An Interface Tracking Method with Triangle Edge Cuts

Mengdi Wang^{a,*}, Matthew Cong^b, Bo Zhu^a

^aSchool of Interactive Computing, Georgia Institute of Technology, Atlanta, 30332, Georgia, USA ^bNVIDIA Corporation, 2788 San Tomas Expy, Santa Clara, 95051, California, USA

Abstract

This paper introduces a volume-conserving interface tracking algorithm on unstructured triangle meshes. We propose to discretize the interface via *triangle edge cuts* which represent the intersections between the interface and the triangle mesh edges using a compact 6 numbers per triangle. This enables an efficient implicit representation of the sub-triangle polygonal material regions without explicitly storing connectivity information. Moreover, we propose an efficient advection algorithm for this interface representation that is based on geometric queries and does not require an optimization process. This advection algorithm is extended via an area correction step that enforces volume-conservation of the materials. We demonstrate the efficacy of our method on a variety of advection problems on a triangle mesh and compare its performance to existing interface tracking methods including VOF and MOF.

Keywords: Interface Tracking, Interface Representation, Volume of Fluid, Volume-conserving Advection Algorithm

1. Introduction

The interface between immiscible fluids plays a fundamental role in understanding many real-world phenomena such as bubble dynamics, fluid fragmentation, and ocean surface swirls. *Interface tracking*, i.e., numerically modeling and evolving the interface, therefore has been a long-standing challenge in computational fluid dynamics.[1, 2, 3] The physical simulator requires precise information about the interface's location and various related physical quantities, such as total mass within a cell, and/or boundary conditions at the liquid-air interface. Essentially, interface

Preprint submitted to Journal of Computational Physics

^{*}Corresponding author.

¹E-mail addresses: mengdi.wang@gatech.edu (Mengdi Wang), matthew.d.cong@gmail.com (Matthew Cong), bo.zhu@gatech.edu (Bo Zhu).

tracking encompasses two primary components: A data structure for representing the interface and an advection algorithm for evolving the interface over time. In a typical simulation setup, the interface is discretized using primitives such as points, segments, and polygons and evolved intending to accurately preserve both the interface geometry and the total mass of materials.

Traditionally, interface tracking methods are classified as either Lagrangian or Eulerian. In a typical Lagrangian method, the interface geometry is represented by a low-dimensional Lagrangian data structure independent of the computational grid, e.g., marker points [4, 5], connected meshes and/or curves [6, 7, 8], and various other structures [9, 10]. While these methods can represent complex interfaces, accurately time-stepping the Lagrangian data structures often requires frequent remeshing to represent topological transitions and adequately sample large geometric distortions. These geometric operations may hinder the efficiency of implementation, especially on parallel systems in the presence of topology changes which often require (global) remeshing. On the other hand, Eulerian methods involve representing the interface on an Eulerian grid or mesh fixed in the space. For example, the level set (LS) method [11, 12, 13, 14] adopts an implicit function ϕ that defines the interface as the iso-surface $\phi(\mathbf{x}) = 0$. The level set method has been widely used in computational physics applications to track and solve dynamic interface problems and has proven its efficacy in handling complex topological and geometrical evolution without tackling local mesh repairments. However, it also suffers from volume loss, especially where/when thin features occur, due to the inherent limitations of representing these features on a grid with fixed resolution.

The volume of fluid (VOF) [15, 16, 17, 18] is an Eulerian method that has gained the most significant attention for its ability to conserve fluid volume. The volume of fluid $F_{i,j} \in [0, 1]$ is defined as the fraction of liquid inside a cell (i, j), and its value is tracked throughout the algorithm. The commonly used piece-wise linear interface calculation (PLIC-VOF) method [19, 20] assumes that the interface in a cell (i, j)is a line segment, which is reconstructed from $F_{i,j}$ using certain numerical schemes [21, 22]. In the traditional direction-splitting [23] advection method for PLIC-VOF, the advection problem of a cell (i, j) is decomposed into two simple one-dimensional advection problems along the x and y axes respectively, and their fluxes are combined to obtain the change of $F_{i,j}$. Figure 2(a) illustrates an advection step of the central cell along the x-axis in split PLIC-VOF advection.

The PLIC method often suffers from the artifacts of the interface reconstruction, especially for thin geometric features. Typically, the PLIC method reconstructs a thin fluid sheet as a group of broken droplets. López et al. have proposed a double-PLIC method [24] to tackle this problem by allowing up to two parallel interface segments to coexist in one cell and adding a marker point at the midpoint of each interface segment. These marker points are connected as segment meshes, whose topology will be used for spline interpolation. While double-PLIC is a hybrid Lagrangian-Eulerian approach and challenging for parallelization, it is important to recognize that representing the interface within a cell using only a single line segment is inherently inadequate for capturing thin interfaces. By placing two interface segments within a cell, double-PLIC effectively achieves sub-grid accuracy, improving the representation of these thin interfaces.

In recent years, many improvements to VOF method have emerged, aiming to enhance representation and advection accuracy [25, 26]. Notably, many of these methods address the advection problem by back-tracking a cell through the flow map to acquire its shape in the last frame. We formally describe this process using the concepts of *image* and *pre-image*. As illustrated in Figure 1, with superscript ndenoting the discretized time step $t = n\Delta t$, we define the *image* \vec{p}^n and *pre-image* \vec{p}^n of a point p^n as a mapping of point positions between different time steps [27],

$$\overrightarrow{p^{n}} = p^{n+1} = p^{n} + \int_{t^{n}}^{t^{n}+\Delta t} u(p(t), t) dt,$$

$$\overleftarrow{p^{n}} = p^{n-1} = p^{n} + \int_{t^{n}}^{t^{n}-\Delta t} u(p(t), t) dt.$$
(1)

The current state-of-the-art VOF methods adopt unsplit advection schemes [26],



Figure 1: The pre-image of a point p^n at the previous time step n-1, and its image at the subsequent time step n+1.

which significantly reduce the geometric errors introduced by operator splitting. This error arises because the split advection steps can distort the interface, leading to artifacts such as the staircase phenomenon [28] and errors from clamping the VOF values to [0, 1]. Denoting the rectangle shape defined by cell (i, j) as $C_{i,j}$, a typical unsplit advection scheme like EMFPA [29] and CCU [30] calculates the new value of $F_{i,j}$ by determining the fluid area inside its pre-image $C_{i,j}$ at the last time step, ensuring volume conservation. Other methods [31, 26] may use different types of control regions, but ultimately also involve intersecting their pre-images with the liquid region.

The MOF method [32, 33, 34, 35, 36] is also an improvement to the VOF method that operates explicitly on the pre-images of cells. In addition to the volume of fluid, also known as the zeroth moment of the fluid, the MOF method also tracks the first moment of fluid in a cell $C_{i,j}$, which is defined as $M_1 = \int_{C_{i,j}} \mathbf{x} d\mathbf{x}$, for higher accuracy of reconstruction. During the MOF advection, the moments of fluid are calculated from the intersection between the pre-image $\overline{C_{i,j}}$ and the liquid region at the last time step, which results in a set of polygons. The zeroth moment of fluid is given by summing up the areas of these polygons similar to unsplit VOF advection. However, the first moment is calculated by forward-tracking these polygons to get their images and then performing a weighted sum of the images' centroids [37]. This distinction arises because advection does not conserve the first moment. We illustrate the MOF advection process in Figure 2(b), where dotted lines draw the pre-image of the central cell, and its intersection with the liquid region is shown in red. The reconstruction step of the MOF method preserves the zeroth moment and tries to make the first moment of the reconstructed liquid polygon as close to the predicted value as possible, through a nonlinear optimization process [38].

Another interface tracking method, known as the **polygon area mapping** (PAM) method [27, 39, 40], goes a step further by not explicitly relying on VOF values. Instead, it adopts a combined Lagrangian-Eulerian perspective by storing the liquid region inside each cell as a set of polygons. The advection process of PAM begins with a similar intersection calculation of the cell pre-image and the liquid polygons and uses the images of the resulting polygons as an initial approximation of the liquid region inside the cell. Subsequently, PAM applies a correction step involving removing small polygons and vertices [27] to limit the number of polygons and vertices inside a cell. The advection step of PAM is illustrated in Figure 2(c). PAM is very similar to MOF, but it allows for a more complex interface in a cell.

Motivated by the previously mentioned methods, including double PLIC, unsplit VOF advection, MOF, and PAM, we aspire to formulate a sub-grid-accurate interface tracking algorithm by combining sub-grid interface and pre-image geometric calculation methods, which is particularly suited for handling thin fluid features. At the same time, we aim to design an algorithm that has lower complexity, a characteristic that is less emphasized in many existing methods. Although the PLIC interface can be reconstructed solely from VOF values, in practice, simulators often require an



Figure 2: Different advection methods of direction-splitting PLIC-VOF, MOF, PAM, and EBIT. (a) The cell is translated along the x-axis, and its intersection with the interface (red area) is calculated as the flux. (b) In MOF advection, the pre-image $\overleftarrow{C_{i,j}}$ of cell (i, j) is first calculated, and then the zeroth and first moments are calculated by intersecting it with the liquid region at the last time step. (c) PAM advection is similar to MOF, but it accommodates more interface segments inside a cell. (d) EBIT advection along the x-axis. Red and grey points represent new marker points located on grid lines.

explicit interface, especially when calculating geometric quantities like interface curvature in surface tension modeling [20, 41]. Therefore, PLIC-VOF requires 3 DoFs for each cell. In order to represent two parallel interfaces, double-PLIC takes 4 DoFs for two segments and 4 for two marker points, resulting in a total of 8 DoFs per cell.

