Iterative Volume-of-Fluid interface positioning in general polyhedrons with Consecutive Cubic Spline interpolation

Tomislav Marić

Mathematical Modeling and Analysis, Technische Universität Darmstadt

Abstract

A straightforward and computationally efficient Consecutive Cubic Spline (CCS) iterative algorithm is proposed for positioning the planar interface of the unstructured geometrical Volume-of-Fluid method in arbitrarily-shaped cells. The CCS algorithm is a two-point root-finding algorithm [1, chap. 2], specifically designed for the VOF interface positioning problem, where the volume fraction function has diminishing derivatives at the ends of the search interval. As a two-point iterative algorithm, CCS re-uses function values and derivatives from previous iterations and does not rely on interval bracketing. The CCS algorithm only requires only two iterations on average to position the interface with a tolerance of $10^{-12}$, even with numerically very challenging volume fraction values, e.g. near $10^{-9}$ or $1 - 10^{-9}$.

The proposed CCS algorithm is very straightforward to implement because its input is already calculated by every geometrical VOF method. It builds upon and significantly improves the predictive Newton method [2] and is independent of the cell’s geometrical model and related intersection algorithm. Geometrical parametrizations of truncated volumes used by other contemporary methods [3, 4, 5, 6] are completely avoided. The computational efficiency is comparable in terms of the number of iterations to the fastest methods reported so far. References are provided in the results section to the open source implementation of the CCS algorithm and the performance measurement data.

Keywords:
volume of fluid, interface reconstruction, iterative positioning, unstructured meshes

*maric@mma.tu-darmstadt.de
1. Introduction

An essential task of any multiphase simulation method is the identification and tracking of immiscible fluid phases, schematically shown in fig. [1a]. Each phase with distinctive physical properties can be associated with a subset \( \Omega^\pm(t) \) of the domain \( \Omega = \Omega^+(t) \cup \Omega^-(t) \), separated by the fluid interface \( \Sigma(t) \) with outward-oriented normal vectors \( \mathbf{n}_\Sigma \). An indicator function is introduced to distinguish \( \Omega^\pm(t) \), as

\[
\chi(t, \mathbf{x}) = \begin{cases} 
1 & \text{if } \mathbf{x} \in \Omega^+(t) \\
0 & \text{if } \mathbf{x} \in \Omega^-(t) 
\end{cases}
\tag{1}
\]

It is impossible to calculate \( \chi(t, \mathbf{x}) \) analytically when the interface motion is driven by a complex two-phase flow velocity. Therefore, approximations such as those shown in fig. [1b] are used, specifically by the geometrical Volume-of-Fluid method, to approximately calculate \( \Omega^\pm(t) \). The domain \( \Omega \) is discretized as a union of non-overlapping polyhedrons \( V_c \), such that \( \Omega = \bigcup_c V_c, c \in C \). For each polyhedron \( V_c \), a volume fraction is defined as

\[
\alpha_c(t) := \int_{V_c} \chi(t, \mathbf{x}) \, dV,
\tag{2}
\]

or, in other words, a fill level of the polyhedron \( V_c \) with \( 0 \leq \alpha_c \leq 1 \). The Piecewise Linear Interface Calculation (PLIC) [7, 8, 9] is still prevalently used by the geometrical VOF method to approximate the indicator function \( \chi(t, \mathbf{x}) \) in each \( V_c \) using
\[ \chi(t, x) \approx H_c := H_c(p_c(t), n_c(t)) = \begin{cases} 1 \text{ if } (x - p_c(t)) \cdot n_c(t) \geq 0, \\ 0 \text{ otherwise.} \end{cases} \]  

(3)

where \( H_c(p_c(t), n_c(t)) \) is the positive halfspace of the PLIC plane given by the plane position \( p_c \) and plane normal \( n_c \) at time \( t \). The interface reconstruction algorithm computes the halfspace \( H_c(p_c(t), n_c(t)) \), by approximating the interface unit normal vector \( n_c \) and computing the point \( p_c \), such that the intersection between the positive halfspace and the polyhedron \( V_c \) satisfies

\[ \alpha_c(t) := \int_{V_c} \chi(t, x) \, dV = \frac{|V_c \cap H_c(p_c(t), n_c(t))|}{|V_c|}. \]  

(4)

The computation of \( p_c \) in each \( V_c \) is the interface positioning part of the reconstruction algorithm, given \( \{n_c\}_{c \in C} \), \( \{\alpha_c\}_{c \in C} \), under the condition \( 0 < \alpha_c < 1 \), and a polyhedral domain discretization \( \Omega = \bigcup_c V_c, c \in C \). Since the goal of the interface positioning is to find \( p_c(t) \) at a fixed \( t \), \( t \) can be disregarded from now on.

---

An explicit (closed-form) solution of eq. (4) for \( p_c \) does not exist for an arbitrary volume \( V_c \). For example, the positioning of the PLIC plane in an endo-dodecahedron\(^1\) from fig. 2a is shown in fig. 2b with \( \alpha_c = 0.5 \) and the normal vector \( n_c \) that is

---

\(^{1}\)A model for a non-convex dodecahedron with planar faces.
collinear with the Z-axis. The resulting intersection between the positive halfspace of the PLIC plane \( H_c(n_c, p_c) \) and the endo-dodecahedron is the shaded volume in fig. 2b. To simplify the solution process for \( p_c \) in eq. 4, the position of \( p_c \) on the \( n_c \) axis can be parameterized with a scalar \( s \), as shown in fig. 2b. Minimal and maximal point \( x_{min, max} \) of the volume \( V_c \), projected onto \( n_c \), as shown in fig. 2b, are defined as

\[
\begin{align*}
x_{min} &= x_{ref} + \min_p ((x_p - x_{ref}) \cdot n_c) \cdot n_c, \quad p \in P, \\
x_{max} &= x_{ref} + \max_p ((x_p - x_{ref}) \cdot n_c) \cdot n_c, \quad p \in P,
\end{align*}
\]

(5)

where \( x_{ref} \) is a reference point for the \( s \)-axis. Any point can be chosen as \( x_{ref} \) and usually the origin of the coordinate system is used. However, using the origin as a reference point increases the error of floating-point calculations used in eq. (5), and using a point that belongs to \( V_c \) (e.g. the first point, or the centroid of \( V_c \)) reduces these errors [10, section 2]. The set \( P \) in eq. (5) is the set all points \( x_p \) of the volume \( V_c \). Using eq. (5), the halfspace position \( p_c \) is parameterized with

\[
\begin{align*}
H_c(s) &= H_c(p_c(s), n_c) \\
p_c(s) &= x_{min} + s n_c,
\end{align*}
\]

(6)

(7)

where \( s \in [s_{min}, s_{max}] \). Note that the parametrization determines the values of the parameter \( s \); for example, with the \( x_{min}, x_{max} \), parametrization by eq. (7), \( s \in [0, s_{max}] \). This, in turn, leads to the parametrization of the intersection volume in eq. (4) as

\[
V(s) = V_c \cap H_c(p_c(s), n_c).
\]

(8)

The parameterization of the intersection volume reformulates eq. (4) as

\[
\tilde{\alpha}_c(s) := \frac{|V(s)|}{|V_c|} - \alpha_c = \alpha_c(s) - \alpha_c = 0,
\]

(9)

where \( \alpha_c \) is the given volume fraction for \( V_c \). The root \( s^* \) of \( \tilde{\alpha}(s) \) is sought to compute \( p_c(s^*) \), the position of the PLIC plane \( (p_c(s^*), n_c) \) in \( V_c \) such that eq. (4) is satisfied, as shown in fig. 2b.

