $\widetilde{\Omega}_{p,i}$, i.e., $\{\widetilde{\Omega}_{p,i}\cup\Delta_{i,j}\}\setminus\{\widetilde{\Omega}_{p,i}\cap\Delta_{i,j}\}\neq\emptyset$ as shown by case C in Fig. 8. However, whenever a simplex extends outside of $\widetilde{\Omega}_{p,i}$, another simplex of opposite sign also extends outside and the net contribution is zero.

This systematic approach to constructing the simplices is readily extendable to three dimensions. Similarly to the two-dimensional implementation, the vertices of a computational cell face are transported back in time using Eq. (20), the simplices are created using a predefined ordering of the vertices, and the sign of the simplex determines the direction of the flux. Fig. 9 provides details on the predefined ordering of the vertices used to make the simplices. Details of discretizations that use 6 and 20 simplices per flux volume are included.

The ordering of the vertices in three dimensions is important to ensure that the faces that are shared between neighboring flux volumes are discretized in a consistent way and that no gaps or overlaps are formed in the geometry. Fig. 10 illustrates two ways in which the face of a flux volume can be shared with a neighboring flux volume. In the figure, the shaded faces are shared between neighboring flux volumes and should be discretized using the same representation by all flux volumes that share the face. The ordering of the vertices provided in Fig. 9 has been constructed such that flux volume faces are consistently discretized by neighboring flux volumes. This is trivial for the 20 simplex discretization because all faces are discretized the same way using the four corners and the face barycenter. Contrarily, the 6 simplex discretization approximates each face with two triangles that depend on which diagonal is used. Therefore, an equivalent to the 20 sim-



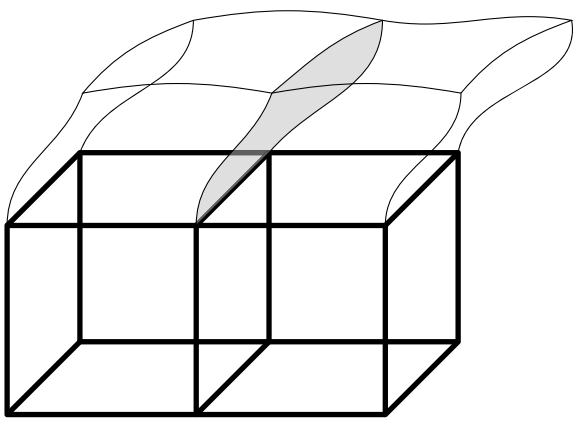

6 Simplices

| | | | | |
|---|---|---|---|---|
| 1 | ***g*** | ***d*** | ***c*** | ***f*** |
| 2 | ***f*** | ***c*** | ***b*** | ***d*** |
| 3 | ***f*** | ***b*** | ***a*** | ***d*** |
| 4 | ***g*** | ***h*** | ***d*** | ***f*** |
| 5 | ***f*** | ***e*** | ***h*** | ***d*** |
| 6 | ***f*** | ***e*** | ***d*** | ***a*** |

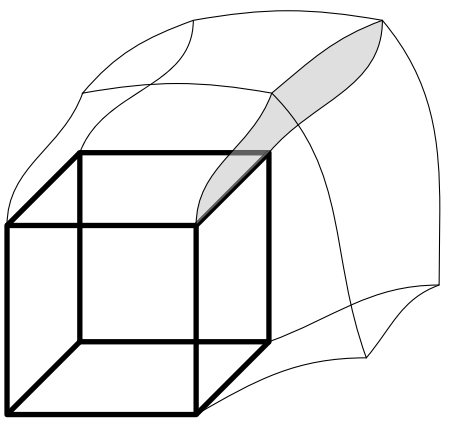
20 Simplices

| | | | | |
|---|---|---|---|---|
| 1 | ***a*** | ***e*** | ***j*** | ***i*** |
| 2 | ***e*** | ***f*** | ***j*** | ***i*** |
| 3 | ***f*** | ***b*** | ***j*** | ***i*** |
| 4 | ***b*** | ***a*** | ***j*** | ***i*** |
| 5 | ***b*** | ***f*** | ***k*** | ***i*** |
| 6 | ***f*** | ***g*** | ***k*** | ***i*** |
| 7 | ***g*** | ***c*** | ***k*** | ***i*** |
| 8 | ***c*** | ***b*** | ***k*** | ***i*** |
| 9 | ***c*** | ***g*** | ***l*** | ***i*** |
| 10 | ***g*** | ***h*** | ***l*** | ***i*** |
| 11 | ***h*** | ***d*** | ***l*** | ***i*** |
| 12 | ***d*** | ***c*** | ***l*** | ***i*** |
| 13 | ***d*** | ***h*** | ***m*** | ***i*** |
| 14 | ***h*** | ***e*** | ***m*** | ***i*** |
| 15 | ***e*** | ***a*** | ***m*** | ***i*** |
| 16 | ***a*** | ***d*** | ***m*** | ***i*** |
| 17 | ***e*** | ***h*** | ***n*** | ***i*** |
| 18 | ***h*** | ***g*** | ***n*** | ***i*** |
| 19 | ***g*** | ***f*** | ***n*** | ***i*** |
| 20 | ***f*** | ***e*** | ***n*** | ***i*** |

**Fig. 9.** Ordering of vertices used in the construction of simplices. The figures on the left show the ordering of the vertices and the tables on the right provide a list of the four vertices used to construct each simplex. The vertices $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ are located on the corners of the cell face. The vertices $\boldsymbol{e}$, $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{h}$ are the locations of the vertices $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ at time $t^n$, respectively, found by solving Eq. (20). Vertex $\boldsymbol{i}$ is located at the barycenter of the cell face $\boldsymbol{abcd}$. Vertex $\boldsymbol{n}$ is the location of vertex $\boldsymbol{i}$ at time $t^n$, found by solving Eq. (20). Vertices $\boldsymbol{j}$, $\boldsymbol{k}$, $\boldsymbol{l}$, and $\boldsymbol{m}$ are found by solving Eq. (20) for $\frac{1}{2}\Delta t$ from points $\frac{1}{2}(\boldsymbol{a}+\boldsymbol{b})$, $\frac{1}{2}(\boldsymbol{b}+\boldsymbol{c})$, $\frac{1}{2}(\boldsymbol{c}+\boldsymbol{d})$, and $\frac{1}{2}(\boldsymbol{d}+\boldsymbol{a})$, respectively.



(a) Face shared between flux volumes associated with neighboring computational cells.



(b) Face shared between flux volumes associated with the same computational cell.

**Fig. 10.** Illustration of flux volume faces that are shared by neighboring flux volumes. The shaded faces are shared between the two flux volumes shown in each figure. The discrete representation of the non-flat face needs to be consistent between all flux volumes that share the face.

plex discretization that uses face barycenters could be used to discretize flux volumes that are produced in the context of an unstructured code.

Using this framework, the calculation of the volume and liquid volume within the flux volume becomes systematic and straightforward. First, the cell face vertices are transported back in time to the beginning of the time-step, $t^n$. Then, the flux volume is approximated with a collection of predefined simplices, provided in Fig. 9 for a Cartesian mesh. These simplices are defined to be non-overlapping and their sign is consistent with the definitions from Section 2. Finally, the volume and the amount of liquid within a simplex remain to be calculated.

### 3.2.3. Flux calculation

In order to solve Eq. (18), the quantities $\alpha_{p,i}$ and $\mathcal{U}_{p,i}$ defined using Eqs. (17) and (14) need to be calculated. The problem is simplified thanks to the discrete representation of the flux volume as a collection of signed simplices, thus we only need to calculate the liquid volume fraction and volume of a simplex, denoted $\widetilde{\alpha}_{\Delta_{p,i,j}}$ and $\widetilde{\mathcal{V}}_{\Delta_{p,i,j}}$, respectively. The discrete quantities $\widetilde{\alpha}_{p,i}$ and $\widetilde{\mathcal{U}}_{p,i}$ can then be generated using

$$\widetilde{\alpha}_{p,i} = \frac{\sum_{j=1}^{N_{\text{sims}}} \widetilde{\alpha}_{\Delta_{p,i,j}} \widetilde{\mathcal{V}}_{\Delta_{p,i,j}}}{\sum_{j=1}^{N_{\text{sims}}} \widetilde{\mathcal{V}}_{\Delta_{p,i,j}}} \tag{24}$$

and

$$\widetilde{\mathcal{V}}_{p,i} = \sum_{j=1}^{N_{\text{sims}}} \widetilde{\mathcal{V}}_{\Delta_{p,i,j}}, \tag{25}$$

along with the discrete equivalent of Eq. (14).

The signed volume of a simplex can be calculated easily by combining the sign convention, Eq. (22), and the Cayley–Menger determinant [27], which in three dimensions can be written as

$$\widetilde{\mathcal{V}}_{\Delta_{p,i,j}} = -\frac{(\boldsymbol{a}-\boldsymbol{d})\cdot((\boldsymbol{b}-\boldsymbol{d})\times(\boldsymbol{c}-\boldsymbol{d}))}{6} \tag{26}$$

where $\Delta_{p,i,j}$ has vertices $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$.

The liquid volume fraction in a simplex is more complicated to compute and depends on the location of the gas–liquid interface. In our method, the gas–liquid interface is represented using the PLIC scheme described in Section 3.1, which uses a piecewise planar representation of the interface that is local to each computational cell. To calculate $\widetilde{\alpha}_{p,i}$, a given simplex is cut by cell faces and divided into regions that are local to each computational cell, and then cut by the gas–liquid interface, resulting in regions that are exclusively within the liquid phase or exclusively within the gas phase. Then, the liquid volume is easily calculated as the sum of volumes within the liquid phase. To make the algorithm tractable, the shapes that are created from cutting a simplex by a plane are partitioned into a new collection of simplices that can easily be cut again. The process continues until each simplex lies within only one phase.