MOF shows higher accuracy than the original version of split PLIC-VOF advection, at the cost of having 5 DoFs for each cell (2 additional DoFs for the first moment), while still representing only one interface segment inside a cell. Based on MOF, Shashkov and Kikinzon [35] propose MOF2, a second-order MOF that models sub-grid geometry by representing the liquid polygon as the union or intersection of two MOF-style liquid polygons. MOF2 tracks 3 second moments of fluid, increasing the DoFs per cell to 8. The complexity of the MOF method is also reflected in its interface reconstruction process which requires a costly nonlinear optimization. The complexity of moment tracking and interface reconstruction will further increase when extending these methods to 3D space. Similarly, although the PAM method can model complex sub-grid interfaces within a cell, its explicit definition of Lagrangian-style liquid polygons significantly increases the complexity of its data structure. For example, the maximum number of polygon vertices inside a cell can be up to 10 [27], making a total of 20 DoFs per cell.

Recently, an efficient interface tracking method, **edge-based interface tracking** (EBIT) [42, 3], has gained community attention. EBIT delineates the interface by a combination of vertex materials and a set of marker points along grid lines that indicate the intersection between the interface and the grid. In each cell, the

intersection points implicitly define the interface within it as a segment, therefore, the interface in the whole computational field can be represented as a segment mesh. EBIT adopts a direction-splitting advection scheme, where the interface is moved along x, y axes sequentially, and the new set of marker points is produced through calculating the intersection between the interface and the grid. Figure 2(d) illustrates the advection in the x direction. EBIT has only 2 DoFs for each cell, as each marker point is shared by two cells, highlighting its ability to represent the interface with a small amount of memory usage. However, EBIT struggles to conserve volume, and since it only allows one intersection on one edge, it often fails to represent features like thin fluid sheets. These shortcomings will be addressed in our proposed method.

The EBIT method inspired us to represent the interface elegantly using only a few degrees of freedom based on its intersections with the grid edges. As we will discuss in Section 8, we found that designing such an algorithm is more straightforward on a triangle mesh compared to a lattice grid.

Therefore, we propose an Eulerian interface-tracking algorithm that accurately tracks sub-triangle geometric features and ensures mass conservation. Our algorithm circumvents the issues related to maintaining connectivity information, requires less memory, and is easy to implement. In our proposed algorithm, the interface representation on an unstructured triangle mesh is divided into the interface representation inside each triangle element using *triangle edge cuts*, a novel data structure for representing two material regions inside the triangle. This data structure defines the intersections between the interface and the triangle edges. Assuming at most 2 intersections on each edge, we can store the interface using no more than 6 values per triangle, and the material regions are automatically reconstructed as polygons without ambiguity (see Section 2). The triangle edge cuts representation enables us to model sub-triangle interface geometries, including arbitrarily thin fluid features, with a low memory footprint. Moreover, we describe an efficient interface advection method for this representation in Section 3, which queries only segment intersections between the pre-images of triangle edges and the interface. To ensure higher accuracy and mass conservation, we propose an area correction method in Section 4 based on pre-image polygon intersections similar in spirit to MOF and PAM. In Section 5, we discuss the implementation details of our proposed algorithm. We then validate our algorithm using several static reconstruction and dynamic advection tests, comparing its accuracy and efficiency with state-of-the-art algorithms in Section 6. Finally, Section 7 discusses how to extend our proposed algorithm to 3D, while Section 8 provides a summary of this paper and outlines our future plans.

Variable	Definition
χ	Indicator function
Ω	Material region
\mathcal{P}	Polygon
$ \mathcal{T} $	Triangle
\mathcal{E}	Triangle edge cut
A	Liquid area
F	Relative area fraction
E_q	Absolute error
$\tilde{E_r}$	Relative error

2. Triangle Edge Cut Interface Representation

Table	1:	Terminology	y Table
-------	----	-------------	---------

In this paper, we focus on the interface tracking problem in a physical system consisting of two materials in \mathbb{R}^2 . Without loss of generality, we denote materials as 0 (air) and 1 (liquid) using the indicator function χ defined for any position p:

$$\chi(\boldsymbol{p}) = \begin{cases} 0, \text{ if } \boldsymbol{p} \text{ is air,} \\ 1, \text{ if } \boldsymbol{p} \text{ is liquid,} \end{cases}$$
(2)

We denote the air and liquid regions respectively as:

$$\Omega_0 = \{ \boldsymbol{p} : \chi(\boldsymbol{p}) = 0 \}, \Omega_1 = \{ \boldsymbol{p} : \chi(\boldsymbol{p}) = 1 \},$$
(3)

and the liquid-air interface is the boundary of the material regions, which is a codimension-1 geometric structure, *i.e.*, a curve in \mathbb{R}^2 , or a surface in \mathbb{R}^3 .

In our algorithm, we use the *triangle edge cut* to represent the material region as polygons inside a triangle. A triangle edge cut \mathcal{E} is defined as:

$$\mathcal{E} = (\mathcal{T}, c, R). \tag{4}$$

Here $\mathcal{T} = \Delta v_1 v_2 v_3$ is a triangle defined by its three vertices (v_1, v_2, v_3) . In the following sections of the paper, we will define its three edges as

$$e_{1} = \overline{\boldsymbol{v}_{1}\boldsymbol{v}_{2}},$$

$$e_{2} = \overline{\boldsymbol{v}_{2}\boldsymbol{v}_{3}},$$

$$e_{3} = \overline{\boldsymbol{v}_{3}\boldsymbol{v}_{1}}.$$
(5)

The boolean variable c indicates the material of v_1 :

$$c = \chi(\boldsymbol{v}_1). \tag{6}$$

R is a 3×2 matrix that satisfies the following conditions:

$$R_{i,j} \in [0,1],$$

 $R_{i,1} \le R_{i,2}.$
(7)

Each row R_i represents the vertices of material polygons on e_i as interpolation factors. Specifically, $R_{i,j}$ corresponds to a vertex defined as

$$\boldsymbol{r}_{i,j} = (1 - R_{i,j})\boldsymbol{v}_i + R_{i,j}\boldsymbol{v}_{i+1}.$$
(8)

We also call it a *cut* because $\mathbf{r}_{i,j}$ is essentially an intersection between the triangle edge and the interface. If $R_{i,j} \in \{0,1\}$, we refer to it as an *invalid cut* because, in that case, $\mathbf{r}_{i,j}$ coincides with a triangle vertex and does not serve as a vertex of a material polygon; thus, we simply ignore it. Otherwise, it's a *valid cut*. For convenience, if there is only one valid cut on e_i , we will assume it to be $R_{i,1}$. For example, $R_i = (0, 0.4)$ and $R_i = (0.4, 1)$ represent the same polygon vertices, and we will always use the second one. Note that by the definition of R, we restrict each edge to have at most 2 valid cuts.

Figure 3 depicts an example of a triangle edge cut with

$$c = 0, R = \begin{bmatrix} 0.3 & 1\\ 0.5 & 1\\ 0 & 1 \end{bmatrix}.$$
 (9)

Here, $R_{1,1}, R_{1,2} = (0.3, 1)$ indicates that there is one valid cut $\mathbf{r}_{1,1}$ on e_1 , represented as a red dot. This implies that $\chi(\mathbf{v}_2) = 1$, because there must be a liquid-air interface intersecting e_1 . Similarly, $\mathbf{r}_{2,1}$ is the only valid cut on e_2 , and we can deduce that $\chi(\mathbf{v}_3) = 0$. Therefore, we can reconstruct the liquid region as a triangle $\Delta \mathbf{r}_{1,1}\mathbf{v}_2\mathbf{r}_{2,1}$, as shown in the figure, where the small blue triangle denotes the liquid region, while the remaining white part represents air.

We can further perform similar interface reconstructions for all possible combinations of (c, R). Fortunately, we don't need to specify all the reconstructions explicitly, because it's easy to see that exchanging the roles of liquid and air, or cyclically permuting the vertices does not really change the results of interface reconstruction. Now consider the three vertices of the triangle: either all of them have the same material, or two vertices have the same material while the third one is different.



Figure 3: A simple example of a triangle edge cut. The small blue triangle denotes the liquid region.

Therefore, without loss of generality, we can assume that either all three vertices are air, *i.e.*, $c = \chi(\boldsymbol{v}_1) = \chi(\boldsymbol{v}_2) = \chi(\boldsymbol{v}_3) = 0$, or \boldsymbol{v}_1 is the only liquid vertex, *i.e.*, $c = \chi(\boldsymbol{v}_1) = 1$ and $\chi(\boldsymbol{v}_2) = \chi(\boldsymbol{v}_3) = 0$.

For the first case where c = 0, any two vertices on the same edge are of the same material, implied by the Jordan curve theorem [43], which states that the line segment connecting them must intersect the interface an even number of times. Under our restriction that each edge has at most two valid cuts, this number must be 0 or 2. Therefore, there can be 0, 1, 2, or 3 edges with 2 valid cuts, while the others have no valid cuts. Due to cyclic symmetry, we can again assume that if there is one edge with 2 valid cuts, it must be e_1 , and if there are two edges with 2 valid cuts, they must be e_1 and e_2 . In the second case where c = 1, there must be one valid cut on both e_1 and e_3 , since these edges connect two vertices of different materials. Edge e_2 can have either 0 or 2 valid cuts.