The function \( \alpha(s) \) is visualized in fig. 3 for the endo-dodecahedron shown in fig. 2a, using \( n_c = (0, 0, 1) \) and \( \alpha^* \in [0, \alpha_1^*, \alpha_2^*, \ldots, \alpha_i^*, 1] \), \( \alpha_i^* = \frac{i}{N} \), \( N = 100 \). If \( V_c \) is non-convex, interface positioning sometimes separates \( V_c \) into disjoint volumes, similar to the result shown in fig. 3b for \( s \approx 2 \). Geometrical operations used for the intersection must therefore support disjoint sets. Non-convex cells with non-planar faces are decomposed into tetrahedrons using the centroid of the cell and centroids
of the non-planar faces. The magnitude of the truncated volume that lies inside the halfspace $H_c(p_c(s), n_c)$ is, therefore, equal to the sum of volume magnitudes of the truncated tetrahedrons. The tetrahedral decomposition is the most straightforward approach to the volume truncation of non-convex cells with non-planar faces [11], but it is more computationally expensive, compared to sophisticated approaches such as [6]. The choice of the truncation is arbitrary for the proposed CCS method: an alternative volume truncation reduces the total CPU time, however, it has no influence on the total number of iterations, whose very straightforward reduction the main contribution of the proposed method.

Following boundary conditions are given for $\tilde{\alpha}(s)$:

$$\tilde{\alpha}(s_{\min}) = 1 - \alpha_c,$$

$$\tilde{\alpha}'(s_{\min}) = 0,$$

$$\tilde{\alpha}(s_{\max}) = -\alpha_c,$$

$$\tilde{\alpha}'(s_{\max}) = 0.$$  \hspace{1cm} (13)

Boundary conditions given by eq. (13) complicate the root finding in eq. (9), because
the derivatives of $\tilde{\alpha}(s)$ are diminishing at $s = s_{\text{min}}, s_{\text{max}}$. Furthermore, although highly improbable, it is possible that the normal vector $\mathbf{n}_c$ becomes collinear with the normal of a planar face of the cell. In this case either $\tilde{\alpha}'(s_{\text{min}}) \neq 0$ or $\tilde{\alpha}'(s_{\text{max}}) \neq 0$. This special case of the boundary condition is handled by the proposed CCS method and addressed in detail below.

Positioning algorithms can be categorized as iterative or bracketing algorithms. Iterative algorithms rely solely on root-finding to solve eq. (9) up to a prescribed tolerance. Bracketing algorithms intersect the volume $V_c$ incrementally, until a bracketed volume is found that contains the interface. In the bracketed volume, modern positioning algorithms \cite{4, 5, 6} exactly position the interface using an exact function for the volume fraction within the bracketed interval.

Kothe et al. \cite{12}, Rider and Kothe \cite{9} have proposed an iterative algorithm that relied on Brent’s root finding method, combining inverse quadratic interpolation with bisection to avoid divergence near interval boundaries. Bracketing intersects the volume $V_c$ with the halfspace given by $H(p_p, \mathbf{n}_c), p \in P$, where $\{p_p\}_{p \in P}$ are cell corner points. This brackets a volume $V_c$ as

$$B_{c,i,j} = H(p_i, \mathbf{n}_c) \cap V_c \setminus H(p_j, \mathbf{n}_c) \cap V_c, \quad s_i \leq s^* \leq s_j, p_{i,j} \in P,$$

resulting in the bracketed volume $B_{c,i,j}$ that contains the root $s^*$ of $\tilde{\alpha}$ given by eq. (9). Within the bracketed volume, Brent’s algorithm requires additional iterations to find the root of eq. (9) up to a prescribed tolerance. This procedure was widely adopted for interface positioning \cite{13, 14, 15, 16}. Scardovelli and Zaleski \cite{17} have proposed an analytical interface positioning method for rectangular meshes that (used in \cite{18}) and Yang and James \cite{19} have proposed an analytical positioning method for tetrahedral and triangular meshes.

López and Hernández \cite{20} have extended the work of Scardovelli and Zaleski \cite{17} and Yang and James \cite{19}. They have used and indexed face-set\footnote{An indexed face-set models a volume using a global set of unique points and its polygonal boundary as a set of polygons, such that each boundary polygon is modeled as a sequence of indexes of unique points.} as the boundary representation of a polyhedron Ghali \cite[Chapter 28]{21} and an analytical expression for a volume of a convex polyhedron \cite{22} to position the interface exactly within the bracketed interval. López and Hernández \cite{20} introduce the Central Sequential Bracketing (CSB) procedure that utilizes sorted signed distances associated with the corner points $i_p$ of the volume $V_c$, calculated using the interface normal $\mathbf{n}_c$. The goal of the algorithm is to locate indices $k_{\text{min}}, k_{\text{max}}$ in this list, such that $k_{\text{max}} - k_{\text{min}} = 1$, $|V_{T,k_{\text{max}}}| \geq \alpha_c |V_c|$ and $|V_{T,k_{\text{min}}}| \leq \alpha_c |V_c|$, where $V_{T,k}$ is the truncated volume given by
the plane normal $n_c$ and the vertex $i_p$ associated with the $k$-th signed distance in the list. The algorithm starts with the central index $k_c = \text{INT}[(I_p + 1)/2]$ in the signed distance list, and computes the truncated volume $V_T$ passing through this point. If $|V_T| > \alpha_c|V_c|$, $k_{\text{max}} = k_c$, otherwise $k_{\text{min}} = k_c$. The next iteration continues with the reduced list of indices between $[k_{\text{min}}, k_{\text{max}}]$. The authors state that the algorithm complexity in terms of the CPU time is $O(\frac{1}{4}I_p + 1)$ and $O(\log_2 I_p)$ if so-called Binary Bracketing (BB) is used to set $k_c = (k_{\text{min}} + k_{\text{max}})/2$, where $I_p$ is the total number of points of the volume $V_c$. They utilize BB for cells with $I_p > 8$ because then the logarithmic complexity outperforms the linear complexity of the CSB bracketing algorithm. López et al. [5] have published the source code that implements their methods from [20].