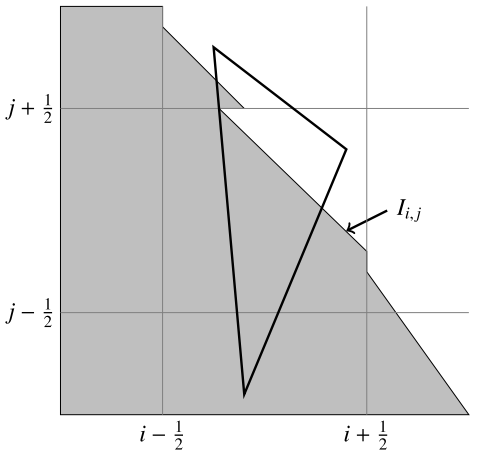
For example, Fig. 11 shows the process used to cut a simplex by two planes that are coincident with the computational grid, followed by a cut by the PLIC representation of the interface. The simplex shown in Fig. 11(a) is first cut by the plane indicated by the $j-\frac{1}{2}$ face of the cell considered, which results in two shapes. The bottom shape is a triangle and the top shape is a quadrilateral. The triangle is already a simplex, but the quadrilateral needs to be partitioned into two simplices as shown by Fig. 11(b). Next, the three simplices are cut by the plane at the $j+\frac{1}{2}$ index, which again leads to the partition of a simplex into a triangle and a quadrilateral. The quadrilateral is divided into two simplices as shown by Fig. 11(c). Finally, the simplices are cut by the reconstructed gas–liquid interface and partitioned into more simplices as shown by Fig. 11(d). Now it is straightforward to compute the liquid volume within the original simplex from the volumes of the new simplices that contain liquid. For example, the liquid volume within the simplex in Fig. 11(a) is equal to the sum of the volumes of the shaded simplices in Fig. 11(d).

The number of planes by which the simplex is cut depends on the location of the simplex vertices. In our implementation, we identify which planes the simplex needs to be cut by using an initialization routine that is based on the location of the vertices.

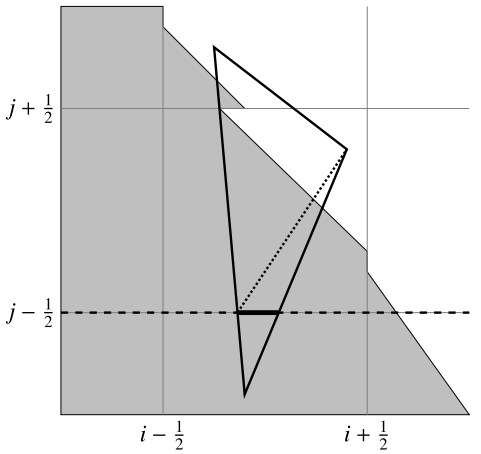
## 3.3. Construction of conservative fluxes

As described in Section 2.2, the flux velocity must be solenoidal to ensure discrete conservation. In our implementation a staggered velocity field is used, meaning that the face-normal component of the velocity is available at the center of each computational cell face. There is no guarantee that interpolating this velocity to cell face vertices and projecting the vertices back in time to create a flux volume will produce a flux velocity $\mathcal{U}_{p,i}$ that is also solenoidal. Therefore, the flux volume is modified to ensure discrete conservation. The modification is typically small and does not alter the second-order accuracy of the scheme as shown in the verification tests.

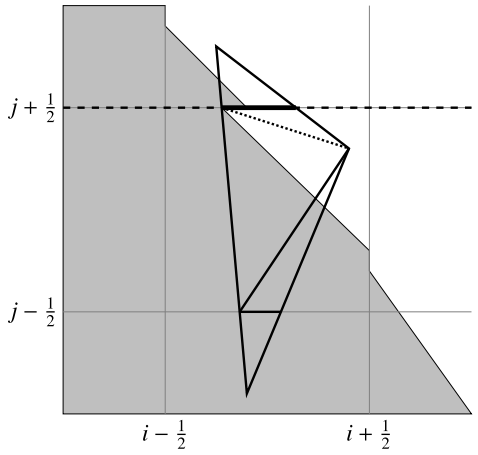
Several approaches have been used previously to modify the fluxes to improve conservation. Liovic et al. [28] scaled the multi-dimensional fluxes with conservative one-dimensional fluxes. Mencinger and Žun [29] used a parametric correction



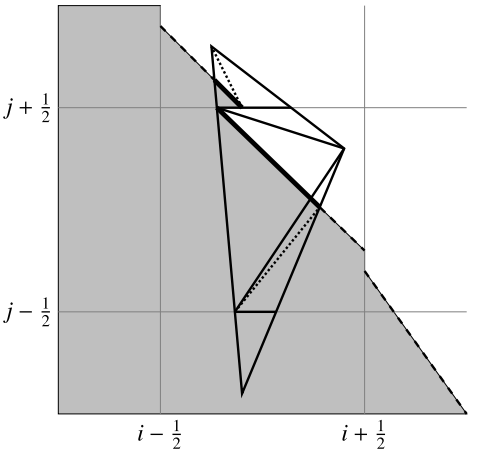

(a) Original simplex and PLIC reconstruction of interface indicated with $I_{i,j}$ in cell $i, j$.




(b) Simplex is cut by plane at $j-\frac{1}{2}$ (thick solid line). Resulting shapes are partitioned into simplices (dotted line).




(c) Simplices in (b) are cut by plane at $j+\frac{1}{2}$ (thick solid line). Resulting shapes are partitioned into simplices (dotted line) that are unique to one computational cell.




(d) Simplices from (c) are cut by liquid-gas interface reconstruction $I$ (thick solid line). Resulting shapes are partitioned into simplices (dotted lines) that are within one computational cell and one phase.

**Fig. 11.** Steps used to calculate the liquid volume fraction within a simplex that crosses multiple planes.

approach to modify the time-step used to project each vertex in the construction of the flux volume, but the extension to three-dimensions is unclear. López et al. [16] and Hernández et al. [17] used analytical relations to modify the size of the flux volume so that a conservative flux is constructed. In their approaches, the flux volume is constrained by the volume of a one-dimensional conservative flux built using a solenoidal face velocity, e.g., $\widehat{\mathcal{U}}_{p,i} = (\boldsymbol{u} \cdot \boldsymbol{n}_{CV})_{p,i}$, where $\widehat{\mathcal{U}}_{p,i}$ is the modified flux velocity and $(\boldsymbol{u} \cdot \boldsymbol{n}_{CV})_{p,i}$ is the normal component of the solenoidal velocity on the $i$th face of the $p$th computational cell. The resulting modification is equivalent to adjusting the time-step used in Eq. (20) to project the vertices along streak-lines. A similar approach is used in the proposed scheme. However, modifying the time-step can create overlapping regions between neighboring flux volumes when the faces are non-planar as shown in Fig. 12. Therefore, the modification is performed in a way that avoids creating overlapping regions that would result in a conservation error.

The proposed method consists of adding additional simplices to the flux volume such that $\widehat{\mathcal{U}}_{p,i} = (\boldsymbol{u} \cdot \boldsymbol{n}_{CV})_{p,i}$. The additional simplices must not modify the discretization of faces of the flux volume that are shared with neighboring flux volumes in order to ensure that no overlapping regions are created. Therefore, the simplices are added onto the projected face, e.g., the face of the flux volume opposite from the cell face $CS_{p,i}$. The number of additional simplices is equal to the number of simplices used to discretize the projected face. The volume of the additional simplices $\mathcal{V}_{\text{cor}}$ can be calculated from the difference between the pre-modified flux velocity, the solenoidal flux velocity, and Eq. (14), leading to

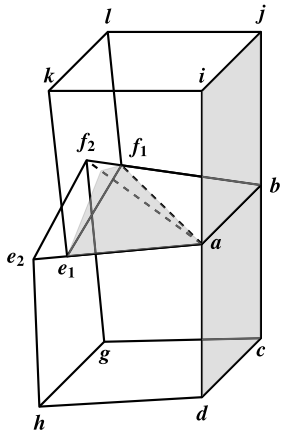



**Fig. 12.** Overlapping region formed between neighboring flux volumes $\boldsymbol{abcde_2f_2gh}$ and $\boldsymbol{baijf_1e_1kl}$ when simple rescaling (i.e., time-step adjustment) is used to create conservative fluxes. The face with vertices $\boldsymbol{abf_1e_1}$ is triangulated with diagonal $\boldsymbol{af_1}$. Rescaling the bottom flux volume moves the projected vertices creating the shaded overlapping region.

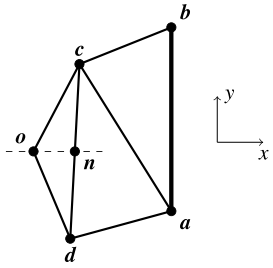



**Fig. 13.** Modification of two-dimensional flux volume to create solenoidal flux. Original flux volume associated with face $\boldsymbol{ab}$ has vertices $\boldsymbol{abcd}$. The additional simplex with vertices $\boldsymbol{cdo}$ is added.

$$\mathcal{V}_{\text{cor}} = \left((\boldsymbol{u}\cdot\boldsymbol{n}_{CV})_{p,i} - \mathcal{U}_{p,i}\right)\Delta t\mathcal{A}_{p,i}. \tag{27}$$

Fig. 13 shows a two-dimensional example for the flux through an $x$-face $\boldsymbol{ab}$ with flux volume $\boldsymbol{abcd}$. The flux volume is modified by adding the simplex $\boldsymbol{dco}$. A closed form analytic expression for the coordinates of vertex $\boldsymbol{o}$ can be formed by enforcing two constraints. We choose to constrain the $y$ coordinate of vertex $\boldsymbol{o}$ by the $y$ coordinate of $\boldsymbol{n}$, which is the barycenter of the face $\boldsymbol{cd}$. The $x$ coordinate is constrained by the volume $\mathcal{V}_{\text{cor}}$. Similarly, the additional simplex on a $y$-face flux volume is constrained by the $x$ component of the barycenter and the correction volume.

This approach to modify the flux volume is easily extended to three dimensions, as shown for example in Fig. 14. The flux volume $\boldsymbol{abcdefgh}$, associated with the $x$-face $\boldsymbol{abcd}$ is modified by adding the two simplices $\boldsymbol{efho}$ and $\boldsymbol{fgho}$. The location of vertex $\boldsymbol{o}$ is determined by enforcing three constraints. Two constraints come from enforcing that the $y$ and $z$ coordinates of vertex $\boldsymbol{o}$ are equal to the $y$ and $z$ coordinates of $\boldsymbol{n}$, the barycenter of face $\boldsymbol{efgh}$. The $x$ coordinate is constrained by the correction volume, i.e.,

$$\mathcal{V}_{\text{cor}} = \mathcal{V}_{\Delta(efho)} + \mathcal{V}_{\Delta(fgho)} = -\frac{(\boldsymbol{e}-\boldsymbol{o})\cdot((\boldsymbol{f}-\boldsymbol{o})\times(\boldsymbol{h}-\boldsymbol{o}))}{6} - \frac{(\boldsymbol{f}-\boldsymbol{o})\cdot((\boldsymbol{g}-\boldsymbol{o})\times(\boldsymbol{h}-\boldsymbol{o}))}{6}. \tag{28}$$

Similar constraints are enforced for flux volumes associated with $y$ and $z$ computational cell faces. The three constraints provide an analytic relation that relates the location of vertices $\boldsymbol{e}$, $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{h}$ and the volume $\mathcal{V}_{\text{cor}}$ to the location of vertex $\boldsymbol{o}$, which can be evaluated to quickly compute the modification to the flux volume. The analytic relations for computing $\boldsymbol{o}$ are provided in Algorithm 1 for all the faces of a computational cell.