Figure 5 lists the four cases with c = 0 and the two cases with c = 1, which we refer to as the basic cases. The interface reconstruction of any possible triangle edge cut \mathcal{E}^2 can be derived from some \mathcal{E}^1 in one of the six basic cases, and an example of such derivation is shown in Figure 4. In the remainder of this paper, we will focus on the interface tracking algorithms on these basic cases. On an unstructured triangle mesh, the interface in the whole computational domain can be represented by defining the edge cut \mathcal{E}_i for each triangle \mathcal{T}_i . We denote the liquid polygon in \mathcal{E}_i as \mathcal{P} ; therefore, the liquid region in the whole computational domain consists of a set of polygons $\{\mathcal{P}_i\}$.



Figure 4: An example of deriving the interface reconstruction of a triangle edge cut \mathcal{E}^2 from a basic case \mathcal{E}^1 . First, we exchange the roles of the liquid and the air, changing the value of c from 0 to 1; accordingly, the air polygon is swapped with the liquid polygon. Second, we apply a cyclic permutation of vertex indices $1 \ 2 \ 3 \rightarrow 3 \ 1 \ 2$ to obtain \mathcal{E}^2 .



Figure 5: Six basic cases of triangle edge cuts. Black dots represent liquid, white dots represent air, and red dots represent valid cuts. Blue polygons represent the liquid regions inside triangles. In cases 1, 2, 3, 4, all vertices are air, while in cases 5, 6, there is one liquid vertex and two air vertices.

3. Interface Advection

Now consider the interface advection inside a triangle \mathcal{T} . We want to calculate $\mathcal{E}^{n+1} = (\mathcal{T}, c^{n+1}, R^{n+1})$ using the interface $\{\mathcal{E}^n_i\}$ defined at time step n. Note that the Eulerian triangle \mathcal{T} doesn't have a time step superscript. Using the interface reconstruction described in Section 2, we can explicitly represent the liquid region at time step n as a set of polygons $\{\mathcal{P}^n_i\}$. Algorithm 1 describes a simple advection algorithm that is carried out in three steps:

Step 1. Calculate pre-image \mathcal{T}' of the triangle by advecting all vertices v_i back by $-\Delta t$ using the 4th order Runge-Kutta method (RK4), and connecting these three advected vertices $\overleftarrow{v_1}, \overleftarrow{v_2}, \overleftarrow{v_3}$:

$$\mathcal{T}' = \Delta \overleftarrow{\boldsymbol{v}_1} \overleftarrow{\boldsymbol{v}_2} \overleftarrow{\boldsymbol{v}_3}. \tag{10}$$

Throughout the evolution of the interface, each particle retains its material, i.e., the Lagrangian transportation equation

$$\frac{\mathrm{D}\chi(\boldsymbol{p})}{\mathrm{D}t} = 0 \tag{11}$$

is satisfied. Therefore, we can directly calculate the material c^{n+1} from the interface at the *n*-th time step with

$$c^{n+1} = \chi^n \left(\overleftarrow{v_1}\right),\tag{12}$$

which is done by checking if $\overleftarrow{v_1}$ lies inside any polygon in $\{\mathcal{P}_i^n\}$ (including edges). If so, it will be classified as a liquid vertex; otherwise, it's air. Then we calculate the intersections of edges of \mathcal{T}' with the edges of $\{\mathcal{P}_i^n\}$:

$$\{\boldsymbol{r}_{i,j}^{\prime}\} = e_i^{\prime} \cap \{\mathcal{P}_i^n\}.$$
(13)

We will provide more implementation details about (12) and (13) in Section 5.

Step 2. Followed by the first step, we use the RK4 method again to calculate the images of the intersections $\{r'_{i,j}\}$ as the predicted edge cuts for \mathcal{E}^{n+1} , that

$$\{\boldsymbol{r}_{i,j}^{n+1}\} = \{\overrightarrow{\boldsymbol{r}_{i,j}'}\}.$$
(14)

However, due to the non-linear nature of RK4 time integration, $r_{i,j}^{n+1}$ may not lie on edge e_i , necessitating the next step.

Algorithm 1: InterfaceAdvection

Data: \mathcal{T} , $\{\mathcal{E}_i^n\}$ Result: $\mathcal{E}^{n+1} = (\mathcal{T}, c^{n+1}, R^{n+1})$ // Step 1 (pre-image query) 1 $\mathcal{T}' \leftarrow \Delta \overleftarrow{v_1} \overleftarrow{v_2} \overleftarrow{v_3}$; 2 $c^{n+1} \leftarrow \chi^n (\overleftarrow{v_1})$; 3 $\{r'_{i,j}\} = e'_i \cap \{\mathcal{P}_i^n\}$; // Step 2 (advection) 4 $\{r^{n+1}_{i,j}\} \leftarrow \{\overrightarrow{r'_{i,j}}\}$; // Step 3 (reconstruction) 5 $R^{n+1}_{i,j} \leftarrow (r^{n+1}_{i,j} - v_i) \cdot (v_{i+1} - v_i) / |v_{i+1} - v_i|^2$; 6 return $(\mathcal{T}, c^{n+1}, R^{n+1})$;

Step 3. In the last step, the predicted edge cuts $\{r_{i,j}^{n+1}\}$ are projected onto corresponding edges to calculate the final result R^{n+1} :

$$R_{i,j}^{n+1} = \frac{(\boldsymbol{r}_{i,j}^{n+1} - \boldsymbol{v}_i) \cdot (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)}{|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i|^2}.$$
(15)

Thus, we acquire the new triangle edge cut $\mathcal{E}^{n+1} = (\mathcal{T}, c^{n+1}, R^{n+1}).$

This interface advection algorithm only performs segment-segment intersections and cross products (to check the material of a point) in the pre-image; however, it exhibits appreciable errors and does not preserve the liquid area. In Section 4, we will discuss the area-correction step we used to address these issues. Furthermore, in Section 5, we will provide more details about the implementation of this method.

4. Area Correction

There are two major reasons for the area error. First, the interface reconstruction of case 2 in Figure 5 underestimates the liquid area. As shown in the left part of Figure 6, two cuts $\mathbf{r}_{1,1}, \mathbf{r}_{1,2}$ suggest the presence of a liquid region, depicted as the green dashed curve, however, it's reconstructed as pure air. Second, the triangle edge cut representation simplifies the interfaces inside \mathcal{T} to linear segments. As depicted in the right part of Figure 6, the actual interface is represented by the green dashed curve, but it is reconstructed as a straight segment, resulting in area errors.



Figure 6: Two main reasons for shape errors. Left: incorrect interface reconstruction in case 2. Right: errors from segment simplification of the interface inside \mathcal{T} . The green dashed curve represents the actual interface, and the blue region illustrates the reconstructed liquid region.

4.1. Additional Vertex for Case 2

To preserve the liquid area in case 2, we introduce a new liquid polygon vertex, denoted as v_t , inside \mathcal{T} . The liquid region is then reconstructed as a triangle $\Delta r_{1,1}r_{1,2}v_t$, as shown in Figure 7. Taking v_t into account, the liquid polygons in all

Case $\#$	c	t	Liquid Polygon \mathcal{P}
1	0	(0, 0, 0)	Ø
2	0	(2, 0, 0)	$(m{r}_{1,1},m{r}_{1,2},m{v}_t)$
3	0	(2, 2, 0)	$(m{r}_{1,1},m{r}_{1,2},m{r}_{2,1},m{r}_{2,2})$
4	0	(2, 2, 2)	$(m{r}_{1,1},m{r}_{1,2},m{r}_{2,1},m{r}_{2,2},m{r}_{3,1},m{r}_{3,2})$
5	1	(1, 0, 1)	$(m{v}_1,m{r}_{1,1},m{r}_{3,1})$
6	1	(1, 2, 1)	$(m{v}_1,m{r}_{1,1},m{r}_{2,1},m{r}_{2,2},m{r}_{3,1})$

Table 2: Corresponding liquid polygon of each basic case, including the additional vertex v_t in case 2.

basic cases are summarized in Table 2. Here, t is defined as a 3×1 vector indicating the number of valid cuts on each edge:

$$t_i = |(0,1) \cap \{R_{i,1}, R_{i,2}\}|.$$
(16)

When advecting the interface, we determine the position of v_t^{n+1} in the occurrence of case 2 using the method outlined in Algorithm 2. First, we calculate the



Figure 7: Case 2 after adding the additional vertex inside the triangle. The liquid area is now represented by the blue triangle $\Delta \mathbf{r}_{1,1}\mathbf{r}_{1,2}\mathbf{v}_t$.

intersection of \mathcal{T}' defined in (10) with the liquid polygons at time step n:

$$\{\Pi_k\} = \mathcal{T}' \cap \{\mathcal{P}_i^n\}.$$
(17)

Different from (13) that yields a set of points $\{\mathbf{r}'_{i,j}\}$, the polygon-polygon intersections here result in a set of polygons $\{\Pi_k\}$. Similar to MOF [32] and PAM [27], the image of a polygon Π_k is obtained by connecting its RK4-advected vertices forward in time Δt . The images of $\{\Pi_k\}$ are referred to as $\{\overrightarrow{\Pi_k}\}$, which serves as an approximation of the liquid region inside \mathcal{T} at time step n + 1. The centroid \mathbf{x}_c of this liquid region is then computed as

$$\boldsymbol{x}_{c} = \frac{\sum_{k} \int_{\overrightarrow{\Pi_{k}}} \boldsymbol{x} \mathrm{d} \boldsymbol{x}}{\sum_{k} |\overrightarrow{\Pi_{k}}|}.$$
(18)

Unlike the liquid area calculation (24) which will be discussed later and is performed in the pre-image, we advect the liquid polygons $\{\Pi_k\}$ forward in time in (18) only in the presence of case 2. The reason is the same as in the MOF method's first moment calculation, namely, the first moment conservation is not guaranteed.