Ahn and Shashkov [23] have proposed a stabilized bisection-secant method as an iterative positioning algorithm, used by the author in [24, 25]. Their algorithm benefits from the super-linear (golden ratio) convergence of the secant method outside of the regions with diminishing derivatives, and relies on the bisection method to ensure convergence otherwise. They rely on the tetrahedral decomposition of the volume for the calculation of the truncated volume, however the root finding algorithm is independent of that choice.

Diot et al. [3], Diot and François [4] have proposed an analytical positioning method that does not rely on the analytical expression for the volume of the convex polygon (polyhedron). Instead, the polygon (polyhedron) is decomposed into sub-volumes whose magnitudes are exactly calculated using mixed vector and scalar products. The proposed method is very accurate, however it involves relatively complex subdivisions of the $V_c$ volume.

López et al. [26] present a detailed review of interface positioning algorithms and propose an enhancement of their CSB and BB bracketing algorithms from [20] on convex polyhedral cells. Their new Interpolation Bracketing (IB) algorithm uses signed distances to linearly interpolate $H(p_c(s), n_c)$, in effect interpolating linearly the volume fraction function shown in fig. [33] The authors couple their IB algorithm with a new explicit function for interface positioning in the bracketed interval. The authors state that the explicit positioning function costs as much as 1.7 times the volume truncation operation, which must be taken into account when its overall computational complexity is considered. The new explicit function is combined with the IB algorithm into the new Coupled Interpolation-Bracketed Analytical Volume Enforcement (CIBRAVE) method.

Recently, Chen and Zhang [2] have noticed that the fundamental theorem of calculus can be used to increase the convergence of the iterative positioning algorithm. In fig. [4] the intersection of the plane $(p(s), n_c)$ and the volume $V_c$ is the polygon with
the area $A(s)$ that lies on the plane $(p(s), n_c)$: the so-called *cap polygon*. The area of this polygon ($\text{cap area}$) is computed geometrically, with machine-epsilon accuracy, at any $s$, and it can be used to integrate parametrized volume $V(s)$ as

$$V(s_2) = V(s_1) - \int_{s_1}^{s_2} A(s) ds = V(s_1) - \int_{s_1}^{s_2} V'(s) ds, \quad s_1 < s_2.$$  \hfill (15)

It is obvious that $A(s) = V'(s)$ is the exact derivative of the volume $V(s)$ from eq. (9), whose root determines the position of the interface $p(s^*)$. A direct consequence of this is

$$\alpha'(s) = \frac{|V'(s)|}{|V_c|} = \frac{|A(s)|}{|V_c|}. \hfill (16)$$

Chen and Zhang \cite{2} use the exact derivative $\alpha'(s)$ in the Newton’s root finding method to solve eq. (9). Knowing the exact derivative ensures the quadratic convergence of the Newton’s method, provided that the condition $\alpha'(s) \neq 0$ is fulfilled. However, it does not solve issue of divergence of the Newton’s method near interval boundaries $s_{\text{min}}, s_{\text{max}}$. If an iteration of the Newton’s method computes a root outside of $(s_{\text{min}}, s_{\text{max}})$, Chen and Zhang \cite{2, Algorithm 1} apply the Hermite cubic spline
interpolation at \( s \), across \((s_{\text{min}}, s_{\text{max}})\). Unfortunately, interpolating across the whole search interval \((s_{\text{min}}, s_{\text{max}})\) introduces large interpolation errors and substantially slows down convergence for volume fraction values \( \alpha_c \approx 0, \alpha \approx 1 \). To understand why, consider the case when \( \alpha_c \approx 1 \): the derivative \( \alpha'(s) \approx 0 \), Newton’s method shoots out of the search interval \( s_{\text{min}}, s_{\text{max}} \), and then Hermite interpolation over \( s_{\text{min}}, s_{\text{max}} \) is performed. If \( \alpha \approx 1 \), the root lies near \( s_{\text{min}} \). However, the Hermite interpolation introduces the conditions on \( \tilde{\alpha}(s) \) at \( s_{\text{max}} \), that lies on the other end of the search interval. Therefore, the accuracy is lost in this case, and vice versa for \( \alpha_c \approx 0 \). The method of Chen and Zhang \cite{2} is named Newton Cubic Spline (NCS) method to facilitate comparison with the proposed method.

Similar to Brent’s method, which is a multipoint method\(^3\), the algorithm proposed here reuses function values \( \tilde{\alpha}(s) \) and derivatives \( \tilde{\alpha}'(s) \) from previous iterations to effectively double the order of accuracy of the interpolated volume fraction. This is the basis of the proposed Consecutive Cubic Spline (CCS) interpolation. An additional stabilization for the ends of the interval \([s_{\text{min}}, s_{\text{max}}]\) is developed, that effectively handles diminishing derivatives at the ends of the search interval.

Chen and Zhang \cite{2} increase the computational efficiency of the Moment-of-Fluid Method \cite{27}, by expressing the centroid of the cap polygon \( A(s) \) as a function of spherical interface orientation angles \( \theta, \phi \). This is relevant for increasing the efficiency of interface positioning in the context of algorithms that improve the orientation of the VOF interface. The CCS algorithm proposed in this manuscript achieves significantly higher computational efficiency compared to NCS, especially for challenging values \( \alpha_c \approx 1, \alpha_c \approx 0 \), even without utilizing the information about the interface orientation. Of course, CCS can be coupled to MoF or any other algorithm that improves interface orientation in the same way as NCS. Since the convergence order of the CCS algorithm, demonstrated in the following section, is significantly higher than NCS without the use of interface orientation, adding this information would only increase the computational efficiency of the interface reconstruction algorithm. In future work, CCS will be coupled with the simplified Swartz reconstruction algorithm \cite{25}.

\(^3\)Multipoint root finding methods re-use function values and derivative from previous iterations to increase the convergence order.
2. Iterative positioning with Consecutive Cubic Spline interpolation

Chen and Zhang \[2\] use $V(s_n), V'(s_n) = A(s_n)$ in a Newton iteration

$$s_{n+1} = s_n - \frac{\tilde{\alpha}(s_n)}{\tilde{\alpha}'(s_n)},$$

(17)

known to be second-order convergent if $\tilde{\alpha}'(s_n) \neq 0$ within the search interval, which is not the case for interface positioning, where the function derivatives diminish on the boundaries of the search interval.

The proposed Consecutive Cubic Spline (CCS) algorithm interpolates the volume fraction $\tilde{\alpha}(s)$ with a Hermite polynomial between two consecutive iteration steps when they contain the root, by re-using volume fraction values and derivatives from previous iterations.