If the flux volume $\boldsymbol{abcdefgh}$ in the three-dimensional example is discretized with 20 simplices, four additional simplices are added because the projected face is discretized with four simplices. The additional simplices are $\boldsymbol{heno}$, $\boldsymbol{efno}$, $\boldsymbol{fgno}$, and $\boldsymbol{ghno}$. The $y$ and $z$ coordinates of $\boldsymbol{o}$ are constrained by the barycenter $\boldsymbol{n}$ and the $x$ coordinate is constrained by the volume $\mathcal{V}_{\text{cor}}$. An analytical relation can be found to solve for the $x$ coordinate that is similar to the relation in Eq. (28).

### 3.4. Parallelization

The proposed scheme has been implemented using a domain decomposition parallelization strategy within the NGA computational platform [30]. The geometric transport routines require the geometry of neighboring cells and the PLIC reconstruction in those cells. Using standard operations within NGA to update ghost cells on domain boundaries, the interface

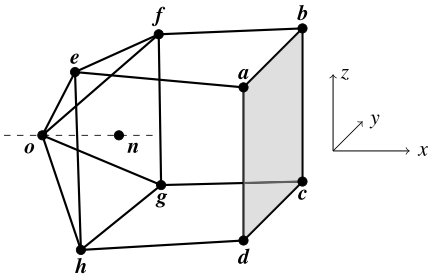



**Fig. 14.** Modification of three-dimensional flux volume to create solenoidal flux. Original flux volume associated with face ***abcd*** has vertices ***abcdefgh***. Two additional simplices with vertices ***efho*** and ***fgho*** are added if six simplices are used to discretize the original volume.

**Algorithm 1** SolenoidalFlux: routine to compute two additional simplices to construct conservative flux. Algorithm assumes the original flux is discretized with six simplices as shown in Fig. 9.

**function** SolenoidalFlux($D, \boldsymbol{V}, \mathcal{V}_{\text{cor}}$)
  **input** $D$ ▷ Direction of face
  **input** $\boldsymbol{V}$ ▷ Vertices on flux volume as shown in Fig. 14
  **input** $\mathcal{V}_{\text{cor}}$ ▷ Volume of additional simplices
  $\boldsymbol{e} \leftarrow \boldsymbol{V}_5$
  $\boldsymbol{f} \leftarrow \boldsymbol{V}_6$
  $\boldsymbol{g} \leftarrow \boldsymbol{V}_7$
  $\boldsymbol{h} \leftarrow \boldsymbol{V}_8$
  $\boldsymbol{n} \leftarrow \frac{1}{4}(\boldsymbol{e}+\boldsymbol{f}+\boldsymbol{g}+\boldsymbol{h})$
  **switch** $D$ **do**
    **case** 1
      $o_1 \leftarrow (6\mathcal{V}_{\text{cor}} + e_1f_2h_3 - e_1f_3h_2 - e_2f_1h_3 + e_2f_3h_1 + e_3f_1h_2 - e_3f_2h_1 - e_1f_2e_3 + e_1f_3e_2 + e_2f_1e_3$
        $+e_3h_1e_2 - e_3f_1e_2 + f_1g_2h_3 - f_1g_3h_2 - f_2g_1h_3 + f_2g_3h_1 + f_3g_1h_2 - f_3g_2h_1 + e_1h_2e_3 - e_1h_3e_2$
        $-e_2h_1e_3 - f_1g_2e_3 + f_1g_3e_2 + f_2g_1e_3 - f_3g_1e_2 - g_1h_2e_3 + g_1h_3e_2 + g_2h_1e_3 - g_3h_1e_2)$
        $/(e_2f_3 - e_3f_2 - e_2h_3 + e_3h_2 + f_2g_3 - f_3g_2 + g_2h_3 - g_3h_2)$
      $o_2 \leftarrow n_2$
      $o_3 \leftarrow n_3$
    **case** 2
      $o_1 \leftarrow n_1$
      $o_2 \leftarrow -(6\mathcal{V}_{\text{cor}} + e_1f_2h_3 - e_1f_3h_2 - e_2f_1h_3 + e_2f_3h_1 + e_3f_1h_2 - e_3f_2h_1 - e_1f_2e_3 + e_2f_1e_3 - e_2f_3e_1$
        $+e_3f_2e_1 + f_1g_2h_3 - f_1g_3h_2 - f_2g_1h_3 + f_2g_3h_1 + f_3g_1h_2 - f_3g_2h_1 + e_1h_2e_3 - e_2h_1e_3 + e_2h_3e_1$
        $-e_3h_2e_1 - f_1g_2e_3 + f_2g_1e_3 - f_2g_3e_1 + f_3g_2e_1 - g_1h_2e_3 + g_2h_1e_3 - g_2h_3e_1 + g_3h_2e_1)$
        $/(e_1f_3 - e_3f_1 - e_1h_3 + e_3h_1 + f_1g_3 - f_3g_1 + g_1h_3 - g_3h_1)$
      $o_3 \leftarrow n_3$
    **case** 3
      $o_1 \leftarrow n_1$
      $o_2 \leftarrow n_2$
      $o_3 \leftarrow (6\mathcal{V}_{\text{cor}} + e_1f_2h_3 - e_1f_3h_2 - e_2f_1h_3 + e_2f_3h_1 + e_3f_1h_2 - e_3f_2h_1 + e_1f_3e_2 - e_2f_3e_1 - e_3f_1e_2$
        $+e_3f_2e_1 + f_1g_2h_3 - f_1g_3h_2 - f_2g_1h_3 + f_2g_3h_1 + f_3g_1h_2 - f_3g_2h_1 - e_1h_3e_2 + e_2h_3e_1 + e_3h_1e_2$
        $-e_3h_2e_1 + f_1g_3e_2 - f_2g_3e_1 - f_3g_1e_2 + f_3g_2e_1 + g_1h_3e_2 - g_2h_3e_1 - g_3h_1e_2 + g_3h_2e_1)$
        $/(e_1f_2 - e_2f_1 - e_1h_2 + e_2h_1 + f_1g_2 - f_2g_1 + g_1h_2 - g_2h_1)$
  **return** $N_{\text{Add}} \leftarrow 2$ ▷ Number of additional simplices
  **return** $\boldsymbol{S_1} = [\boldsymbol{e}, \boldsymbol{f}, \boldsymbol{h}, \boldsymbol{o}]^{\mathsf{T}}$ ▷ First additional simplex
  **return** $\boldsymbol{S_2} = [\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h}, \boldsymbol{o}]^{\mathsf{T}}$ ▷ Second additional simplex
**end function**

normal vector and the liquid volume fraction are communicated. With the communicated information, the PLIC reconstruction is computed on each processor and the geometric algorithm is used to advect the liquid volume fraction. The resulting scheme has minimal communication requirements and is expected to have excellent scale-up properties.

### 3.5. Extension to unstructured meshes

The proposed algorithm can be implemented within the context of an unstructured mesh with only minor modifications. The discretization of the flux volumes should be consistent with the cell face geometry, and faces of the flux volumes that are shared with neighboring flux volumes should be discretized such that there are no overlaps or gaps. The sign of the simplices should be constructed so that positive fluxes are represented by positively orientated simplices and negative fluxes are represented by negative simplices. The modification of the flux volume to make conservative fluxes can be performed using the proposed method. Once the flux volume is formed as a collection of simplices, the simplices are cut into regions that are unique to one computational cell using, for example, a polyhedron clipping and capping algorithm [18,21]. Finally, the simplices are cut by the PLIC reconstruction within each cell and the fluxes are computed. In summary, the main ideas of the proposed method can all be extended to unstructured grids, namely partitioning the flux volume into a collection

**Algorithm 2** UpdateVOF: framework to update the liquid volume fraction.

**function** UpdateVOF($\boldsymbol{\alpha}_p^n, \Delta t$)
  **input** $\boldsymbol{\alpha}_p^n$ ▷ Liquid volume fraction at $t^n$
  **input** $\Delta t$ ▷ Time-step
  $[\boldsymbol{n}] \leftarrow$ InterfaceNormal ▷ Compute interface normal vectors
  $[\boldsymbol{I}] \leftarrow$ InterfaceReconstruction($\boldsymbol{n}, \boldsymbol{\alpha}_p^n$) ▷ Compute interface reconstruction (Section 3.1)
  $[\boldsymbol{\alpha}_{p,i}, \boldsymbol{\mathcal{U}}_{p,i}] \leftarrow$ CalcFlux($\boldsymbol{I}, \Delta t$) ▷ Compute fluxes (Algorithm 3)
  **for** $p = 1 \rightarrow N_{CV}$ **do** ▷ Loop over control volumes that comprise computational mesh
    $\alpha_p^{n+1} \leftarrow \alpha_p^n - \frac{\Delta t}{\mathcal{V}_p} \sum_{i=1}^{N_S} (\alpha_{p,i} \mathcal{U}_{p,i} \mathcal{A}_{p,i})$ ▷ Update liquid volume fraction using Eq. (18)
  **end for**
  **return** $\boldsymbol{\alpha}_p^{n+1}$
**end function**

**Algorithm 3** CalcFlux: returns $\boldsymbol{\alpha}_{p,i}$ and $\boldsymbol{\mathcal{U}}_{p,i}$, i.e., arrays of the liquid volume fraction fluxes and the mean flux velocity associated with the flux volume $\Omega_{p,i}$.