We define the first tentative value v_t^* of the additional vertex v_t^{n+1} such that the reconstructed liquid polygon has a centroid x_c . This can be achieved by taking

$$\boldsymbol{v}_t^* = 3\boldsymbol{x}_c - \boldsymbol{r}_{1,1}^{n+1} - \boldsymbol{r}_{1,2}^{n+1}, \qquad (19)$$

where $\boldsymbol{r}_{1,1}^{n+1}, \boldsymbol{r}_{1,2}^{n+1}$ are given by (14). However, \boldsymbol{v}_t^* may fall outside \mathcal{T} , prohibiting us from taking it as the additional vertex. If that happens, we then try to find two lines

 L_1, L_2 in $\{\mathcal{P}_i^n\}$ that generate the edge cuts $r_{1,1}, r_{1,2}$. The image of their intersection is then calculated as a second tentative value v_t^{**} :

$$\boldsymbol{v}_t^{**} = \overrightarrow{L_1 \cap L_2}.\tag{20}$$

If the line intersection also fails, we will try to find a vertex of $\{\overrightarrow{\Pi_k}\}$ that lies inside \mathcal{T} and is farthest from e_1 . This vertex is considered as the third tentative value v_t^{***} :

$$\boldsymbol{v}_t^{***} = \operatorname*{argmax}_{\boldsymbol{p}} \mathrm{D}_{\perp}(\boldsymbol{p}, e_1), \boldsymbol{p} \in \mathcal{T} \wedge \boldsymbol{p} \text{ is a vertex of}\{\overline{\Pi_k}'\}.$$
(21)

If that fails again, meaning there are no vertices of $\{\overrightarrow{\Pi_k}\}$ falling in \mathcal{T} , we will consider it as a degenerate case and fall back to case 1.

4.2. Edge Cut Correction

In this section, we will discuss the quadratic edge cut correction step we used to preserve the material areas, which is applied at the end of the advection algorithm. Before that, we will first briefly discuss the material areas in a triangle edge cut. We can easily observe that the relative material areas, *i.e.*, the proportion of air and liquid in \mathcal{T} only depend on c and R. We denote the relative air area by F_0 , and the relative liquid area by F_1 . Thus

$$F_0 = F_0(c, R), F_1 = F_1(c, R).$$
(22)

In all basic cases, F_0 , F_1 are quadratic functions of elements in R, as listed in Table 3, where (u, v, w) satisfying

$$\begin{aligned} \boldsymbol{v}_t &= w \boldsymbol{v}_1 + u \boldsymbol{v}_2 + v \boldsymbol{v}_3, \\ u &+ v + w = 1, \end{aligned}$$
(23)

are the barycentric coordinates of the additional vertex \boldsymbol{v}_t in case 2.

Suppose that we have established a triangle edge cut

$$\mathcal{E}^{n+1} = (\mathcal{T}, c^{n+1}, R^{n+1})$$

at the end of Algorithm 1. Then, we want to further modify \mathcal{E}^{n+1} to preserve the liquid area:

$$A = \sum_{k} |\Pi_k|, \tag{24}$$

with Π_k defined in (17). Note that we calculate the liquid area in the pre-image similarly to MOF [32], for the same reason of mass conservation.

Algorithm 2: FindAdditionalVertex

```
Data: \mathcal{T}, \, r_{1,1}^{n+1}, \, r_{1,2}^{n+1}, \, \{\mathcal{P}_i^n\}
      Result: v_t^{n+1}
      // Centroid reconstruction
 1 {\Pi_k} \leftarrow \mathcal{T}' \cap \{\mathcal{P}_i^n\};
 2 \boldsymbol{x}_{c} \leftarrow \sum_{k} \int_{\overrightarrow{\Pi_{k}}} \boldsymbol{x} \mathrm{d}\boldsymbol{x} / \sum_{k} |\overrightarrow{\Pi_{k}}|;
 s v_t^* \leftarrow 3x_c - r_{1,1}^{n+1} - r_{1,2}^{n+1};
 4 if oldsymbol{v}_t^* \in \mathcal{T} then
 5 | return v_t^*;
 6 else
             // Line intersection
          L_1, L_2 \leftarrow \text{two lines in } \{\mathcal{P}_i^n\} \text{ corresponding to } r_{1,1}, r_{1,2};
 7
        v_t^{**} \leftarrow \overrightarrow{L_1 \cap L_2};
 8
           	ext{ if } v_t^{**} \in \mathcal{T} 	ext{ then }
 9
            return v_t^{**};
10
              else
11
                     \boldsymbol{v}_t^{***} \leftarrow \operatorname{argmax}_{\boldsymbol{p}} \mathcal{D}_{\perp}(\boldsymbol{p}, e_1), \text{ where } \boldsymbol{p} \in \mathcal{T} \text{ and } \boldsymbol{p} \text{ is a vertex of } \{\overrightarrow{\Pi_k}\};
12
                      \text{ if } \nexists \boldsymbol{v}_t^{***} \text{ then } \\
\mathbf{13}
                             // Algorithm fails, fall back to case 1
                             return \emptyset;
\mathbf{14}
                     else
15
                        return v_t^{***};
\mathbf{16}
```

Case #	F_0	F_1
1	1	0
2	$1 - F_1$	$v(R_{1,2}-R_{1,1})$
3	$1 - F_1$	$(1 - R_{1,1})R_{2,2} - (1 - $
		$R_{1,2}R_{2,1}$
4	$R_{1,1}(1 - R_{3,2}) + (1 - $	$1 - F_0$
	$R_{1,2})R_{2,1} + (1 - R_{2,2})R_{3,1}$	
5	$1 - F_1$	$R_{1,1}(1-R_{3,1})$
6	$(1 - R_{1,1})R_{2,1} + (1 - $	$1 - F_0$
	$R_{2,2})R_{3,1}$	

Table 3: Relative material areas of different cases

Our edge cut correction only involves modifying the elements in \mathbb{R}^{n+1} . We will not change c^{n+1} or the number of valid cuts along each edge. In other words, we want to find a 3×2 matrix \mathbb{R}' satisfying:

$$t(R') = t(R^{n+1}),$$

$$F_1(c^{n+1}, R') = \frac{A}{|\mathcal{T}|},$$
(25)

where t is defined in (16).

The way we decide R' is modeled by a parameter $\tau \in [0, 1)$ such that

$$R' = (1 - \tau)R^{n+1} + \tau R^*.$$
(26)

Clearly, $\tau = 0$ corresponds to the unchanged situation $R' = R^{n+1}$, that occurs only when

$$F_1(c^{n+1}, R^{n+1}) = \frac{A}{|\mathcal{T}|}$$
(27)

is already satisfied. R^* in (26) is the limit of R' when $\tau \to 1$, a degenerate case that we shall never actually reach. For example, in Figure 8 we have

$$c = 0, R = \begin{bmatrix} 0.7 & 1\\ 0 & 1\\ 0.4 & 1 \end{bmatrix},$$
(28)

and we want to make the liquid area smaller by moving $r_{1,1}$ to $r'_{1,1}$, and $r_{3,1}$ to $r'_{3,1}$, forming the new interface indicated by the green dashed line. In this case, we shall

have

$$R^* = \begin{bmatrix} 0 & 1\\ 0 & 1\\ 1 & 1 \end{bmatrix}.$$
 (29)

Intuitively, we're trying to move $r_{1,1}, r_{3,1}$ toward v_1 , shrinking the liquid region.



Figure 8: An example of edge cut correction. The initial liquid region (indicated by blue) is too large, and we want to make it smaller by moving the interface to a new position (green dashed line).

However, if they do reach v_1 at $\tau = 1$, the interface will degenerate to pure air. For all general cases, Table 3 and (26) imply that $F_1(c^{n+1}, R')$ is a quadratic function of τ . The geometric meaning of τ indicates that $F_1(\tau)$ is monotonic on [0, 1), therefore we either have zero or one solution $\tau_0 \in [0, 1)$ satisfying (25). If the solution τ_0 is present, we will update R^{n+1} accordingly with (26). Otherwise, we consider the area correction step to be failed and leave R^{n+1} untouched.

Next, we will discuss how to find R^* . For an edge e_i , if there is only one valid cut $\mathbf{r}_{i,1}$ on it, then we will want to move it either toward \mathbf{v}_i or \mathbf{r}_{i+1} , thus $R^*_{i,1} \in \{0, 1\}$. If there are two valid cuts $\mathbf{r}_{i,1}, \mathbf{r}_{i,2}$ and we want to separate them further apart, we have

$$R_{i,1}^* = 0,$$

$$R_{i,2}^* = 1.$$
(30)

This means that we are moving them toward the two ends of the edge. Otherwise, if we want to move them closer, we will set

$$R_{i,1}^* = R_{i,2}^* = s_i, (31)$$

with

$$s_i = \frac{R_{1,1}}{R_{1,1} + 1 - R_{1,2}}.$$
(32)

Equation (32) ensures that we always have

$$\frac{R_{1,1}^{n+1}}{1 - R_{1,2}^{n+1}} = \frac{R_{1,1}'}{1 - R_{1,2}'},\tag{33}$$

for any value of τ , keeping the ratio of the lengths of the two outer segments constant.