Using function value and function derivative from two subsequent iterations makes the CCS a two-point root finding algorithm \[1, \text{chap. 2}\]. If the interpolation interval does contain the root, the cubic interpolation exactly recovers the volume fraction $\tilde{\alpha}(s)$, and its root is therefore the root of $\tilde{\alpha}(s)$. The volume fraction $\tilde{\alpha}(s)$ is approximated with the polynomial of order $K < 4$, $P_k(s) = \sum_{k=0}^{K} a_k s^k \approx \tilde{\alpha}(s)$, using the conditions

$$P_k(s_a) = \tilde{\alpha}(s_a),$$
$$P_k'(s_a) = \tilde{\alpha}'(s_a),$$
$$P_k(s_b) = \tilde{\alpha}(s_b),$$
$$P_k'(s_b) = \tilde{\alpha}'(s_b),$$

(18)

that determine $k$, if $s^* \in [s_a, s_b] \subset (s_{\text{min}}, s_{\text{max}})$, where $s^*$ is the root of $\tilde{\alpha}(s)$ given by eq. (9).

To understand why $P_k(s)$ recovers the volume fraction exactly, consider the following. The volume $V_c$ is bounded by planar polygons or, if the boundary of $V_c$ consists of non-planar faces, by sets of triangles resulting from the triangulation of non-planar faces. In both cases, the cap area $A(x = x_{\text{min}} + sn_c)$ in fig. 4 is at most quadratic in $(x,y,z)$ in three dimensions. Because the parametrization $x = x_{\text{min}} + sn_c$ given by eq. (7) is linear, parametrized $A(s)$ is then also at most quadratic in $s$, in three dimensions. By eq. (15), $\tilde{\alpha}(s)$ is then at most cubic in three dimensions. We also know that $\tilde{\alpha}(s)$ is piecewise-polynomial between the points $x_p$ of the volume $V_p$: it is $C^1$ continuous at the boundaries of intervals bracketed by the points $x_p$ of $V_c$, because its derivative, the cap (intersection) area $A(s)$, is $C^0$ continuous in $s$, in cells that are used in unstructured meshes for the discretization of Partial Differential Equations. Therefore,

$$\tilde{\alpha}(s) \in \mathbb{P}_k, \quad k = 0, 1, 2, 3,$$

(19)
(a) CCS initial guess: interpolate $\tilde{\alpha}(s)$ over the initial search interval using boundary conditions given by eq. (18).

(b) CCS first iteration: a cubic spline interpolation using $[s_0, s_2, s_1]$ and $(\tilde{\alpha}(s_0), \tilde{\alpha}(s_1), (\tilde{\alpha}(s_2), \tilde{\alpha}'(s_2)))$.

(c) CCS final iterations: $s_4$ is given by the Newton’s method, because $s^* \notin [s_2, s_3]$. Since $s^* \in [s_4, s_3]$, the CCS polynomial is used over $[s_4, s_3]$ with $((\tilde{\alpha}(s_4), \tilde{\alpha}'(s_4), (\tilde{\alpha}(s_3), \tilde{\alpha}'(s_3)))$.

Figure 5: A schematic representation of the CCS method in the worst case scenario: when the root $s^*$ happens not to lie between two successive iterations. Because of the high accuracy of the spline $P_k(s)$, consecutive iterations are very close to each other and immediately ”land” within a bracketed interval. Within this interval $P_k(s)$ exactly recovers $\tilde{\alpha}(s)$, and thus the interface position.

inside the interval bracketed by points $x_p$ of the volume $V_c$, and it is $C^1$ continuous on the boundaries of bracketed intervals. The $\tilde{\alpha}(s)$, defined by eq. (9), is calculated geometrically using the truncated volume $V_c$. Similarly, $\tilde{\alpha}'(s)$ is geometrically calculated as $A(s)$, the cap area in fig. 4. Therefore, the conditions given by eq. (18) are satisfied exactly up to machine epsilon. Consequentially, $P_k(s)$ exactly interpolates $\tilde{\alpha}(s)$ within the bracketed interval and an exact root of $P_k(s)$ is equivalent to the root of $\tilde{\alpha}(s)$ within the bracketing interval. Note that the calculation of $\tilde{\alpha}'(s)$ comes at no additional computational cost, because $A(s)$ is a byproduct of the volume truncation used to compute $V(s)$ for $\tilde{\alpha}(s)$.

This very straightforward way of interpolating $\tilde{\alpha}(s)$ that exactly recovers $\tilde{\alpha}(s)$
within a bracketed interval is the main contribution of the CCS algorithm, compared to other methods that require complex geometrical parameterizations [3, 4, 5, 6] to achieve the same. Furthermore, directly compared to NCS [2], CSS requires overall significantly less iterations, especially at the ends of the search interval, where the volume fraction derivative diminishes.

The two-point root finding approach is novel compared to only using $\tilde{\alpha}'(s)$ in a Newton iteration by the NCS algorithm of Chen and Zhang [2]. The number of iterations of the CCS algorithm are not impacted by the way $\tilde{\alpha}(s), \tilde{\alpha}'(s)$ are computed. Different geometrical models and algorithms can be used to compute $\tilde{\alpha}(s), \tilde{\alpha}'(s)$. As long as $\tilde{\alpha}(s), \tilde{\alpha}'(s)$ are computed near machine-epsilon accuracy (e.g. using algorithms from [6], or from [11]), the number of iterations performed by CCS will not change. Because the quadratic nature of the intersection area results in eq. (19), extending the two-point CCS approach to include a polynomial interpolation with order $K > 3$, by storing data from more than two consecutive iterations brings no benefit. In fact, increasing the polynomial order beyond 3 destabilizes the root finding process because higher-order polynomials often have more than a single root $s^*$ in the search interval $(s_a, s_b)$.

The first part of the proposed CCS iterative positioning method is shown in algorithm 1 and the CCS interpolation part is shown in algorithm 2. The initial guess of the CCS method is calculated using the Hermite cubic interpolation over $[s_{\text{min}}, s_{\text{max}}]$, with boundary conditions given by eq. (18) at $s_{\text{min}}, s_{\text{max}}$, as shown schematically in fig. 5a. To use the CCS interpolation between consecutive iterations in next iterations, the interval $[s_{a}, s_{b}]$ is updated in algorithm 1 with data from the current iteration $s_n$ ($s_2$ in fig. 5a).

In the first iteration, there is only a single root candidate available, shown as $s_2$ schematically in fig. 5b. Hermite cubic spline is therefore interpolated in the next step over the initial interval $[s_{\text{min}}, s_{\text{max}}]$ ($[s_0, s_1]$ in fig. 5b), together with the first root candidate $s_n$ and its value and derivative pair ($\tilde{\alpha}(s_n), \tilde{\alpha}'(s_n)$).