**function** CalcFlux($\boldsymbol{I}, \Delta t$)
  **input** $\boldsymbol{I}$ ▷ Interface reconstruction
  **input** $\Delta t$ ▷ Time-step
  $\boldsymbol{\mathcal{L}}_p \leftarrow 0$ ▷ Zero arrays
  $\boldsymbol{\mathcal{V}}_p \leftarrow 0$
  **for** $p = 1 \rightarrow N_{CV}$ **do** ▷ Loop over control volumes that comprise computational mesh
    **for** $i = 1 \rightarrow N_S$ **do** ▷ Loop over faces of $p$th control volume
      $[N_{\text{planes}}, \boldsymbol{P}] \leftarrow$ CutPlanes($p, i$) ▷ Construct cut-planes
      $\boldsymbol{V} \leftarrow$ FaceVertices($p, i$) ▷ Create vertices on face according to Fig. 9
      $\boldsymbol{V} \leftarrow$ ProjectVertices($\boldsymbol{V}, \Delta t$) ▷ Project vertices to create flux volume using Eq. (20)
      $[N_{\text{Sim}}, \boldsymbol{S}] \leftarrow$ PartitionFlux($\boldsymbol{V}$) ▷ Partition flux volume into simplices according to Fig. 9
      $\mathcal{V}_{\text{cor}} \leftarrow$ CorrecttionVolume($D_i, N_{\text{Sim}}, \boldsymbol{S}$) ▷ Additional volume needed to correct flux, Eq. (27)
      $[N_{\text{Add}}, \boldsymbol{S}] \leftarrow$ SolenoidalFlux($D_i, \boldsymbol{V}, \mathcal{V}_{\text{cor}}$) ▷ Create additional simplices to construct conservative flux
      **for** $n = 1 \rightarrow N_{\text{Sim}} + N_{\text{Add}}$ **do** ▷ Loop over simplices and update signed volume and liquid volume fluxes
        $\mathcal{V}_{p,i} \leftarrow \mathcal{V}_{p,i} +$ SimplexVolume($\boldsymbol{S}(n)$) · SimplexSign($\boldsymbol{S}(n)$)
        $\mathcal{L}_{p,i} \leftarrow \mathcal{L}_{p,i} +$ SimplexLiquidVolume($\boldsymbol{S}(n), \boldsymbol{P}, N_{\text{planes}}, \boldsymbol{I}$) · SimplexSign($\boldsymbol{S}(n)$)
      **end for**
      $\alpha_{p,i} \leftarrow \mathcal{L}_{p,i} / \mathcal{V}_{p,i}$ ▷ Compute liquid volume fraction flux
      $\mathcal{U}_{p,i} \leftarrow \frac{\mathcal{V}_{p,i}}{\Delta t \mathcal{A}_{p,i}}$
    **end for**
  **end for**
  **return** $\boldsymbol{\alpha}_{p,i}$
  **return** $\boldsymbol{\mathcal{U}}_{p,i}$
**end function**

of simplices, assigning a sign to each simplex to determine the flux contribution, and correcting the flux volume with additional simplices. Furthermore, Algorithm 5 and the look-up tables that provide an efficient method to cut a simplex by a plane are not mesh dependent and could be used in an unstructured code. Additional details will depend on the detailed topology of the mesh and are beyond the scope of this paper.

### 3.6. Implementation

In this section, the process used to calculate the liquid and volume fluxes is detailed using pseudo-code. Algorithm 2 shows the general framework for updating the liquid volume fraction. First, the interface normal and PLIC reconstruction are computed using the methodology described in Section 3.1. Next, the fluxes are calculated, then the liquid volume fraction is updated.

Algorithm 3 provides the methodology to compute the fluxes. In the algorithm, the flux volume is created on every face on the computational mesh. Then, the flux volume is divided into simplices using PartitionFlux, which, due to the order of the vertices used to create the simplices, have the same sign as the flux volume they represent. Next, the flux volume is modified by adding additional simplices constructed using SolenoidalFlux (Algorithm 1). Finally, the signed volume and signed liquid volume within each simplex are calculated and added to running sums. The sign follows from the orientation of the simplex that is evaluated using SimplexSign, which should be based on Eq. (22). Note that the computational cost can be reduced by computing each flux once and using the value to update both computational cells that share the face. Care must be taken to ensure the sign of the flux is correct when updating each cell.

The liquid volume within a simplex is calculated with SimplexLiquidVolume (Algorithm 4). The algorithm takes a simplex, cuts the simplex by a plane and divides the resulting shapes into new simplices using CutSimplex. The new simplices are cut by another plane and divided into more new simplices. The process continues until each of the simplices is contained within a single computational cell and on one side of the reconstructed gas–liquid interface.

**Algorithm 4** SimplexLiquidVolume: returns the amount of liquid within the simplex $\boldsymbol{S}$.

```
1: function SimplexLiquidVolume(S, P, N_planes, I)
2:     input S                                              ▷ Array of simplex vertices
3:     input P                                              ▷ Array of cut-planes
4:     input N_planes                                       ▷ Number of planes in the array P
5:     input I                                              ▷ Array of planes that represent the gas–liquid interface
6:     L_vol ← 0
7:     [N_1, S_1] ← CutSimplex(S(:), P(1))                  ▷ Cut by first plane, partition S into N_1 simplices S_1
8:     for i = 1 → N_1 do
9:         [N_2, S_2] ← CutSimplex(S_1(i,:), P(2))          ▷ Cut by second plane
           ⋮
10:        for j = 1 → N_{N_planes−2} do
11:            [N_{N_planes−1}, S_{N_planes−1}] ← CutSimplex(S_{N_planes−2}(j,:), P(N_planes − 1))   ▷ Cut by N_planes − 1 plane
12:            for k = 1 → N_{N_planes−1} do
13:                [N_{N_planes}, S_{N_planes}] ← CutSimplex(S_{N_planes−1}(k,:), P(N_planes))      ▷ Cut by N_planes plane
14:                for m = 1 → N_{N_planes} do
15:                    p ← SimplexIndex(S_{N_planes}(m,:))   ▷ Get index of cell in which this simplex is
16:                    [N_I, S_I] ← CutSimplex(S_{N_planes}(m,:), I_p)   ▷ Cut by gas–liquid interface within this cell
17:                    for n = 1 → N_I do
18:                        if Distance(1/4 Σ_{v=1}^{4} S_I(n,v), I_p) < 0 then   ▷ Simplex is on liquid side
19:                            L ← L + SimplexVolume(S_I(n,:))   ▷ Add to liquid volume
20:                        end if
21:                    end for
22:                end for
23:            end for
               ⋮
24:        end for
25:    end for
26:    return L                                             ▷ Volume of liquid within S
27: end function
```

The operation to cut a simplex by a plane is performed by CutSimplex (Algorithm 5). This algorithm computes the distance between each vertex of the simplex and the plane that the simplex is being cut with. Based on the sign of the four distances, the number of intersections between the plane and the simplex edges is calculated and the intersection points are saved. Finally, the simplex is partitioned into a collection of new simplices using the original vertices and the intersection points. Note that the orientation of the simplices used in the partition of the original simplex is not important, only the orientation of the original simplex is used to determine the sign of the flux contribution, i.e., SimplexSign only appears in Algorithm 3 and depends on the original simplex.

To improve the efficiency of the CutSimplex algorithm we have introduced look-up tables (Algorithm 6). A case number, $Case \in \{1, \ldots, 16\}$, is created that classifies the simplex based on the sign of the distances between the simplex vertices and the cut-plane. The case, in conjunction with the look-up tables, provides

- NumberIntersect($Case$): the number of intersections between the edges of the simplex and the cut-plane,
- IndexEndPoint($v$, $n$, $Case$): the index of the $v$th end point on the end of the edge involved in the $n$th intersection between the simplex and the cut-plane,
- NumberSimsPart($Case$): the number of simplices in the partition of the original simplex,
- IndexSimVert($v$, $n$, $Case$): the index of the $v$th vertex on the $n$th simplex used to partition the original simplex.

For example, $Case = 2$ corresponds to a simplex with the 1st vertex on the positive side of the plane and the 2nd, 3rd, and 4th vertices on the negative side, as shown in Fig. 15. This simplex has three edges that intersect the plane and therefore NumberIntersect$(2) = 3$. The first intersection is between the plane and the edge with vertices 1 and 2, thus IndexEndPoint$(:, 1, 2) = [1, 2]$. The second intersection is with the edge with vertices 1 and 3, and the third intersection is with the edge with vertices 1 and 4, thus IndexEndPoint$(:, 2, 2) = [1, 3]$ and IndexEndPoint$(:, 3, 2) = [1, 4]$. The number of simplices used to partition the cut simplex is four, which is provided by NumberSimsPart$(2) = 4$. Finally, the indices of the vertices used to construct the new simplices are provided by IndexSimVert$(v, n, Case)$, where $v$ is the vertex number and $n$ is the simplex number. For our example, [IndexSimVert$(:, :, 2) = [[1, 5, 6, 7], [4, 2, 3, 6], [4, 2, 5, 6], [4, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]$, which provides the information needed to create the four simplices in our partition. The first simplex has vertices that correspond to the points with the index 1, 5, 6, and 7. The second simplex has vertices 4, 2, 3, and 6. The third and fourth simplices have vertices 4, 2, 5, 6 and 4, 5, 6, 7, respectively. Note that the zeros in IndexEndPoint and IndexSimVert indicate null values and are used to pad the arrays.

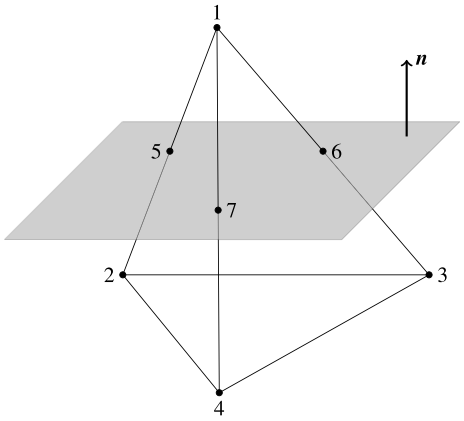
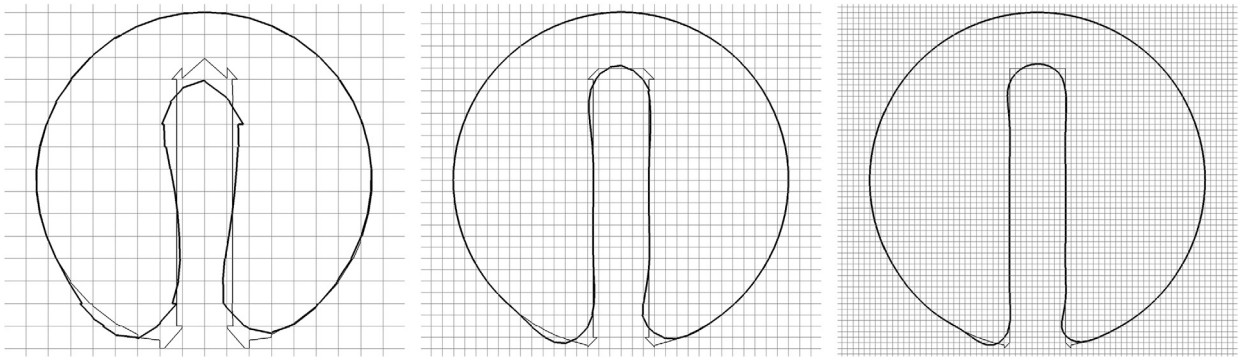



**Fig. 15.** Example simplex and cut-plane classified as $Case = 2$. Vertices indicated with their index number.



(a) $50^2$ mesh (b) $100^2$ mesh (c) $200^2$ mesh

**Fig. 16.** Zalesak's disk after one rotation of various meshes. The thick line is the computed solution and the thin line indicates the initial condition on each mesh.