Special care must be taken for case 2 with the additional vertex v_t . If we want to expand the liquid region, we will move v_t toward v_3 in the same manner parameterized by τ as in (26). Conversely, if we want to shrink the liquid region, we will calculate a target point

$$\boldsymbol{s}_t = \frac{w}{u+w} \boldsymbol{v}_1 + \frac{u}{u+w} \boldsymbol{v}_2, \tag{34}$$

and move \boldsymbol{v}_t toward it. The barycentric coordinates of \boldsymbol{s}_t are $\left(\frac{u}{u+w}, 0, \frac{w}{u+w}\right)$, which means that we will keep the ratio u/w of the barycentric coordinates constant during the movement of \boldsymbol{v}_t .

Table 4 shows the area correction formulas. In case 2, the target points of v_t are indicated by v_t^* . The illustrations of the quadratic area correction for expanding and shrinking the liquid region are presented in Figure 9 and Figure 10. The quadratic

	Case	$ R_{1,1}^* $	$ R_{1,2}^* $	$ R_{2,1}^* $	$ R_{2,2}^* $	$ R_{3,1}^* $	$R_{3,2}^{*}$	$oldsymbol{v}_t^*$
	2	0	1					$oldsymbol{v}_3$
Freed	3	0	1	0	1			—
Expand	4	0	1	0	1	0	1	—
	5	1				0		
	6	1		0	1	0		
	2	s_1	s_1					$oldsymbol{s}_t$
Shrink	3	s_1	s_1	s_2	s_2			—
SIITIIK	4	s_1	s_1	s_2	s_2	s_3	s_3	
	5	0				1		
	6	0		s_2	s_2	1		

Table 4: Movements of edge cuts

edge cut correction is summarized in Algorithm 3. For cases 2, 3, 5, 6, we can exactly recover any relative liquid area in (0, 1), because

$$\lim_{t \to 0} F_1 = 0 (\text{pure air}),$$

$$\lim_{t \to 1} F_1 = 1 (\text{pure liquid}),$$

(35)



Figure 9: Area corrections when trying to expand the liquid region. Brown lines indicate the trajectories of cut points with $\tau \in [0, 1)$.



Figure 10: Area corrections when trying to shrink the liquid region. Brown lines indicate the trajectories of cut points with $\tau \in [0, 1)$. Green dots correspond to s_i defined in (32), or s_t defined in (34).

Algorithm 3: EdgeCutCorrection

 $\begin{array}{l} // \ F_1^* \text{ is the target value of liquid proportion} \\ \textbf{Data: } c^{n+1}, \ R^{n+1}, \ F_1^* \\ \textbf{Result: } R' \\ \textbf{1} \ R^* \leftarrow \text{Table 4;} \\ \textbf{2} \ f(\tau) \leftarrow F_1 \left(c^{n+1}, (1-\tau) R^{n+1} + \tau R^* \right); \\ // \ \text{Solve for the quadratic equation} \\ \textbf{3} \ \tau_0 \leftarrow f(\tau_0) = F_1^*; \\ \textbf{4} \ \textbf{return} \ (1-\tau_0) R^{n+1} + \tau_0 R^*; \end{array}$

and the monotonicity of $F_1(\tau)$ guarantees that there must be one solution. However, for case 4 we have

$$\lim_{t \to 0} F_1 = 0 (\text{pure air}),$$

$$\lim_{t \to 1} F_1 < 1,$$
(36)

because even in the limit case $\tau = 1$, there will still be a liquid triangle $s_1 s_2 s_3$ inside the triangle. This is the only case where our algorithm fails to preserve the liquid area. Fortunately, case 4 is very uncommon because it means there are 2 cuts on all three edges of \mathcal{T}^{n+1} , indicating thin intricate geometries of the interface that rarely occur.

By adding the additional vertex in case 2 and the incorporating the quadratic edge cut correction step into Algorithm 1, we have completed our interface advection algorithm with area correction in Algorithm 4. Compared to Algorithm 1, we have introduced a fix for case 2 (lines 6 - 7) and included an extra edge cut correction step (lines 8 - 11).

5. Implementation

In this section, we talk about the program implementation of our interface tracking algorithm. For each triangle \mathcal{T} and its triangle edge cut $\mathcal{E} = (\mathcal{T}, c, R)$, we store the R matrix with six floating-point variables. Additionally, c can be put into the sign bit of $R_{1,1}$, which avoids any extra memory usage. For case 2 in Figure 7, noting that we have all $R_{i,j} \in [0, 1]$, we can store the barycentric coordinates (u, v) defined in (23) for the extra vertex by setting $R_{2,1} = u + 2$, $R_{2,2} = v + 2$ to distinguish them from valid cuts. Table 5 compares the memory costs of different interface-tracking methods, including our proposed approach. With the triangle edge cut representation, our method enables sub-triangle modeling with a low memory cost of 6 DoFs for each element.

Algorithm 4: InterfaceAdvectionWithAreaCorrection

Data:
$$\mathcal{T}$$
, $\{\mathcal{E}_i^n\}$
Result: $\mathcal{E}^{n+1} = (\mathcal{T}, c^{n+1}, R^{n+1})$
// Step 1 (pre-image query)
1 $\mathcal{T}' \leftarrow \Delta \overline{v_1} \overline{v_2} \overline{v_3};$
2 $c^{n+1} \leftarrow \chi^n (\overline{v_1});$
3 $\{r'_{i,j}\} \leftarrow e'_i \cap \{\mathcal{P}_i^n\};$
// Step 2 (advection)
4 $\{r_{i,j}^{n+1}\} \leftarrow \{\overrightarrow{r'_{i,j}}\};$
// Step 3 (reconstruction)
5 $R_{i,j}^{n+1} \leftarrow (r_{i,j}^{n+1} - v_i) \cdot (v_{i+1} - v_i) / |v_{i+1} - v_i|^2;$
6 if $(\mathcal{T}, c^{n+1}, R^{n+1})$ falls into case 2 then
// Algorithm 2
7 $\lfloor v_t^{n+1} \leftarrow \text{FindAdditionalVertex}(\mathcal{T}, r_{1,1}^{n+1}, r_{1,2}^{n+1}, \{\mathcal{P}_i^n\});$
// Step 4 (edge cut correction)
8 $\{\Pi_k\} \leftarrow \mathcal{T}' \cap \{\mathcal{P}_i^n\};$
9 $F_1^* \leftarrow \sum_k |\Pi_k| / |\mathcal{T}|;$
10 if $F_1(c^{n+1}, R^{n+1}) \neq F_1^*$ then
 $\lfloor // \text{Algorithm 3}$
11 $\lfloor R^{n+1} \leftarrow \text{EdgeCutCorrection}(c^{n+1}, R^{n+1}, F_1^*);$
12 return $(\mathcal{T}, c^{n+1}, R^{n+1});$

Algorithm	Element DoFs	Sub-grid
Level Set	1	No
PLIC-VOF	3	No
MOF	5	No
Double-PLIC	8	Yes
MOF2	8	Yes
PAM	20	Yes
Proposed	6	Yes

Table 5: Comparison of memory costs of different interface tracking algorithms.



Figure 11: Difference between interior segments and boundary segments of the liquid polygon. Yellow lines $\overline{r_{1,2}r_{2,1}}$ and $\overline{r_{2,2}r_{1,1}}$ are interior segments. Green lines $\overline{r_{1,1}r_{1,2}}$ and $\overline{r_{2,1}r_{2,2}}$ are boundary segments.

For clarity, before introducing the implementation of the interface advection algorithm, we first distinguish between two types of liquid polygon edges shown in Figure 11: *interior segments* and *boundary segments*. An interior segment, as the name suggests, is fully inside a triangle, while a boundary segment lies along the boundary of a triangle. In Figure 11, interior segments are drawn with yellow lines, and boundary segments with green lines.

The most important steps in the interface advection Algorithm 4 are the first two: calculating the materials of vertex images (12) and finding the intersections between edges and the interface (13). To determine the material $\chi^n(\overleftarrow{v})$ of a vertex image \overleftarrow{v} , we first locate its belonging triangle \mathcal{T} of \overleftarrow{v} in the mesh, and check if \overleftarrow{v} lies inside any liquid polygon defined by its corresponding edge cut $\mathcal{E}^n = (\mathcal{T}, c^n, R^n)$. In practice, we don't need to perform the inclusion test for all the liquid polygons. Instead, for each interior segment, it is sufficient to check whether \overleftarrow{v} is on its left or right side. By using this method, we can reduce the computational effort. For example, in Figure 12, we only need to check if \overleftarrow{v} is on the right side of $\overrightarrow{r_{1,2}r_{2,1}}$ or $\overrightarrow{r_{2,2}r_{1,1}}$. If so, we conclude that \overleftarrow{v} is air, like the case of $\overleftarrow{v_i}$. Otherwise, \overleftarrow{v} is in the shadowed area, indicating it's liquid like $\overleftarrow{v_j}$. To avoid degeneracies, if \overleftarrow{v} lies exactly on an interior segment, we also consider it to be liquid.