For all further iterations ($n > 0$), as shown in algorithm 2, the CCS interpolation with $P_k(s)$ is performed between $[s_a, s_b]$ using values and derivatives from previous root candidates. If $[s_a, s_b]$ is a subset of a bracketing interval, and the root of $P_k(s)$ is inside $[s_a, s_b]$, the position of the interface (root of $\tilde{\alpha}(s)$) is exactly equal to the root of $P_k(s)$. For the results presented in this manuscript, Brent’s method [28] is used to find the root of the polynomial with machine epsilon tolerance. Finding the root of $P_k(s)$ using the Brent’s method costs only a small fraction of overall computational cost and the choice of the polynomial root-finding algorithm has no impact on the number of iterations of the CCS method as long as the polynomial root is calculated within a machine-epsilon tolerance.
An important detail regarding \( P_k(s) \) is the accurate calculation of \( k \) the polynomial coefficients for \( P_k(s) \). An error in the calculation of \( P_k(s) \) coefficients means \( P_k(s) \) cannot exactly interpolate \( \tilde{a}(s) \) in \([s_a, s_b] \), resulting, in turn, with an erroneous interface position, even if the root of \( P_k(s) \) is calculated exactly. Parameterizing \( s \) such that \( s \in [0, 1] \), enables a numerically stable way to exactly calculate the coefficients of \( P_k(s) \) as

\[
\begin{align*}
a_0 &= \tilde{a}(s_a) \\
a_1 &= -(\tilde{a}'(s_a)s_a - \tilde{a}'(s_b)s_b) \\
a_2 &= -(2\tilde{a}'(s_a)s_a + 2\tilde{a}'(s_b)s_b - \tilde{a}'(s_0)s_0 + \tilde{a}'(s_0)s_0 + 3\tilde{a}(s_a) - 3\tilde{a}(s_b)) \\
a_3 &= -(\tilde{a}'(s_a)s_a - \tilde{a}'(s_b)s_b + \tilde{a}'(s_a)s_a - \tilde{a}'(s_b)s_b - 2\tilde{a}(s_a) + 2\tilde{a}(s_b)).
\end{align*}
\]

An unnecessary intersection is removed by line 37 in algorithm \boxed{2} in the case when the...
Algorithm 2 Consecutive Cubic Spline (CCS) iterative positioning: Part II

27: \textbf{if} \( n == 0 \) \textbf{then} \quad \triangleright \text{A single root candidate is available in the first iteration.} \\
28: \quad \text{Interpolate} \ P_k(s) \ \text{over} \ [s_0, s_n, s_1]. \quad \triangleright \ [s_0, s_1] \ \text{is the initial search interval.} \\
29: \quad P_k(s_0) = \tilde{\alpha}(s_0), \ P_k(s_1) = \tilde{\alpha}(s_1), \ P_k(s_n) = \tilde{\alpha}(s_n), \ P'_k(s_n) = \tilde{\alpha}'(s_n). \\
30: \quad s_{n+1} = \text{root}(P_k(s)) \in [s_0, s_1] \\
31: \textbf{else} \\
32: \quad \text{Interpolate} \ P_k(s) \ \text{over} \ [s_a, s_b] \ \text{by Consecutive Cubic Spline.} \\
33: \quad P_k(s_a) = \tilde{\alpha}(s_a), \ P'_k(s_a) = \tilde{\alpha}'(s_a), \ P_k(s_b) = \tilde{\alpha}(s_b), \ P'_k(s_b) = \tilde{\alpha}'(s_b). \\
34: \quad s_{n-1} = s_n \quad \triangleright \text{Store the old position before update.} \\
35: \quad \textbf{if} \ P_k(s_a) \cdot P_k(s_b) < 0 \ \textbf{then} \\
36: \quad \quad s_{n+1} = \text{root}(P_k(s)) \in [s_a, s_b]. \quad \triangleright \text{Compute the CCS root.} \\
37: \quad \quad \textbf{if} \ |P_k(s_{n+1})| < \epsilon_p \ \textbf{then} \quad \triangleright \text{If the polynomial root value is below tolerance.} \\
38: \quad \quad \quad \textbf{return} \ s_{n+1}, p(s_{n+1}) \quad \triangleright \text{Terminate.} \\
39: \quad \textbf{end if} \\
40: \quad \textbf{if} \ n > 2 \ \text{and} \ |\tilde{\alpha}'(s_a)| < \epsilon \ \text{or} \ |\tilde{\alpha}'(s_b)| < \epsilon \ \textbf{then} \quad \triangleright \text{Non-zero derivative at} \ s_a \ \text{or} \ s_b. \\
41: \quad \quad s_{n+1} = 0.5(s_a + s_b) \quad \triangleright \text{Bisect.} \\
42: \quad \textbf{end if} \\
43: \quad \textbf{else} \\
44: \quad \quad s_{n+1} = s_n - \frac{\tilde{\alpha}(s_n)}{\tilde{\alpha}'(s_n)} \quad \triangleright \text{Newton iteration.} \\
45: \textbf{end if} \\
46: \textbf{end if} \\
47: \textbf{end for}

root value of \( P_k(s) \) falls below the positioning tolerance, because when \( s^* \in [s_a, s_b], \)
\( P_k(s) \) recovers \( \tilde{\alpha}(s) \) exactly.

In the case when \( s^* \notin [s_a, s_b], \) a Newton iteration is used, as shown schematically in fig. 5c. A separate special case of \( s^* \notin [s_a, s_b], \) happening over multiple iterations, is caused by the collinearity of the halfspace normal \( \mathbf{n}_c \) with the normal vector of a planar face of \( V_c. \) This leads to \( \tilde{\alpha}'(s_b) \neq 0 \) \( (\tilde{\alpha}'(s_a) \neq 0) \) where \( s_b = s_{max} \ (s_a = s_{min}), \)
which is contrary to the boundary conditions 18. This special case is handled in a very straightforward way by CCS, by lines 40 – 42 in algorithm 2. In the case where CCS needs more than 2 iterations and one of the boundary derivatives is zero, this is a possible special case with a falsely assumed zero derivative at one of the interval boundaries. In this case a simple bisection step is performed, because it will force a truncation and thus the calculation of the actual \( \tilde{\alpha}'(s_{n+1}), \) replacing the assumed zero derivative at \( s_{min} \) or \( s_{max} \) with the calculated derivative at \( s_{n+1}. \)

3. Results

The CCS positioning algorithm is tested in cells with different shapes, using challenging volume fraction values and interface orientations, to ensure robust con-
vergence on unstructured meshes. The cells used for testing are shown in fig. 6, namely: the tetrahedron (TET), the unit cube (CUBE), the dodecahedron (DOD), the endo-dodecahedron (ENDO) and the non-convex dodecahedron with non-planar faces (NPDO). The endo-dodecahedron represents a model of a non-convex polyhedron that has planar faces. The non-planar dodecahedron is ubiquitous in unstructured polyhedral meshes that are generated by agglomerating tetrahedral cells into polyhedral cells.