## 4. Verification tests

The proposed scheme is studied using a variety of test cases. All of the tests use a second-order Runge–Kutta method to solve Eq. (20). The flux volumes are discretized using eight simplices, six from the initial discretization plus two additional simplices to construct conservative fluxes. All tests use a specified velocity field and do not require the velocity field to be obtained from a Navier–Stokes solver.

### 4.1. Zalesak's disk

The first verification test case is known as Zalesak's disk [31] and tests the ability of the proposed VOF scheme to transport a two-dimensional shape with sharp corners. The velocity field is specified to produce solid body rotation using

$$\begin{aligned} u &= -2\pi y, \\ v &= +2\pi x. \end{aligned} \tag{29}$$

The shape is a notched disk with diameter 0.3, notch width of 0.05, initially centered at $(x, y) = (0, 0.25)$ within a square domain $[-0.5, 0.5]^2$. The disk shape should not change given the specified velocity field and should simply rotate about the origin. The disk is rotated for one revolution using various meshes. Fig. 16 shows the shape of Zalesak's disk after it has been rotated. The images in the figure, and subsequent figures, show the PLIC representation of the gas–liquid interface. Even on the coarsest $50^2$ mesh, the method is able to maintain the notch and the shape of the rotated disk closely resembles the reference solution.

To test the proposed VOF scheme quantitatively we use the error norms

$$E_{\text{mass}}(t) = \sum_{p=1}^{N_{CV}} \mathcal{V}_p \alpha_p(t) - \sum_{p=1}^{N_{CV}} \mathcal{V}_p \alpha_p^e(t), \tag{30}$$

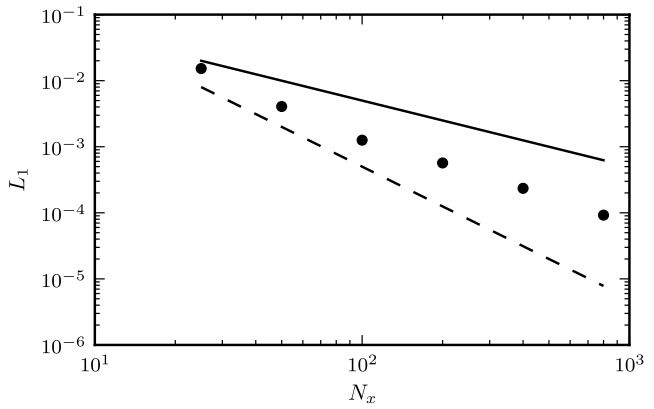



**Fig. 17.** Convergence of the $E_{\text{shape}}$ error at the end of the simulation for the transport of Zalesak's disk. Solid and dashed lines show first- and second-order convergence, respectively.

**Table 1**
Error norms and timing data for the transport of Zalesak's disk simulations.

| $N_x$ | $E_{\text{shape}}$ | $E_{\text{mass}}$ | $E_{\text{bound}}$ | Time/Timestep (s) |
|---|---|---|---|---|
| 25 | 1.526e−02 | 4.629e−18 | 3.526e−17 | 0.07 |
| 50 | 4.066e−03 | 3.011e−17 | 6.389e−18 | 0.16 |
| 100 | 1.257e−03 | 4.409e−18 | 9.588e−18 | 0.31 |
| 200 | 5.684e−04 | 3.705e−18 | 1.082e−17 | 0.66 |
| 400 | 2.348e−04 | 2.317e−18 | 1.227e−17 | 1.47 |
| 800 | 9.221e−05 | 1.937e−17 | 1.407e−17 | 3.38 |

$$E_{\text{bound}}(t) = \max\Big(-\min_{p=1,\ldots,N_{CV}} \mathcal{V}_p \alpha_p(t), \max_{p=1,\ldots,N_{CV}} \mathcal{V}_p\big(\alpha_p(t)-1\big)\Big) \tag{31}$$

and

$$E_{\text{shape}}(t) = \sum_{p=1}^{N_{CV}} \mathcal{V}_p \big|\alpha_p(t) - \alpha_p^e(t)\big|, \tag{32}$$

where $\alpha_p(t)$ and $\alpha_p^e(t)$ are the computed and exact liquid volume fraction within the $p$th computational cell at time $t$, respectively. $E_{\text{mass}}$ provides a measure of how the amount of liquid mass within the domain compares to the reference solution. $E_{\text{bound}}$ is an error norm that measures overshoots or undershoots of $\alpha$. $E_{\text{shape}}(t)$ depends on the distribution of the liquid within the domain and provides an error for the liquid shape at time $t$ [16,17]. The errors are expected to increase throughout the simulation, hence the errors are computed at the end of the simulation and are indicated with the shorthand notation $E_{\text{mass}}$, $E_{\text{bound}}$, and $E_{\text{shape}}$, respectively.

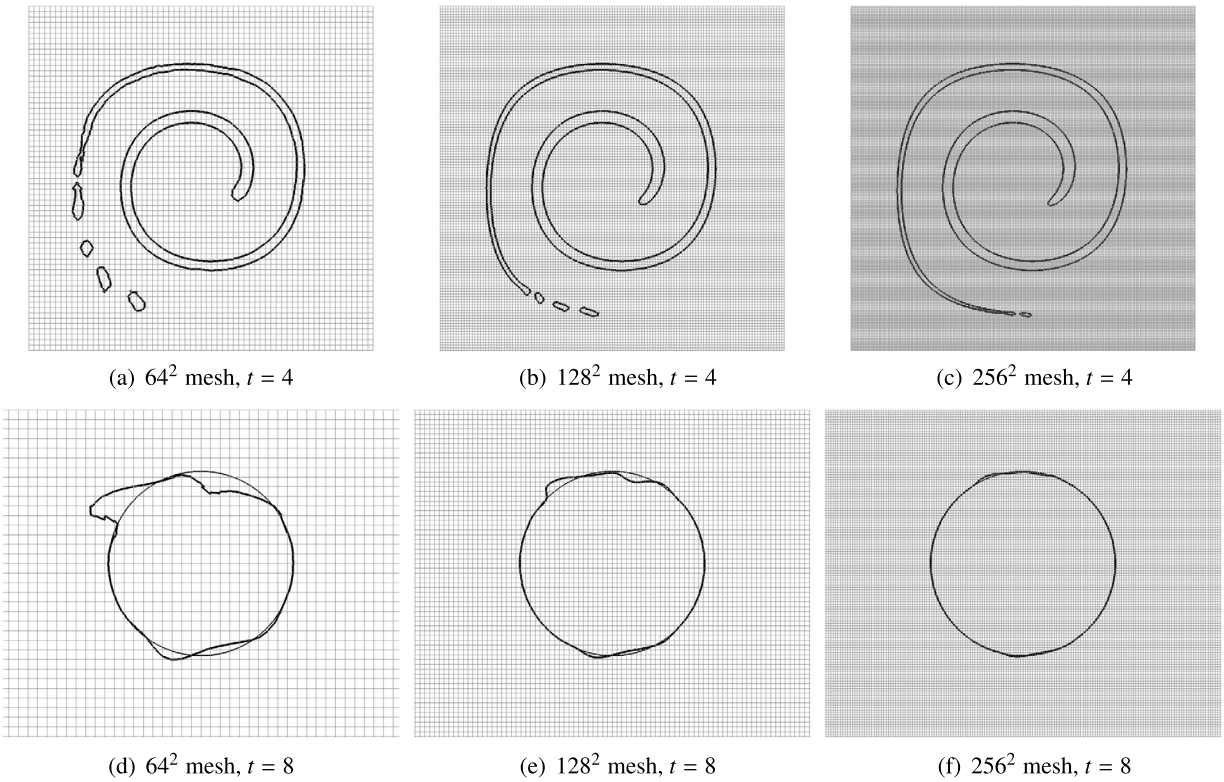
Fig. 17 shows how the $E_{\text{shape}}$ error converges under mesh refinement. A convergence rate between first- and second-order is observed. It is likely that the sharp corners in the solution reduce the convergence rate from the expected second-order, which is the rate observed in all other verification tests below. Table 1 provides the mass and boundedness errors, which remain at machine precision for all of the meshes.

Table 1 also reports timing data. All the simulations in this paper are performed using compute nodes with dual 6-core X5670 3 GHz CPUs with 48 GB of RAM. The results show the average time per time-step averaged throughout the simulation. One compute node is used for each simulation of Zalesak's disk. Note that all of the timing results are computed using a three-dimensional implementation of the proposed method. The two-dimensional tests are performed using a one-cell thick three-dimensional computational domain. An optimized two-dimensional implementation is expected to be more efficient for the two-dimensional tests since the three-dimensional implementation uses more simplices to represent the flux volumes and requires cutting by more planes that comprise the computational mesh.

### 4.2. Two-dimensional deformation

This test case, proposed by Leveque [32], consists of stretching and un-stretching a disk in a vortex. The simulation is initialized with a two-dimensional disk of diameter 0.3 centered at $(x, y) = (0, 0.25)$ within a unit square domain $[-0.5, 0.5]^2$. The disk is stretched using

$$\begin{aligned} u &= -2\sin^2(\pi x)\sin(\pi y)\cos(\pi y)\cos(\pi t/8), \\ v &= +2\sin^2(\pi y)\sin(\pi x)\cos(\pi x)\cos(\pi t/8). \end{aligned} \tag{33}$$



(a) $64^2$ mesh, $t = 4$ (b) $128^2$ mesh, $t = 4$ (c) $256^2$ mesh, $t = 4$

(d) $64^2$ mesh, $t = 8$ (e) $128^2$ mesh, $t = 8$ (f) $256^2$ mesh, $t = 8$

**Fig. 18.** Effect of mesh size on two-dimensional deformation test case. The top images show the disk at maximum deformation. The bottom images show the result at the end of the simulation with the thick line and the thin line indicates the initial condition.

The velocity field stretches the disk until time is $t = 4$; then, the velocity field is reversed for another four time units and the disk returns to its initial state.