There are two types of edge-interface intersections (13): intersections with interior segments, and intersections with boundary segments. For an edge e'_i in (13), we simply iterate over all interior segments in its intersecting triangles in the mesh to perform segment-segment intersections. However, extra attention must be paid to the boundary segments to avoid generating duplicate edge cuts, as an edge is shared by two triangles in the mesh. First, we calculate the intersections between edge e'_i



Figure 12: Determining the materials of $\overleftarrow{v_i}$ and $\overleftarrow{v_j}$ by checking if they're on the right side of interior segment $\overline{r_{1,2}r_{2,1}}$ or $\overline{r_{2,2}r_{1,1}}$.

and the edges of its intersecting triangles. If an intersection p is found, we will find its two neighboring triangle edge cuts \mathcal{E}^1 , \mathcal{E}^2 , and calculate the materials using the method described in Figure 12. If the results in \mathcal{E}^1 , \mathcal{E}^2 differ, p is considered as an edge cut, like point a in Figure 13. However, if the results are the same, p shall be inside of a larger liquid or air polygon, and thus not considered as an edge cut, like point b.



Figure 13: Calculating possible edge cuts between an edge e'_i and boundary segments. Point a is considered an edge cut since it has different materials in two adjacent triangles. However, b is not considered an edge cut because it lies inside a larger liquid polygon.

In our implementation, we also take efforts to avoid degenerate cases. For example, if an edge of \mathcal{T}' in (13) coincides with an edge in the liquid polygons $\{\mathcal{P}_i^n\}$, the intersection will be a segment instead of a single cut point. Or, if an edge of \mathcal{T}' passes through a vertex of a liquid polygon \mathcal{P}_i^n , we may find two duplicate cuts on it, however, none of them should be taken into account because, in this situation, the edge does not actually cut through the liquid region. We use two efforts to deal

with degeneracy. First, before the backward RK4 to calculate \mathcal{T}' , we add a random perturbation in the magnitude of $10^{-6}\Delta x$ to each vertex \boldsymbol{v}_i . This random perturbation largely eliminated the edge coincidence cases, especially when the velocity field is **0**. Then, we also use the materials of the vertices as an additional safeguard for degenerate cases when performing edge-interface intersections (13). We first query the materials

$$c_{1} = \chi^{n} \left(\overleftarrow{v_{i}} \right),$$

$$c_{2} = \chi^{n} \left(\overleftarrow{v_{i+1}} \right).$$
(37)

According to the Jordan curve theorem, if $c_1 = c_2$, there are either 0 or 2 cuts on e'_i , and if $c_1 \neq c_2$, there is only one cut. In the case of $c_1 = c_2$, if only one cut is found, we discard it. If we find more than two cuts, we keep only the first one and the last one, sorted by their distances from $\overleftarrow{v_i}$. Similarly, in the case of $c_1 \neq c_2$, we will only keep the first cut.

6. Experiments

In this section, we evaluate our interface advection algorithm with area correction on a static reconstruction test and four dynamic advection tests.

6.1. Static Reconstruction

We begin the numerical experiments with the convergence analysis of static reconstruction tests with three shapes: a circle, a snake shape, and a heart shape. In our numerical tests, we first divide the computational domain into a lattice grid and then divide each grid cell into a lower-right triangle and an upper-left triangle. The triangle edge cut is defined in each triangle, representing the interface.

In the circle test, we place a liquid circle with a radius of r = 0.15 at the center of the computational domain $[0, 1]^2$. The snake shape test is introduced by [35]. It also takes a $[0, 1]^2$ computational domain. In this test, the liquid region is the area between

$$y = 0.5 + 0.3\sin(2\pi x),\tag{38}$$

and

$$y = 0.5 + 0.3\sin(2\pi6). \tag{39}$$

The heart shape, also adopted by [35], is defined by a curve

$$x = \frac{16\sin^3(\theta)}{40} + 0.52,$$

$$y = \frac{13\cos(\theta) - 5\cos(2\theta) - 2\cos(3\theta) - \cos(4\theta)}{40} + 0.55,$$
(40)



Figure 14: Static reconstruction results for circle, snake, and heart shapes using an 8×8 lattice mesh (top row) and an unstructured triangle mesh (bottom row) at refinement level 0. Our interface representation method can effectively recover the shape even at such low resolutions.

for $\theta \in [-\pi, \pi]$.

The reconstruction results of three shapes at 8×8 resolution are shown in the top row of Figure 14. To quantify the accuracy of our method, we calculate the absolute shape error E_q using the expression [24]

$$E_g = \sum_{i,j} |A_{i,j} - \tilde{A}_{i,j}|, \qquad (41)$$

where $\tilde{A}_{i,j} = A(\mathcal{E}_1) + A(\mathcal{E}_2)$ represents the sum of liquid areas in two triangles of cell (i, j), and $A_{i,j}$ is the liquid area in cell (i, j), calculated using a high-resolution representation of the liquid shape consisting of 1000 vertices.

To assess the convergence rate across different resolutions, we employ the following formula [27]:

$$\mathcal{O}_n = \log_2\left(\frac{E_g(n/2)}{E_g(n)}\right). \tag{42}$$



Figure 15: Absolute shape errors for static reconstruction tests of three different shapes. Our proposed method demonstrates the expected second-order convergence.

We tested these three shapes at $N \times N$ resolutions for $N = \{8, 16, 32, 64, 128, 256\}$. Equation (42) implies that the convergence order corresponds to the slope in the $\log_2(N) - \log_2(E_g)$ diagram, which is shown in Figure 15. We use least squares fitting to estimate the slope, and the results indicate that our proposed algorithm achieves second-order accuracy, which is expected because we use straight lines to represent the interface. Higher convergence order requires using higher-order discretizations of the interface, such as curves.

We also evaluated our algorithm's static reconstruction results on unstructured meshes, which were generated by the Triangle mesh generator [44, 45]. For a refinement level $l = \{0, 1, 2, 3\}$, we restrict the maximum triangle area in the mesh to be $0.01/(2^l)^2$, and thus, the convergence order can be calculated as

$$\mathcal{O}_n = \log_2\left(\frac{E_g(l-1)}{E_g(l)}\right). \tag{43}$$

The numbers of triangles inside the $[0, 1]^2$ domain are $\{150, 620, 2472, 9899\}$ respectively for four refinement levels. The reconstruction results for l = 0 are shown in the bottom row of Figure 14. Based on the surface reconstruction approach proposed in this paper, we can use a relatively simple method to compute curvature: For each

interior segment \overline{pq} , we find the nearest interior segment within the two adjacent triangles to \overline{pq} . Then, using all the vertices of these segments, including p and q, we fit a parabola $y = ax^2 + bx + c$ and compute the curvature at the midpoint $(x_p + x_q)/2$ with the formula

$$\kappa = \frac{2a}{(1 + (2ax + b)^2)^{3/2}}$$

If the range of y coordinates for given points is larger than the range of x coordinates, we fit a parabola $x = ay^2 + by + c$ instead and compute κ similarly. We measure the L^{∞} error for the curvature computation as

$$E_{\kappa} = \max_{e_i} \frac{|\tilde{\kappa_i} - \kappa_i|}{|\kappa_i|}$$

for all interior segments e_i , where $\tilde{\kappa}_i$ is the computed curvature for e_i , and κ_i is the ground truth. Table 6 summarizes the shape error E_g and the curvature error E_{κ} , along with their convergence orders for the same three shapes on unstructured meshes. The shape error E_g shows a similar second-order convergence as lattice grids. The simple curvature scheme achieves first-order convergence, and its performance decreases at higher resolutions, because the points used for parabolic fitting are close to each other, leading to reduced accuracy.

Higher-order curvature computation schemes are beyond the scope of this paper. However, we adopted an interface discretization scheme similar to PLIC-VOF, and in our method, each edge cut corresponds to an edge in the triangle mesh. This correspondence is not altered during area correction, allowing us to establish a local segment mesh for each interface segment. As a result, the curvature computation schemes based on PLIC-VOF and the segment mesh are compatible with our algorithm. Readers may refer to [25, 46] for more details.

6.2. Rider-Kothe Reversed Single Vortex

We evaluate the efficacy of our method in handling shearing flows through the Rider-Kothe reversed single vortex test [18]. In this test, the velocity field is derived from the stream function

$$\psi(x,y) = \frac{1}{\pi} \sin^2(\pi x) \sin^2(\pi y) \cos\left(\frac{\pi t}{T}\right),\tag{44}$$

where T = 8. The corresponding velocity components are computed as

$$u = -2\cos(\pi t/T)\cos(\pi y)\sin^{2}(\pi x)\sin(\pi y), v = 2\cos(\pi t/T)\cos(\pi x)\sin(\pi x)\sin^{2}(\pi y).$$
(45)

Shape		Refinement $level(l)$					
		0	1	2	3		
	E_g	3.34×10^{-3}	9.30×10^{-4}	2.09×10^{-4}	4.56×10^{-5}		
airela	O_n		1.84	2.15	2.20		
circie	E_{κ}	6.43×10^{-1}	3.25×10^{-1}	1.25×10^{-1}	8.62×10^{-2}		
	O_{κ}		0.99	1.38	0.53		
anaka	E_g	4.55×10^{-3}	1.20×10^{-3}	3.14×10^{-4}	8.01×10^{-5}		
Shake	O_n		1.92	1.93	1.97		
hoart	E_g	7.30×10^{-3}	1.22×10^{-3}	2.81×10^{-4}	8.70×10^{-5}		
neart	O_n		2.58	2.12	1.69		

Table 6: Static reconstruction results and convergence orders for three shapes on unstructured triangle meshes. O_{κ} is the convergence order of E_{κ} .