Point coordinates of dodecahedrons from figs. 6c to 6e are defined using

\[
\begin{align*}
(\pm 1, \pm 1, \pm 1) + \delta_{nonplanar}, \\
(0, \pm (1 + h), \pm (1 - h^2)) + \delta_{nonplanar}, \\
(\pm (1 + h), \pm (1 - h^2), 0) + \delta_{nonplanar}, \\
(\pm (1 - h^2), 0, \pm (1 + h)) + \delta_{nonplanar},
\end{align*}
\]

where \(\delta_{nonplanar} = (t, t, t)\) is the perturbation vector added to odd points and subtracted from even points in order to cause non-planarity of polyhedron faces. The dodecahedron in fig. 6c is computed with \(h = 0.5(-1 + \sqrt{5})\), \(\delta_{nonplanar} = 0\), the endo-dodecahedron is computed with \(h = -0.25\), \(\delta_{nonplanar} = 0\), and the non-planar dodecahedron is computed using \(h = 0.5(-1 + \sqrt{5})\), \(\delta_{nonplanar} = (0.2, 0.2, 0.2)\).

The CCS positioning is compared to NCS \[2\] using following test parameters

\[
\begin{align*}
\theta & \in \left(0, \frac{\pi}{N_\theta}, \frac{2\pi}{N_\theta}, \frac{3\pi}{N_\theta}, \ldots, \pi\right) \\
\phi & \in \left(0, \frac{\pi}{2N_\theta}, \frac{2\pi}{2N_\theta}, \frac{3\pi}{2N_\theta}, \ldots, 2\pi\right) \\
\alpha_c & \in \left(1e - 09, 1e - 08, 1e - 07, \ldots, 1e - 03, 1e - 03 + \frac{1 - 2e - 03}{N_\alpha}, 1e - 03 + 2\frac{1 - 2e - 03}{N_\alpha}, \ldots, 1 - 1 - e03, 1e - 04, 1e - 05, 1e - 06, 1e - 07, 1e - 08, 1e - 09\right)
\end{align*}
\]

where \((\phi, \theta)\) are the spherical angles that parametrize the interface normal as

\[
n_c = (\sin(\theta) \cos(\phi), \sin(\theta) \sin(\phi), \cos(\theta)).
\]

Values of \(\alpha_c\) in eq. (27) are based on the reconstruction tolerance \(\epsilon_R = 1e-9\), proposed in \[23\] and used by the author in \[24, 25\]. It is crucial to include values boundary volume fraction values \((\alpha_c \approx 0, \alpha_c \approx 1)\) in the test data, because positioning algorithms that rely on nonzero derivatives fail at the boundaries of the
Figure 6: Test polyhedrons.
\( \alpha_c \in [0, 1] \) interval. Boundary volume fractions are common in multiphase flow simulations, where the so-called *wisps* (dimensionally unsplit VOF) and *splitting-errors* (dimensionally split VOF) introduce small variations in \( \alpha_c \) in cells that are otherwise either full (\( \alpha_c = 1 \)) or empty (\( \alpha_c = 0 \)). Since the derivatives of \( \alpha_c(s) \) vanish for these volume fraction values (cf. fig. 2b), a positioning algorithm that does not take into account diminishing derivatives will experience either slower convergence, or in the worst case, divergence. The positioning condition is based on the positioning tolerance \( \epsilon_R < 1e-12 \), also used by López et al. [5]: this forces the CCS and NCS iterative algorithms to perform one more iteration, which further reduces \( \epsilon_P \) below machine epsilon for the vast majority of tests. The test results presented here are generated with \( N_\theta = 40 \) and \( N_\phi = 20 \), because there is no correlation observed between the interface orientation and the number of iterations or CPU time. Furthermore, as the tests results confirm, there is no need to use many \( \alpha_c \) values between \([1e-03, 1-1e-03]\), as the average iterations do not vary significantly within this interval. That is why \( N_\alpha = 50 \) was used for the tests presented here.

First let us consider the overall results of the CCS algorithm, compared with the NCS algorithm [2]. The increase in efficiency in terms of the average number of iterations is presented in fig. 7a and the decrease in the average CPU time is shown for different cell shapes in fig. 7b. An iteration is defined in this context as the calculation of the next interface position \( s^{n+1} \). Timing was performed on an architecture given by table 1, and the measurements were performed using the *chrono* C++ standard template library. The same tests were performed on a High-Performance Computing cluster without a significant difference in reported results. A Singularity image [29] that contains the source code and the computing environment is publicly available [30] and can be used to easily reproduce the results across different platforms. Results presented in this section using the computing architecture from table 1 are also publicly available, together with the source code and binary executables [31].

The results presented in fig. 7a show that CCS algorithm requires approximately \( 2.9-3.7 \) times less iterations than the NCS algorithm for different polyhedron shapes. In terms of the CPU time shown in fig. 7b the CCS algorithm is approximately \( 1.7-3.0 \) times faster than the NCS algorithm for different polyhedrons. The stabilized secant / bisection of Ahn and Shashkov [23] and Brent’s method were not used in this comparison, because Chen and Zhang [2] already show that those algorithms are significantly outperformed by the NCS algorithm.

The reduction of iterations achieved by the CCS algorithm is reflected in fig. 8 in the iteration distribution for each test polyhedron, when compared to the NCS.
algorithm. Only 2 iterations are required on average to position the interface within test polyhedrons. Results shown in figs. 7a, 7b and 8 confirm that the CCS algorithm significantly improves the computational efficiency of the interface positioning algorithm on polyhedrons, including star-shaped non-convex polyhedra with non-planar faces.