Fig. 18 shows snapshots of the disk in the fully stretched state ($t = 4$) and at the end of the simulation ($t = 8$) on various meshes. On the coarsest mesh, the liquid in the tail region reaches the resolution limit of the mesh and the tail breaks into a series of droplets. This causes the liquid to move away from the reference solution, and the final shape does not match the expected solution. Note that this behavior is expected for methods with good mass conservation properties [6]. On the finest mesh, very little of the liquid in the tail is moved into droplets and the final shape matches the exact solution very well.

Fig. 19 provides quantitative results and shows the $E_{\text{shape}}$ error. Second-order convergence and small values are obtained for the shape error at the end of the simulation. Table 2 provides the mass and boundedness errors which remain at machine precision for all of the meshes. Additionally, Table 2 provides timing data for the simulations which are each performed with one compute node.

Table 3 provides a comparison with results reported by López et al. [16] obtained using EMFPA. EMFPA is very similar to the proposed method, but is limited to two dimensions. Table 3 shows similar shape errors at times $t = 0.5$ and $t = 2$ for both approaches. Note that the reference solution used for computing the shape errors is obtained on an $N_x = 1024$ mesh.

### 4.3. Three-dimensional deformation

This test case is similar to the two-dimensional deformation test case and was also proposed by Leveque [32]. It focuses on the behavior of the VOF scheme when a liquid sheet becomes under-resolved. The simulation is initialized with a droplet of diameter 0.3 centered at $(x, y, z) = (0.35, 0.35, 0.35)$ within a cube domain $[0, 1]^3$. The droplet is stretched until $t = 1.5$, then the velocity field is reversed and the liquid is un-stretched until $t = 3$. The velocity field used to stretch and un-stretch the droplet is

$$\begin{aligned} u &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/3), \\ v &= -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(\pi t/3), \\ w &= -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(\pi t/3). \end{aligned} \tag{34}$$

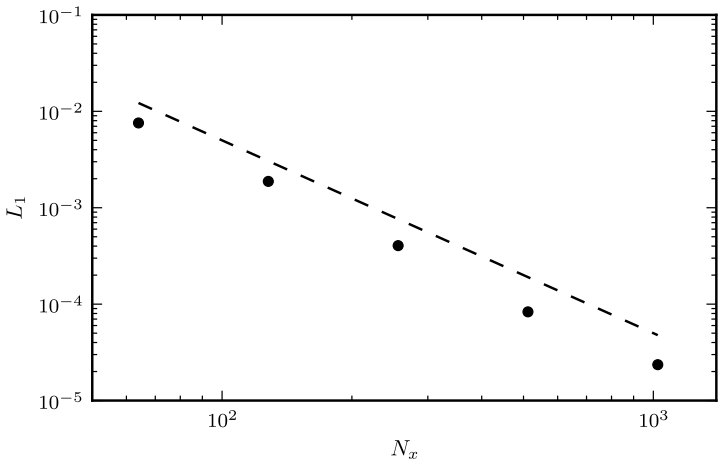



**Fig. 19.** Convergence of the $E_{\text{shape}}$ error at the end of the two-dimensional deformation test. Dashed line shows second-order convergence.

**Table 2**
Error norms and timing data for the two-dimensional deformation test.

| $N_x$ | $E_{\text{shape}}$ | $E_{\text{mass}}$ | $E_{\text{bound}}$ | Time/Timestep (s) |
|---|---|---|---|---|
| 64 | 7.576e−03 | 9.755e−15 | 6.517e−17 | 0.01 |
| 128 | 1.876e−03 | 1.290e−14 | 6.328e−17 | 0.04 |
| 256 | 4.045e−04 | 1.392e−14 | 8.741e−17 | 0.15 |
| 512 | 8.320e−05 | 1.736e−14 | 9.325e−17 | 0.62 |
| 1024 | 2.356e−05 | 1.678e−14 | 1.043e−16 | 2.41 |

**Table 3**
Shape error of proposed scheme compared with EMFPA of López et al. [16] for the two-dimensional deformation test. The error norm is evaluated at $t = 0.5$ and $t = 2$.

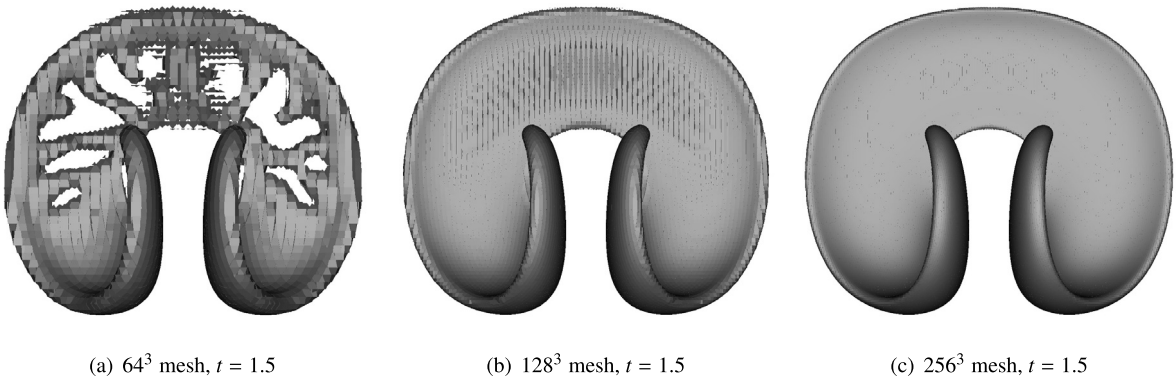
| $N_x$ | $E_{\text{shape}}(t = 0.5)$ | | $E_{\text{shape}}(t = 2)$ | |
|---|---|---|---|---|
| | EMFPA [16] | Proposed | EMFPA [16] | Proposed |
| 32 | 2.93e−03 | 1.58e−03 | 1.22e−02 | 2.00e−02 |
| 64 | 7.58e−04 | 4.43e−04 | 3.35e−03 | 3.33e−03 |
| 128 | 1.75e−04 | 1.19e−04 | 7.95e−04 | 8.90e−04 |

Fig. 20 shows snapshots of the gas–liquid interface at maximum stretching ($t = 1.5$) and at the end of the simulation ($t = 3$), when the droplet shape should match the initial condition. At maximum stretching, a thin sheet is formed that falls below the resolution of the mesh when a $64^3$ mesh is used. The method moves the under-resolved liquid from the sheet into resolvable structures. As the mesh is refined, less of the sheet becomes under-resolved, and on the $256^3$ mesh, the liquid sheet is maintained. Hence, discrepancies in the final shape are reduced with mesh refinement.

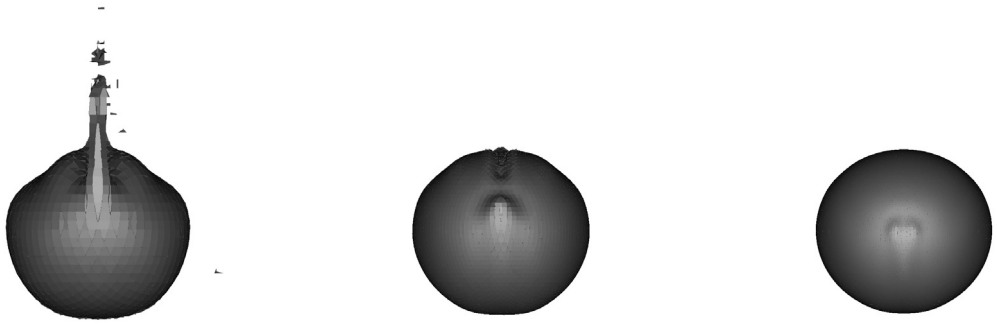
As shown in Fig. 21, second-order convergence is obtained for the $E_{\text{shape}}$ error. In Table 4 the $E_{\text{shape}}$ error is compared with results provided by Hernández et al. [17] using the FMFPA-3D scheme. The proposed method and FMFPA-3D produce very similar results with slightly lower errors using the proposed method. This is expected since both methods are unsplit geometric formulations. However, in addition to the lower errors, the proposed scheme provides discrete conservation, as indicated by $E_{\text{mass}}$ and $E_{\text{bound}}$ in Table 4. The table also provides timing data for the simulations performed using the proposed method. The simulations are performed on four compute nodes.

### 4.4. Droplet in homogeneous isotropic turbulence

This numerical experiment is designed to test the performance of the proposed scheme in a more realistic flow situation. The test consists of the deformation of a three-dimensional droplet in a complex velocity field. The velocity field is created from an instantaneous snapshot of synthetic homogeneous isotropic turbulence, denoted by $\boldsymbol{u}_0$. This solenoidal velocity field is created in spectral space from a Passot–Pouquet model spectrum. The same velocity field is used for all of the test cases presented below. Using that velocity field, the droplet is deformed for 1.5 time units; then, the velocity is reversed for another 1.5 time units. This is achieved using a temporally varying cosine function, i.e.,



(a) $64^3$ mesh, $t = 1.5$ (b) $128^3$ mesh, $t = 1.5$ (c) $256^3$ mesh, $t = 1.5$



(d) $64^3$ mesh, $t = 3.0$ (e) $128^3$ mesh, $t = 3.0$ (f) $256^3$ mesh, $t = 3.0$

**Fig. 20.** PLIC interface for the droplet in three-dimensional deformation flow on various meshes. Snapshots on the top show the droplet at maximum deformation ($t = 1.5$). The droplets at the end of the simulation ($t = 3$) are shown on the bottom.