When subjected to this velocity field, a liquid circle initially centered at (0.5, 0.75) with a radius of 0.15 undergoes significant stretching around the center, reaching its maximum extent at T/2, and subsequently reverses its trajectory back to its initial position. The absolute shape error is then measured using (41), with the initial shape serving as the ground truth. The advection results are summarized in Table 7. Remarkably, even at a relatively low resolution of 64×64 , our method exhibits strong performance, effectively restoring the initial state. As the resolution increases, the errors become imperceptible to the naked eye.

The shape errors and comparisons for the single vortex test are detailed in Table 8 with a Courant number $C_r = 1$. To assess our method's ability to uphold mass conservation, we compute the relative area error:

$$E_m = \left| \frac{\sum_{i,j} A_{i,j}^0 - \sum_{i,j} \tilde{A}_{i,j}}{\sum_{i,j} A_{i,j}^0} \right|,$$
(46)

where $A_{i,j}^0$ denotes the liquid area in cell (i, j) at t = 0, and $\tilde{A}_{i,j}$ represents the advected liquid area in cell (i, j). Notably, our method outperforms prior methods, except for iPAM, which is an improved version of PAM. In comparison to PAM, our method is free of the high computational costs associated with maintaining and simplifying liquid polygons.

6.3. Zalesak's Disk

The Zalesak's disk advection test, originally proposed by Zalesak [48] and then adopted by Rudman [49], has become a widely recognized benchmark for evaluating



Table 7: Results of Rider–Kothe reversed single vortex, all $C_r = 1$.

the efficacy of various advection schemes in capturing sharp interfaces at different orientations. In the existing literature, this test has two different configurations, which we refer to as config A and config B. In config A, a notched disk centered at (2.0, 2.75) with radius R = 0.5 is initialized inside a $[0, 4]^2$ domain. The notch width is s = 0.06 and the width of the bridge between the left and right halves is r = 0.4 [27]. Therefore, we can compute its initial liquid area $A^0 = 0.7494$. The stream function is expressed as [27]

$$\psi(x,y) = -\frac{\omega}{2}[(x-x_o)^2 + (y-y_o)^2], \qquad (47)$$

Algorithm	Resolution	E_m	E_g	O_n
	32×32		4.78×10^{-2}	
Rider and Kothe $[18]$	64×64		6.96×10^{-3}	2.78
	128×128		1.44×10^{-3}	2.27
	32×32		4.64×10^{-2}	
\mathbf{EMFPA} - \mathbf{SIR} [29]	64×64		$5.94 imes 10^{-3}$	2.97
	128×128		5.39×10^{-4}	3.46
	32×32		$5.78 imes 10^{-3}$	
Double DLIC [24]	64×64		1.77×10^{-3}	1.71
Double-PLIC [24]	128×128		$3.30 imes 10^{-4}$	2.42
	256×256		8.69×10^{-5}	1.93
	64×64	$< 10^{-12}$	1.14×10^{-2}	
NIFPA-1 [31]	128×128	$< 10^{-12}$	2.68×10^{-3}	2.01
	256×256	$< 10^{-12}$	5.37×10^{-4}	2.32
	64×64	4.12×10^{-13}	1.04×10^{-2}	
Owkes and Desjardins [47]	128×128	2.53×10^{-13}	1.34×10^{-3}	1.94
	256×256	1.15×10^{-13}	3.50×10^{-4}	2.22
	32×32		3.81×10^{-2}	
CCII [20]	64×64		4.58×10^{-3}	3.06
000 [30]	128×128		1.00×10^{-3}	2.20
	256×256		$1.78 imes 10^{-4}$	2.59
	32×32		6.21×10^{-4}	
iPAM [40] $(h_L = 0.1h)$	64×64		$1.05 imes 10^{-5}$	2.99
	128×128		1.37×10^{-6}	2.94
	32×32		2.33×10^{-2}	
AMR-MOF [37]	64×64		3.15×10^{-3}	2.88
	128×128		$5.04 imes 10^{-4}$	2.64
	32×32		8.75×10^{-3}	
Edge Cut (propaged)	64×64		$1.15 imes 10^{-3}$	2.93
Eage Out (proposed)	128×128		$1.76 imes 10^{-4}$	2.71
	256×256		4.61×10^{-5}	1.93

Table 8: Shape errors of Rider–Kothe reversed single vortex tests at t = T with different methods and parameters. The results of EMFPA-SIR are taken from [24]. The results of NIFPA-1 are taken from [26]. The mass errors reported in [47] are the absolute differences in the liquid area, and we divide them by πr^2 where r = 0.15 to obtain E_m in the table. where $(x_o, y_o) = (2, 2)$ denotes the center of the computational domain, and $\omega = 0.5$ represents the angular velocity. Consequently, the velocity field is defined by:

$$u = -\omega(y - 2),$$

$$v = \omega(x - 2).$$
(48)

At $T = 4\pi$, the notched disk completes a full circle and returns to its initial position. The advection results are depicted in Figure 16.

On the other hand, the computational domain in config B is $[-0.5, 0.5]^2$, and the disk with R = 0.15, r = s = 0.05, centered at (0, 0.25) [47], has $A^0 = 0.05822$. To compensate for the difference between the two configurations, we define the relative shape error to the initial liquid area A^0 as

$$E_r = \frac{\sum_{i,j} |A_{i,j} - \tilde{A}_{i,j}|}{|\sum_{i,j} A_{i,j}^0|} = \frac{E_g}{A^0}.$$

The relative area error E_m and the relative shape error E_g for the Zalesak's disk test are provided in Table 9. This demonstrates that our proposed algorithm achieves state-of-the-art performance in preserving geometric shapes among VOF-based methods. Especially, our method effectively preserves of the interior notch, despite slight distortions at sharp corners attributable to the simplified representation capacity of the triangle cuts. At the same time, our algorithm also accurately maintains mass conservation. Moreover, our method shows better accuracy at $C_r = 1$ compared to $C_r = 0.25$, because the former requires fewer time steps, thus reducing the accuracy loss caused by interface discretization. Specifically, our algorithm achieves near-perfect mass conservation in most cases, except for the uncommon case 4, as discussed in Section 4.2. Using fewer time steps helps reduce the occurrence of such cases. As a result, our algorithm attains high mass conservation accuracy at $C_r = 1$.

6.4. Deformation Field

This deformation field test, originally proposed by Smolarkiewicz [51], serves as a rigorous assessment of advection methods, evaluating their effectiveness in handling highly deformable flows.

The computational domain is consistent with the Rider-Kothe reversed vortex test, covering the unit square $[0, 1]^2$. Initially, the liquid region is a circle centered at (0.5, 0.5). The governing stream function, which describes the flow velocity field, is expressed as [27]:

$$\psi(x,y) = \frac{1}{n\pi} \sin(n\pi(x+0.5)) \cos(n\pi(y+0.5)) \cos\left(\frac{\pi t}{T}\right),$$
(49)





Figure 16: Zalesak's disk at 200×200 resolution and $C_r = 1$.

Algorithm	C_r	Resolution	E_m	E_r
EMFPA-SIR [29]	0.25	200×200		8.74×10^{-3}
		50×50	4.41×10^{-14}	6.83×10^{-2}
Owkes and Desjardins [47]		100×100	7.93×10^{-13}	2.02×10^{-2}
		200×200	7.78×10^{-12}	9.09×10^{-3}
		50×50		8.96×10^{-2}
THINC/QQ $[50](\beta = 6)$	0.25	100×100		3.22×10^{-2}
		200×200		1.67×10^{-2}
AMR-MOF [37]		200×200		2.32×10^{-4}
PAM [27]		200×200		5.30×10^{-4}
	0.25	50×50		8.74×10^{-3}
iPAM $[40](h_L = 5h^2)$	0.23	100×100		4.00×10^{-4}
		200×200		1.79×10^{-5}
		50×50	4.68×10^{-3}	2.05×10^{-2}
	1.0	100×100	5.30×10^{-10}	7.13×10^{-3}
Edge Cut (proposed)		200×200	1.65×10^{-10}	2.20×10^{-3}
Euge Out (proposed)		50×50	1.65×10^{-2}	4.31×10^{-2}
	0.25	100×100	2.46×10^{-4}	1.43×10^{-2}
		200×200	$5.97 imes 10^{-7}$	4.88×10^{-3}

Table 9: Results of Zalesak's disk test. The result of AMR-MOF is taken from [40]. Owkes & Desjardins [47] and THINC/QQ [50] use config B; other tests use config A.

where T = 2 represents the period, and the parameter n = 4 denotes the number of vortices. As the evolution progresses to t = T, the liquid region undergoes deformation and ultimately returns to its initial state. Consequently, the velocity field is characterized by:

$$u = -\cos\left(\frac{\pi t}{T}\right)\sin(n\pi(x+0.5))\sin(n\pi(y+0.5)),$$

$$v = -\cos\left(\frac{\pi t}{T}\right)\cos(n\pi(x+0.5))\cos(n\pi(y+0.5)).$$
(50)

The shape errors are presented in Table 10, along with visual representations of the advected liquid regions in Figure 17. Notably, our method exhibits an exceptional capability to accurately capture even the intricately thin sections of the liquid.



Figure 17: Deformation field test with 128×128 grid and $C_r = 1$.