Chen and Zhang [2] use the NCS algorithm in combination with the Moment-of-Fluid (MOF) method and extend the interface positioning problem with the interface orientation angles $\theta, \phi$ that define the interface orientation vector in eq. (28). Using the implicit formulation for the interface plane $n_{c,x}x + n_{c,y}y + n_{c,z}z + d = 0$, the interface position $d$ is defined by Chen and Zhang [2] as a multivariate function $d := d(\alpha_c, \theta, \phi)$. Chen and Zhang [2] then extend the multivariate function $d$ into Taylor series which results in a prediction of the interface position in the new iteration based on $\partial_{\theta}d$ and $\partial_{\phi}d$. Consequently, when the information about the changing interface orientation is available from an algorithm that tries to improve the orientation (e.g. MOF), it is possible to find the position of the interface faster. However, it is crucial to note that the orientation information does not impact the partial derivative $\partial_{\alpha_c}d$ and the NCS algorithm is used in Chen and Zhang [2] independently of the interface orientation prediction. The CCS positioning algorithm improves significantly the estimation of $\partial_{\alpha_c}d$ by the CCS polynomial interpolation of $\alpha(s)$. Chen and Zhang [2, Figure 12] report an average of 4 iterations when the orientation information is given by $N_\theta = 100(N_\phi = 200)$, and approximately 3 iterations when the orientation information is given by $N_\theta = 1000(N_\phi = 2000)$ for a test case with $\alpha_c \in [1e-03, 1 - 1e - 03]$.

The CCS algorithm positions the interface with only 2 average iterations without relying on the interface orientation information (cf. fig. 8), for different cell shapes,
TET = tetrahedron, DOD = dodecahedron, ENDO = endo dodecahedron, NPDO = non-planar dodecahedron

(a) Comparison of average iterations for each cell shape.

(b) Comparison of average CPU time for each cell shape.

Figure 7: Speedup achieved by the CCS algorithm compared to NCS [2].

and with a more challenging set of $\alpha_c$ values given by eq. (27). This is confirmed by Chen and Zhang [2, Figure 13], with more challenging values $\alpha_c \in [1e^{-09}, 1e^{-09}]$, where the average iterations are reported as follows: NCS 13.53 average iterations,
predicted NCS \((N_\theta = 100)\) 6.57 average iterations, and predicted NCS \((N_\theta = 1000)\) 2.89 average iterations. Therefore, it takes the orientation prediction in [2] 1000 subdivisions of the interval \(\theta \in [0, \pi]\) to provide enough information to the Taylor series expansion in order to reach the average number of iterations given by the CCS that does not require this information. Because the MOF method requires significantly less than 1000 of iterations to improve the interface orientation, Chen and Zhang [2] report an overall increase of 60 – 66% in efficiency in terms of the average number of iterations. It is relevant to note that the implementation of NCS used here for comparison is numerically unstable: a test configuration for the non-planar dodechedron in fig. 8 causes the NCS not to converge even after 100 iterations.

The CIBRAVE method by López et al. [26, 5] uses Interpolation Bracketing based on linear interpolation of the bracketing volume, and an explicit (analytical) calculation of the interface position within the bracketed interval. López et al. [26, Fig. 14] report a \(\log_2(\log_2(I_p))\) complexity for the average number of volume truncation operations for the CIBRAVE method, where \(I_p\) is the number of vertices of the polyhedron. This results in 2.6 bracketing truncations on average for a dodecahedron. However, unlike the iterative algorithms, bracketing algorithms additionally require the evaluation of a geometrically parameterized explicit function to position the interface, and this comes at a cost (cf. [26, Fig 15]). López et al. [26] state that the cost of the coefficient calculation for explicit positioning function is ”around 1.7 times the CPU time needed to make a truncation operation” [26]. López et al. [26] use the test parameters from Diot and François [4], namely \(\alpha \in [1e^{-03}, 1 - 1e^{-03}]\), which is not as challenging as volume fraction values used here for the CCS method and by Chen and Zhang [2, Figure 13] for the NCS algorithm, so the results are difficult to compare directly. Even though López et al. [26] also use different cell shapes, the average number of iterations of CCS is 2, which is comparable to CIBRAVE, while being significantly easier to implement.

The distribution of the CPU time between sub-algorithms of the CCS method is shown in fig. 9. In fig. 9 the CPU time distribution is reported for every test polyhedron and a comparison between the NCS algorithm (blue color) and the CCS algorithm (orange color) is given. The CPU time is distributed between the numerical root finding for the Consecutive Cubic Spline (ROOT), the Consecutive Cubic Spline polynomial interpolation (POLY) and the geometrical volume truncation operation (GEOM). As expected, most of the computational cost is caused by the volume truncation operation, which is why the reduction in the number of iterations in fig. 7a correlates well with the speedup in terms of CPU time in fig. 7b. The GEOM CPU time would need to be reduced by approximately an order of magnitude for some polyhedrons in fig. 9 to reach the absolute level of ROOT and POLY times and
thus impact the relative increase in computational efficiency. For the tetrahedron, this is not relevant, as the average positioning CPU time is only 4.5 microseconds. López et al. [6] extend the volume truncation algorithm used by CIBRAVE [26, 5] to non-convex polyhedrons with the aim to maintain the same computational efficiency.
López et al. [6] compare the computational efficiency of the CIBRAVE method with and without their novel truncation using the Brent’s method as the basis for comparison. An order of magnitude faster positioning is achieved with CIBRAVE and the new truncation, and four times faster when with CIBRAVE and tetrahedral decomposition. The CCS algorithm would benefit in the absolute CPU time by introducing a faster truncation algorithm (e.g. the one proposed by López et al. [5, 6]), however this does not change its relative speedup, resulting from the reduction of the number of truncations. The truncation of a cell in an unstructured mesh is closely related to the data structure used to implement the unstructured mesh. To reduce the CPU time required for the interface positioning further, by introducing a new truncation
Figure 10: Average number of iterations depending on given volume fraction $\alpha_c$.

algorithm, existing mesh data structures (and respective cell models) should be used for the truncation operation. If a different cell data structure from the one in the unstructured mesh is used, the geometrical model of the cell must be translated into the model (data structure) that can be used by the positioning algorithm. This model translation incurs additional memory operations that are often computationally expensive and cache-inefficient, because they require access to unstructured data from the main memory. Additional calculations are also required, if an alternative connectivity of cell faces and edges is required by the positioning model, compared to what is available in the mesh. The CCS significantly increases the computational efficiency of interface positioning in a very straightforward way, irrespective of the available truncation algorithm / cell model, and can be readily used in an existing geometrical VOF code, without modification.

Finally, the reason behind the increased computational efficiency reported in figs. 7a, 7b and 8 is shown in fig. 10. The CCS algorithm outperforms the NCS algorithm [2] across the interval $\alpha_c \in [\epsilon_R, 1 - \epsilon_R]$. Additionally, at the boundaries of the interval $[\epsilon_R, 1 - \epsilon_R]$, the CCS algorithm achieves a significant reduction in
the average number of iterations even when $\epsilon_R = 10^{-9}$, which has not yet been reported for other contemporary positioning algorithms, to the best of the author’s knowledge.