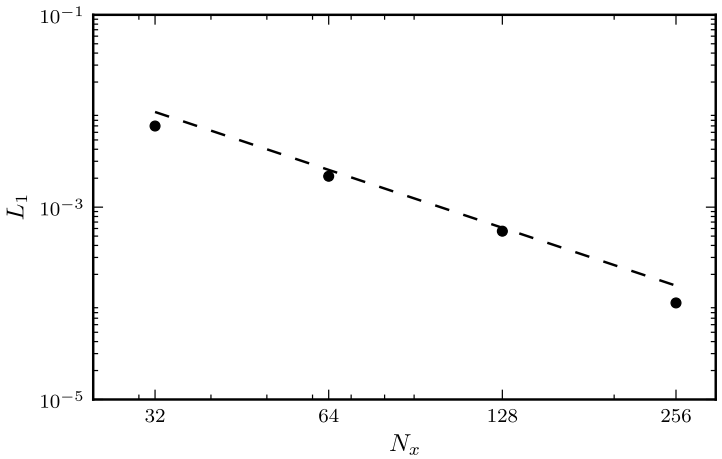
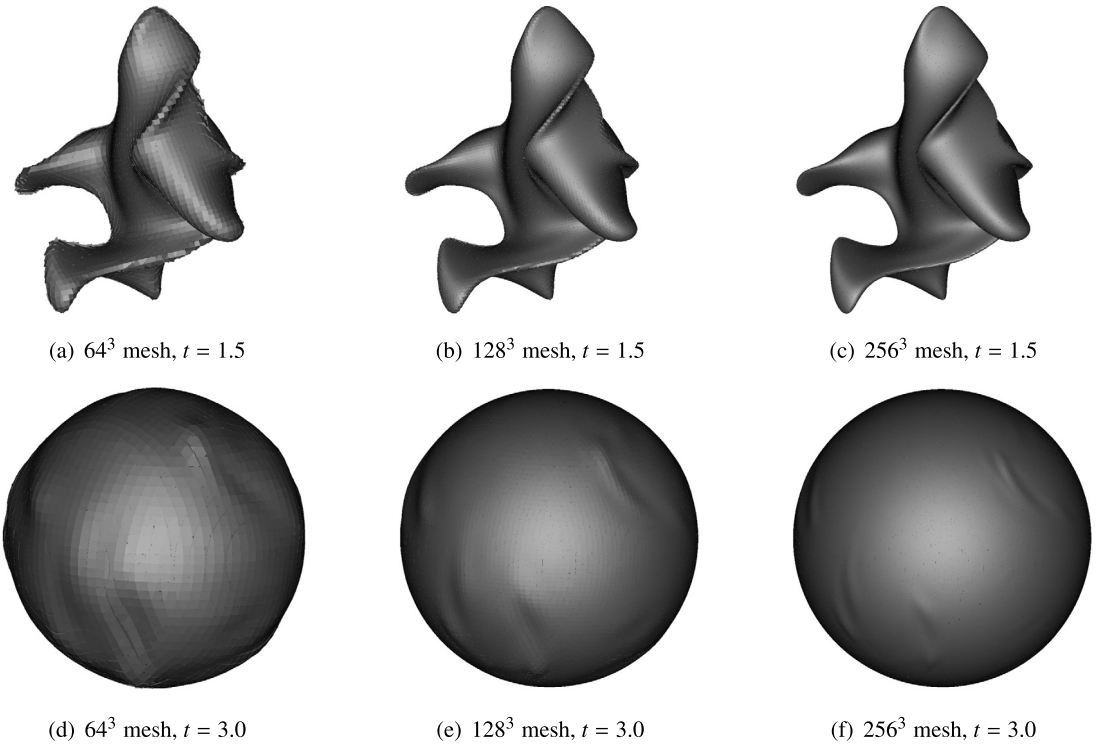



**Fig. 21.** Convergence of the $E_{\text{shape}}$ error for the three-dimensional deformation simulations. The dashed line shows second-order convergence.

**Table 4**
Comparison of $E_{\text{shape}}$ errors at end of three-dimensional deformation test using proposed method and those reported in Hernández et al. [17]. An $E_{\text{shape}}$ error on the $256^3$ mesh was not provided by Hernández et al. The table also provides the average time per time-step for the simulations performed using the proposed code. Mass and boundedness errors are also provided.

| $N_x$ | $E_{\text{shape}}$ | | $E_{\text{mass}}$ | $E_{\text{bound}}$ | Time/time-step (s) |
|---|---|---|---|---|---|
| | FMFPA-3D [17] | Proposed | | | |
| 32 | 7.440e−03 | 6.978e−03 | 1.194e−15 | 1.202e−17 | 0.78 |
| 64 | 2.790e−03 | 2.096e−03 | 2.479e−15 | 2.341e−17 | 2.85 |
| 128 | 7.140e−04 | 5.625e−04 | 1.675e−14 | 2.752e−17 | 12.2 |
| 256 | – | 1.010e−04 | 3.870e−14 | 4.690e−17 | 45.5 |



(a) $64^3$ mesh, $t = 1.5$ (b) $128^3$ mesh, $t = 1.5$ (c) $256^3$ mesh, $t = 1.5$

(d) $64^3$ mesh, $t = 3.0$ (e) $128^3$ mesh, $t = 3.0$ (f) $256^3$ mesh, $t = 3.0$

**Fig. 22.** Gas-liquid interface for the droplet in homogeneous isotropic turbulence test on various meshes. Snapshots on the top show the droplet at maximum deformation ($t = 1.5$). The droplets at the end of the simulation ($t = 3$) are shown on the bottom.

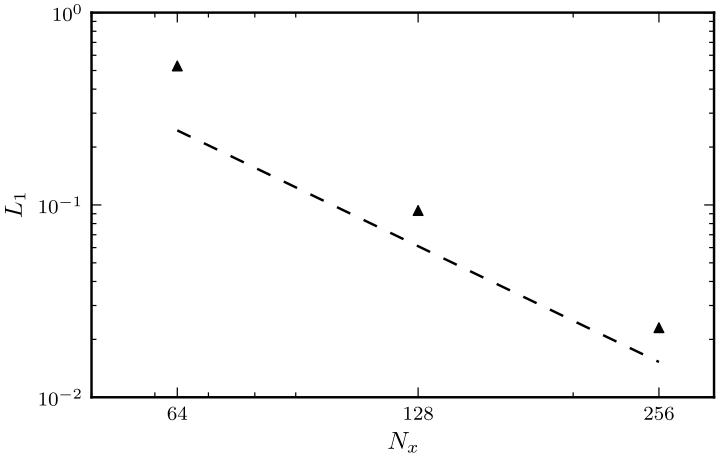



**Fig. 23.** Convergence of the $E_{\text{shape}}$ error for the simulation of the droplet in homogeneous isotropic turbulence. Dashed line shows second-order convergence.

$$\boldsymbol{u} = \boldsymbol{u}_0 \cos\left(\frac{\pi t}{3}\right). \tag{35}$$

The domain used for the simulation is $[0, 2\pi]^3$, and the droplet of diameter $\pi$ is initialized at $(x, y, z) = (\pi, \pi, \pi)$.

Fig. 22 shows the shape of the droplet after it has been deformed by the turbulence ($t = 1.5$), and at the end of the simulation, when the initial shape of the droplet should be recovered. The results are presented on three different meshes, namely $64^3$, $128^3$, and $256^3$. The overall qualitative shape agrees very well between the various cases at the middle and end of the simulations. Fig. 23 shows the convergence of the $E_{\text{shape}}$ error under mesh refinement, showing second-order

**Table 5**
Error norms and timing data for the droplet in homogeneous isotropic turbulence test case.

| $N_x$ | $E_{\text{shape}}$ | $E_{\text{mass}}$ | $E_{\text{bound}}$ | Time/Timestep (s) |
|---|---|---|---|---|
| 64 | 5.281e−01 | 5.472e−16 | 1.124e−17 | 1.61 |
| 128 | 9.357e−02 | 1.198e−15 | 1.198e−17 | 3.70 |
| 256 | 2.300e−02 | 1.290e−14 | 7.598e−17 | 12.0 |

accuracy. The mass and boundedness errors, shown in Table 5, remain at machine precision for all mesh levels. Timing data for this case can also be found in Table 5. These simulations are performed using four compute nodes.

## 5. Conclusions

In this paper, we have developed and tested a bounded, conservative, unsplit, three-dimensional geometric transport scheme that is applied to the piecewise linear interface calculation (PLIC) volume-of-fluid (VOF) method. The scheme leverages two key ideas that make it straightforward to implement. The first is the use of simplices to represent semi-Lagrangian flux volumes. The simplices are created using the same vertices for all flux volume geometries, which greatly simplifies the process of discretizing the complex shapes. The second idea is a simple sign convention for identifying if a simplex contributes positively or negatively to the flux. The scheme is verified using a collection of canonical test cases including Zalesak's disk, two- and three-dimensional deformation tests, and the deformation of a droplet in three-dimensional homogeneous isotropic turbulence. In all of the test cases, the method produced excellent results even on coarse meshes. Second-order convergence, discrete conservation, and boundedness are demonstrated.

## Appendix A. Additional algorithms

**Algorithm 5** CutSimplex: cuts a simplex by a plane and partitions resulting shapes into new simplices using look-up tables.

**function** CutSimplex($\boldsymbol{S}$, $\boldsymbol{P}$)
  **input** $\boldsymbol{S}$ ▷ Array of simplex vertices, i.e. $\boldsymbol{S} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4]$
  **input** $\boldsymbol{P}$ ▷ Plane equation coefficients, i.e. $\boldsymbol{P} = \{[a, b, c, d] \mid ax + by + cz = d\}$
  $\boldsymbol{Pt}(1:4, :) \leftarrow \boldsymbol{S}$ ▷ Copy simplex vertices into point array
  **for** $i = 1 \rightarrow 4$ **do**
    $\boldsymbol{d}(i) \leftarrow$ Distance($\boldsymbol{Pt}(i)$, $\boldsymbol{P}$) ▷ Calculate distance between point and plane
  **end for**
  $Case \leftarrow 1 + 1\ (\frac{1}{2} + \frac{1}{2}\,\text{sign}(\boldsymbol{d}(1)))$ ▷ Create case number (1–16) based on sign of distances
    $+ 2\ (\frac{1}{2} + \frac{1}{2}\,\text{sign}(\boldsymbol{d}(2)))$
    $+ 4\ (\frac{1}{2} + \frac{1}{2}\,\text{sign}(\boldsymbol{d}(3)))$
    $+ 8\ (\frac{1}{2} + \frac{1}{2}\,\text{sign}(\boldsymbol{d}(4)))$
  **for** $n = 1 \rightarrow$ NumberIntersect($Case$) **do** ▷ Loop over intersections between simplex edges and plane
    $I_1 \leftarrow$ IndexEndPoint($1, n, Case$) ▷ Get index of points on end of edge
    $I_2 \leftarrow$ IndexEndPoint($2, n, Case$)
    $\boldsymbol{Pt}(4 + n, :) \leftarrow \boldsymbol{Pt}(I_1, :) - \frac{d(I_1)}{d(I_2) - d(I_1)}(\boldsymbol{Pt}(I_2, :) - \boldsymbol{Pt}(I_1, :))$ ▷ Calculate intersection and append to points array
  **end for**
  $N_{\text{Out}} \leftarrow$ NumberSimsPart($Case$) ▷ Number of simplices in partition
  **for** $n = 1 \rightarrow N_{\text{Out}}$ **do**
    **for** $v = 1 \rightarrow 4$ **do**
      $\boldsymbol{S}_{\text{Out}}(n, :) \leftarrow \boldsymbol{Pt}($IndexSimVert($v, n, Case$)$, :)$ ▷ Create simplices in partition
    **end for**
  **end for**
  **return** $N_{\text{Out}}$ ▷ Number of simplices returned
  **return** $\boldsymbol{S}_{\text{Out}}$ ▷ Vertices of simplices returned
**end function**

**Algorithm 6** Look-up Tables: provide useful quantities based on the case number of the simplex.

▷ Number of intersections between simplex and plane

NumberIntersect ← [0, 3, 3, 4, 3, 4, 4, 3, 3, 4, 4, 3, 4, 3, 3, 0]

▷ Indices of endpoints on line that intersects plane

IndexEndPoint(:, :, 1) ← [[0, 0], [0, 0], [0, 0], [0, 0]]
IndexEndPoint(:, :, 2) ← [[1, 2], [1, 3], [1, 4], [0, 0]]
IndexEndPoint(:, :, 3) ← [[2, 3], [2, 4], [2, 1], [0, 0]]
IndexEndPoint(:, :, 4) ← [[1, 4], [2, 4], [1, 3], [2, 3]]
IndexEndPoint(:, :, 5) ← [[3, 4], [3, 1], [3, 2], [0, 0]]
IndexEndPoint(:, :, 6) ← [[1, 4], [3, 4], [1, 2], [3, 2]]
IndexEndPoint(:, :, 7) ← [[2, 4], [3, 4], [2, 1], [3, 1]]
IndexEndPoint(:, :, 8) ← [[4, 1], [4, 2], [4, 3], [0, 0]]
IndexEndPoint(:, :, 9) ← [[4, 1], [4, 2], [4, 3], [0, 0]]
IndexEndPoint(:, :, 10) ← [[1, 3], [4, 3], [1, 2], [4, 2]]
IndexEndPoint(:, :, 11) ← [[2, 3], [4, 3], [2, 1], [4, 1]]
IndexEndPoint(:, :, 12) ← [[3, 4], [3, 1], [3, 2], [0, 0]]
IndexEndPoint(:, :, 13) ← [[3, 2], [4, 2], [3, 1], [4, 1]]
IndexEndPoint(:, :, 14) ← [[2, 3], [2, 4], [2, 1], [0, 0]]
IndexEndPoint(:, :, 15) ← [[1, 2], [1, 3], [1, 4], [0, 0]]
IndexEndPoint(:, :, 16) ← [[0, 0], [0, 0], [0, 0], [0, 0]]

▷ Number of simplices in partition of original simplex

NumberSimsPart ← [1, 4, 4, 6, 4, 6, 6, 4, 4, 6, 6, 4, 6, 4, 4, 1]

▷ Indices of vertices used to partition simplex

IndexSimVert(:, :, 1) ← [[1, 2, 3, 4], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 2) ← [[1, 5, 6, 7], [4, 2, 3, 6], [4, 2, 5, 6], [4, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 3) ← [[2, 5, 6, 7], [1, 3, 4, 6], [1, 3, 5, 6], [1, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 4) ← [[5, 6, 8, 2], [5, 7, 8, 1], [5, 8, 1, 2], [5, 6, 8, 4], [5, 7, 8, 3], [5, 8, 4, 3]]
IndexSimVert(:, :, 5) ← [[3, 5, 6, 7], [2, 4, 1, 6], [2, 4, 5, 6], [2, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 6) ← [[5, 6, 8, 3], [5, 7, 8, 1], [5, 8, 1, 3], [5, 6, 8, 4], [5, 7, 8, 2], [5, 8, 4, 2]]
IndexSimVert(:, :, 7) ← [[5, 6, 8, 3], [5, 7, 8, 2], [5, 8, 2, 3], [5, 6, 8, 4], [5, 7, 8, 1], [5, 8, 4, 1]]
IndexSimVert(:, :, 8) ← [[1, 2, 3, 7], [1, 2, 6, 7], [1, 5, 6, 7], [4, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 9) ← [[4, 5, 6, 7], [3, 1, 2, 6], [3, 1, 5, 6], [3, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 10) ← [[5, 6, 8, 4], [5, 7, 8, 1], [5, 8, 1, 4], [5, 6, 8, 3], [5, 7, 8, 2], [5, 8, 3, 2]]
IndexSimVert(:, :, 11) ← [[5, 6, 8, 4], [5, 7, 8, 2], [5, 8, 2, 4], [5, 6, 8, 3], [5, 7, 8, 1], [5, 8, 3, 1]]
IndexSimVert(:, :, 12) ← [[4, 1, 2, 7], [4, 1, 6, 7], [4, 5, 6, 7], [3, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 13) ← [[5, 6, 8, 4], [5, 7, 8, 3], [5, 8, 3, 4], [5, 6, 8, 2], [5, 7, 8, 1], [5, 8, 2, 1]]
IndexSimVert(:, :, 14) ← [[3, 4, 1, 7], [3, 4, 6, 7], [3, 5, 6, 7], [2, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 15) ← [[2, 3, 4, 7], [2, 3, 6, 7], [2, 5, 6, 7], [1, 5, 6, 7], [0, 0, 0, 0], [0, 0, 0, 0]]
IndexSimVert(:, :, 16) ← [[1, 2, 3, 4], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

## References


[1] W. Dettmer, P.H. Saksono, D. Perić, On a finite element formulation for incompressible newtonian fluid flows on moving domains in the presence of surface tension, Commun. Numer. Methods Eng. 19 (9) (2003) 659–668.
[2] M. Rudman, Volume-tracking methods for interfacial flow calculations, Int. J. Numer. Methods Fluids 24 (7) (1997) 671–691.
[3] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations, J. Comput. Phys. 79 (1) (1988) 12–49.
[4] C. Hirt, B. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, J. Comput. Phys. 39 (1) (1981) 201–225.
[5] E. Olsson, G. Kreiss, A conservative level set method for two phase flow, J. Comput. Phys. 210 (1) (2005) 225–246.
[6] O. Desjardins, V. Moureau, H. Pitsch, An accurate conservative level set/ghost fluid method for simulating turbulent atomization, J. Comput. Phys. 227 (18) (2008) 8395–8416.
[7] M. Owkes, O. Desjardins, A discontinuous Galerkin conservative level set scheme for interface capturing in multiphase flows, J. Comput. Phys. (2014), in press.
[8] J. McCaslin, O. Desjardins, A localized re-initialization equation for the conservative level set method, J. Comp. Phys. 262 (2014) 408–426.
[9] R. DeBar, Fundamentals of the KRAKEN code, Tech. Rep. UCIR-760, LLNL, 1974.
[10] B. Nichols, C. Hirt, Methods for calculating multi-dimensional, transient free surface flows past bodies, Tech. Rep. LA-UR-75-1932, Los Alamos National Laboratory, 1975.
[11] W.F. Noh, P. Woodward, SLIC (simple line interface calculation), in: Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics, in: Lect. Notes Phys., vol. 59, Springer, Berlin, Heidelberg, 1976, pp. 330–340.
[12] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas–Liquid Multiphase Flows, Cambridge University Press, 2011.
[13] D. Youngs, Time-dependent multi-material flow with large fluid distortion, Numer. Methods Fluid Dyn. (1982) 273–285.
[14] W.J. Rider, D.B. Kothe, Reconstructing volume tracking, J. Comput. Phys. 141 (2) (1998) 112–152.
[15] J.E. Pilliod, E.G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces, J. Comput. Phys. 199 (2) (2004) 465–502.
[16] J. López, J. Hernández, P. Gómez, F. Faura, A volume of fluid method based on multidimensional advection and spline interface reconstruction, J. Comput. Phys. 195 (2) (2004) 718–742.
[17] J. Hernández, J. López, P. Gómez, C. Zanzi, F. Faura, A new volume of fluid method in three dimensions – Part I: Multidimensional advection method with face-matched flux polyhedra, Int. J. Numer. Methods Fluids 58 (8) (2008) 897–921.
[18] H.T. Ahn, M. Shashkov, Multi-material interface reconstruction on generalized polyhedral meshes, J. Comput. Phys. 226 (2) (2007) 2096–2132.
[19] V. Le Chenadec, H. Pitsch, A 3D unsplit Forward/Backward volume-of-fluid approach and coupling to the level set method, J. Comput. Phys. 233 (15) (2013) 10–33.

[20] V. Le Chenadec, H. Pitsch, A monotonicity preserving conservative sharp interface flow solver for high density ratio two-phase flows, J. Comput. Phys. 249 (2013) 185–203.
[21] T. Marić, H. Marschall, D. Bothe, voFoam – a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM, arXiv e-print, arXiv:1305.3417.
[22] C.B. Ivey, P. Moin, Conservative volume of fluid advection method on unstructured grids in three dimensions, Center Turb. Res. Ann. Res. Briefs (2012) 179–192.
[23] J. López, J. Hernández, Analytical and geometrical tools for 3D volume of fluid methods in general grids, J. Comput. Phys. 227 (12) (2008) 5939–5948.
[24] D.B. Kothe, W.J. Rider, S.J. Mosso, J.S. Brock, J.I. Hochstein, Volume tracking of interfaces having surface tension in two and three dimensions, Tech. rep. 1996.
[25] R. Scardovelli, S. Zaleski, Analytical relations connecting linear interfaces and volume fractions in rectangular grids, J. Comput. Phys. 164 (1) (2000) 228–237.
[26] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes, The Art of Scientific Computing, 3rd edition, Cambridge University Press, 2007.
[27] D.M.Y. Sommerville, An Introduction to the Geometry of *n* Dimensions, Dover Publications, 1958.
[28] P. Liovic, M. Rudman, J.-L. Liow, D. Lakehal, D. Kothe, A 3D unsplit-advection volume tracking algorithm with planarity-preserving interface reconstruction, Comput. Fluids 35 (10) (2006) 1011–1032.
[29] J. Mencinger, I. Žun, A PLIC-VOF method suited for adaptive moving grids, J. Comput. Phys. 230 (3) (2011) 644–663.
[30] O. Desjardins, G. Blanquart, G. Balarac, H. Pitsch, High order conservative finite difference scheme for variable density low Mach number turbulent flows, J. Comput. Phys. 227 (15) (2008) 7125–7159.
[31] S.T. Zalesak, Fully multidimensional flux-corrected transport algorithms for fluids, J. Comput. Phys. 31 (3) (1979) 335–362.
[32] R.J. Leveque, High-resolution conservative algorithms for advection in incompressible flow, SIAM J. Numer. Anal. 33 (2) (1996) 627–665, ArticleType: research-article/Full publication date: Apr., 1996/Copyright © 1996 Society for Industrial and Applied Mathematics.