4. Conclusions

A straightforward iterative algorithm is developed that significantly improves the computational efficiency of the VOF interface positioning problem on arbitrary unstructured meshes. The proposed Consecutive Cubic Spline (CCS) algorithm outperforms Brent’s method, the stabilized secant-bisection method of Ahn and Shashkov [23], and the Newton Cubic Spline method by Chen and Zhang [2]. The CCS algorithm is comparable with the CIBRAVE method of López et al. [26, 5, 6] in terms of the average number of volume truncations, with a challenging volume fractions testing sequence and without relying on a relatively complex geometric parameterization of the truncated volume. Its relative simplicity and the usage of geometrical data that are already available in the geometrical VOF method simplifies the adoption of the proposed CCS algorithm in existing numerical codes.

5. Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 265191195 – SFB 1194, sub-project Z-INF. The author is very grateful to Jan-Patrick Lehr M.Sc. for his advice on improving the performance measurements.

References

[1] M. Petkovic, B. Neta, L. Petkovic, J. Dzunic, Multipoint methods for solving nonlinear equations, Academic press, 2012.

[2] X. Chen, X. Zhang, A predicted-Newton’s method for solving the interface positioning equation in the MoF method on general polyhedrons, J. Comput. Phys. 384 (2019) 60–76. doi:10.1016/j.jcp.2018.12.038.

[3] S. Diot, M. M. François, E. D. Dendy, An interface reconstruction method based on analytical formulae for 2D planar and axisymmetric arbitrary convex cells, J. Comput. Phys. 275 (2014) 53–64. doi:10.1016/j.jcp.2014.06.060.

[4] S. Diot, M. M. François, An interface reconstruction method based on an analytical formula for 3D arbitrary convex cells, J. Comput. Phys. 305 (2016) 63–74. doi:10.1016/j.jcp.2015.10.011
[5] J. López, J. Hernández, P. Gómez, F. Faura, VOFTools - A software package of calculation tools for volume of fluid methods using general convex grids, Comput. Phys. Commun. 223 (2018) 45–54. doi:10.1016/j.cpc.2017.09.032.

[6] J. López, J. Hernández, P. Gómez, F. Faura, Non-convex analytical and geometrical tools for volume truncation, initialization and conservation enforcement in VOF methods, J. Comput. Phys. 392 (2019) 666–693. doi:10.1016/j.jcp.2019.04.055.

[7] R. B. DeBar, Fundamentals of the KRAKEN code, Tech. Rep. (1974) UCID–17366.

[8] D. L. Youngs, Time-dependent multi-material flow with large fluid distortion, Numerical Methods for Fluid Dynamics (1982). URL: https://ci.nii.ac.jp/naid/10029507464/en/.

[9] W. J. Rider, D. B. Kothe, Reconstructing Volume Tracking, J. Comput. Phys. 141 (1998) 112–152. doi:10.1006/jcph.1998.5906.

[10] J. R. Shewchuk, Lecture Notes on Geometric Robustness, 2013, pp. 1–95.

[11] H. T. Ahn, M. Shashkov, Multi-material interface reconstruction on generalized polyhedral meshes, J. Comput. Phys. 226 (2007) 2096–2132. doi:10.1016/j.jcp.2007.06.033.

[12] D. D. B. Kothe, W. W. Rider, S. S. Mosso, J. J. Brock, Volume tracking of interfaces having surface tension in two and three dimensions, Aiaa 96-0859 (1996) 25. doi:10.2514/6.1996-859.

[13] K. Shahbazi, M. Paraschivoiu, J. Mostaghimi, Second order accurate volume tracking based on remapping for triangular meshes, J. Comput. Phys. 188 (2003) 100–122. doi:10.1016/S0021-9991(03)00156-6.

[14] J. E. Pilliod, E. G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces, J. Comput. Phys. 199 (2004) 465–502. doi:10.1016/j.jcp.2003.12.023.

[15] 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 (2004) 718–742. doi:10.1016/j.jcp.2003.10.030.
[16] 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 (2006) 1011–1032. doi:10.1016/j.compfluid.2005.09.003.

[17] R. Scardovelli, S. Zaleski, Analytical Relations Connecting Linear Interfaces and Volume Fractions in Rectangular Grids, J. Comput. Phys. 164 (2000) 228–237. doi:10.1006/jcph.2000.6567.

[18] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, A geometrical area-preserving Volume-of-Fluid advection method, J. Comput. Phys. 192 (2003) 355–364. doi:10.1016/j.jcp.2003.07.003.

[19] X. Yang, A. J. James, Analytic relations for reconstructing piecewise linear interfaces in triangular and tetrahedral grids, J. Comput. Phys. 214 (2006) 41–54. doi:10.1016/j.jcp.2005.09.002.

[20] J. López, J. Hernández, Analytical and geometrical tools for 3D volume of fluid methods in general grids, J. Comput. Phys. 227 (2008) 5939–5948. doi:10.1016/j.jcp.2008.03.010.

[21] S. Ghali, Introduction to geometric computing, Springer Science & Business Media, 2008.

[22] P. Schneider, D. H. Eberly, Geometric tools for computer graphics, Elsevier, 2002.

[23] H. T. Ahn, M. Shashkov, Geometric algorithms for 3D interface reconstruction, in: Proc. 16th Int. Meshing Roundtable, IMR 2007, 2008, pp. 405–422. doi:10.1007/978-3-540-75103-8\_23.

[24] 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 preprint arXiv:1305.3417 (2013).

[25] T. Marić, H. Marschall, D. Bothe, An enhanced un-split face-vertex flux-based VoF method, J. Comput. Phys. 371 (2018) 967–993. doi:10.1016/j.jcp.2018.03.048.

[26] J. López, J. Hernández, P. Gómez, F. Faura, A new volume conservation enforcement method for PLIC reconstruction in general convex grids, J. Comput. Phys. 316 (2016) 338–359. URL: http://dx.doi.org/10.1016/j.jcp.2016.04.018. doi:10.1016/j.jcp.2016.04.018.
[27] V. Dyadechko, M. Shashkov, Moment-of-fluid interface reconstruction, Math. Model. Anal. 836 (2005) 1–41. URL: http://math.lanl.gov/~vdyadechko/doc/2005-mof.pdf doi:10.1016/j.cma.2009.08.009.

[28] R. P. Brent, An algorithm with guaranteed convergence for finding a zero of a function, 1971. doi:10.1093/comjnl/14.4.422.

[29] G. M. Kurtzer, V. Sochat, M. W. Bauer, Singularity: Scientific containers for mobility of compute, PloS one 12 (2017) e0177459.

[30] T. Maric, Iterative volume-of-fluid interface positioning in general polyhedrons with consecutive cubic spline interpolation: Singularity container, 2020. doi:10.25534/tudatalib-379.

[31] T. Maric, Iterative volume-of-fluid interface positioning in general polyhedrons with consecutive cubic spline interpolation: source code and results, 2020. doi:10.25534/tudatalib-378.