6.5. Time Cost

In this section, we will provide the timing data acquired on the 128×128 single vortex test in Section 6.2 and discuss the algorithm's efficiency. Table 11 shows the

Algorithm	C_r	Resolution	E_m	E_g	O_h
		64×64		1.12×10^{-2}	
Rider and Kothe [18]		128×128		$5.95 imes 10^{-3}$	0.91
		32×32		2.60×10^{-2}	
THINC [52]		64×64		1.38×10^{-2}	0.91
		128×128		9.26×10^{-3}	0.58
		64×64		1.07×10^{-2}	
	1.0	128×128		$5.58 imes 10^{-3}$	0.94
CCII [20]		256×256		1.47×10^{-3}	1.93
000 [30]		64×64		1.42×10^{-2}	
	0.1	128×128		7.31×10^{-3}	0.96
		256×256		1.60×10^{-3}	2.19
		64×64	7.11×10^{-5}	7.73×10^{-5}	
	1.0	128×128	7.44×10^{-6}	$7.19 imes 10^{-6}$	3.43
PAM [27]		256×256	7.45×10^{-7}	3.85×10^{-7}	4.22
1 AIVI [27]		64×64	7.16×10^{-6}	$8.89 imes 10^{-5}$	
	0.1	128×128	7.42×10^{-7}	1.41×10^{-5}	2.67
		256×256	9.02×10^{-8}	1.86×10^{-6}	2.92
		64×64		4.31×10^{-5}	
iPAM [40] $(h_L = h^{3/2})$		128×128		5.55×10^{-6}	2.96
		256×256		6.96×10^{-7}	3.00
		64×64	6.25×10^{-5}	1.79×10^{-3}	
	1.0	128×128	1.97×10^{-4}	4.26×10^{-4}	2.07
Edge Cut (proposed)		256×256	1.31×10^{-5}	$7.63 imes 10^{-5}$	2.48
Edge Out (proposed)		64×64	1.39×10^{-3}	2.49×10^{-3}	
	0.1	128×128	1.36×10^{-3}	8.20×10^{-4}	1.60
		256×256	1.12×10^{-4}	$1.46 imes 10^{-4}$	2.49

Table 10: Results of the deformation field test.

Algorithm	Running Time (s)
AMR-MOF [37]	107.36
iPAM [40] $(h_L = 0.1h)$	16.32
CCU [30]	5.98
Owkes and Desjardins [47]	0.17
Edge Cut (proposed)	3.07

Table 11: Comparison of running time between different algorithms. Timing data for iPAM [40] and CCU [30] are scaled for comparison.

comparison of execution time between different methods. Our timing data is obtained from a workstation with a 3.6 GHz Intel i7-12700K CPU and 32 GB RAM, which we used to conduct all experiments. In terms of runtime efficiency, our method outperformed most previous methods with high interface-tracking accuracy, attributed to our simple interface representation data structure and advection algorithm, which is easy to implement and is free of costly operations like optimization processes in MOF, or geometric editing of polygons in PAM.

As a prototype program, our algorithm still has significant room for time optimization. In the interface advection (see Algorithm 4), the most time-consuming part is the polygon-polygon intersection in equation (17), which takes 2.95s(96%)of the runtime, and all other parts only take 0.12s. This is primarily due to our use of the exact geometry computation procedure of CGAL [53] to avoid computing errors when degeneracy cases are found, *e.g.*, an edge in the pre-image is very close to a vertex of the liquid polygon, leading to mistakenly identifying two edge cuts on the connected line segments instead of one. Since exact computations can reduce program speed by 1 to 2 orders of magnitude compared to floating-point geometric operations, especially in cases close to degeneracy [54], we anticipate that the runtime efficiency of our program can be significantly improved by adopting floating-point geometric algorithms that can handle degeneracy cases [55] to a similar level to Owkes and Desjardins[47].

We can further optimize our program by leveraging the fact that methods like MOF and PAM rely on actual polygon-polygon intersections to calculate the moments of fluid, while our algorithm only requires the intersection area for edge cut correction in Algorithm 3, and the triangle edge cuts advection (see Algorithm 1) can be built only by querying segment-segment intersections and cross products. Although Algorithm 2 may require exact polygon-polygon intersections, it is seldom called. In the current implementation, we will calculate these actual intersections in all cases, which can be optimized in the future.

In the current implementation, thread-unsafe features in the CGAL library, such as lazy types [56], significantly hinder the parallelism of our algorithm. However, in our proposed Algorithm 4, each triangle is processed separately, only sharing the liquid polygons $\{\mathcal{P}_i^n\}$ as read-only data. This allows for efficient parallelization using thread-safe features from shared-memory programming platforms like OpenMP [57] or TBB [58].

7. Discussion on 3D Extension

7.1. Extending to 3D Space



Figure 18: Four possible cases with 1 fluid vertex and 3 air vertices. (a) No edges have two valid cuts. One interface is reconstructed in the tetrahedron. (b) One edge has two valid cuts. These two cuts are ignored and the interface reconstruction is the same as (a). (c) Two edges have two valid cuts. Two interfaces are reconstructed. (d) Three edges have two valid cuts. Three interfaces are reconstructed.

In this section, we briefly explore the possibility of extending our algorithm to 3D space. Now consider performing interface reconstruction based on the edge cut information of a tetrahedron in 3D space, assuming that all interfaces are planes inside the tetrahedron. Similar to the discussion in Section 2, exchanging the roles

of liquid and air, or rotating the tetrahedron, does not change the interface inside it. Therefore, among these equivalent cases, we can focus on discussing only one of them. Without loss of generality, we can assume that among the four vertices of the tetrahedron, there are 0, 1, or 2 fluid vertices, while the remaining vertices being air. Next, we will discuss these three cases individually.



Figure 19: Four possible cases with 2 fluid vertices and 2 air vertices. (a) No edges have two valid cuts. One interface is reconstructed from four cut points. (b) One edge has two valid cuts. Two interfaces are reconstructed. (c) Two edges have two valid cuts. Cuts on one of them are ignored and the reconstruction is the same as (b). (d) Another possible reconstruction of (c).

Tetrahedron with 1 fluid vertex. Suppose that the four vertices of the tetrahedron are $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ and \mathbf{v}_4 , and $\chi(\mathbf{v}_1) = 1$ while $\chi(\mathbf{v}_2) = \chi(\mathbf{v}_3) = \chi(\mathbf{v}_4) = 0$. We denote the edge connecting \mathbf{v}_1 and \mathbf{v}_2 as e_{12} , and so on. In this case, there must be one valid cut on edges e_{12} , e_{13} , e_{14} , and in the remaining three edges, there may be $\{0, 1, 2, 3\}$ edges with two valid cuts. These four cases are summarized in Figure 18. In Figure 18b, we have to discard two cuts, introducing some geometric errors. In Figure 18c, one interface is defined by four vertices, and we need to fit a plane based on their positions.

Tetrahedron with 2 fluid vertices. Suppose that $\chi(\mathbf{v}_1) = \chi(\mathbf{v}_2) = 1$ and $\chi(\mathbf{v}_3) = \chi(\mathbf{v}_4) = 0$. There must be one valid cut on edges e_{13} , e_{14} , e_{23} and e_{24} . In the



Figure 20: Four possible cases with 0 fluid vertices and 4 air vertices, where there are $\{0, 1, 2\}$ edges with two valid cuts on them. (a) No edges have two valid cuts. (b) One edge has two valid cuts. (c) Two adjacent edges have two valid cuts. (d) Two non-adjacent edges have two valid cuts. In all cases, edge cuts are ignored and no interface is reconstructed.

remaining two edges e_{12} and e_{34} , there might be $\{0, 1, 2\}$ edges with two valid cuts. Figure 19 summarizes these situations. Notably, ambiguity arises in Figures 19c and 19d that they represent two different interface reconstructions for the same edge cut, which cannot be distinguished based on the edge cut information alone.

Tetrahedron with 0 fluid vertices.. In this case, we have $\chi(\mathbf{v}_1) = \chi(\mathbf{v}_2) = \chi(\mathbf{v}_3) = \chi(\mathbf{v}_4) = 0$, and there might be $\{0, 1, 2, 3, 4, 5, 6\}$ edges that have two valid cuts on them. No interface is reconstructed for $\{0, 1\}$ edges, as shown in Figures 20a and 20b. We will further simplify our discussion utilizing the fact that one tetrahedron edge is adjacent to 4 other edges, and only 1 edge is non-adjacent to it. If there are 2 edges with two valid cuts, they can be either adjacent, like Figure 20c, or non-adjacent, like Figure 20d. In either case, there is no interface within the tetrahedron. Figure 21 discusses three different cases in which 3 edges have two valid cuts. Since an edge is non-adjacent to only one edge in the tetrahedron, there must be two adjacent edges in three, which we assume to be e_{12} and e_{23} . The third edge may be e_{13} (Figure 21a), e_{14} and e_{34} , which are equivalent (Figure 21b), or e_{24} (Figure 21c). Only in the case



Figure 21: Three possible cases with 0 fluid vertices and there are 3 edges with two valid cuts on them. Two of them are edges e_{12} and e_{23} . (a) The third edge is e_{13} . (b) The third edge is e_{14} , which is equivalent to the case of e_{34} . (c) The third edge is e_{24} , where two interfaces are reconstructed.

of e_{24} , an interface is shown in the tetrahedron. Next, if there are 4 edges with two cuts each, it implies that there are 2 edges with no cuts. These two edges can either be adjacent (Figure 22a) or non-adjacent (Figure 22b). Finally, Figures 22c and 22d show the cases that 5 and 6 edges have two valid cuts.

Based on the above discussion, there are a total of 18 basic cases for reconstructing the interface within a tetrahedron using edge cut information. Among these, 8 cases will ignore some cut points, leading to geometric errors in the interface reconstruction. Additionally, if there are 2 fluid vertices and 2 edges with two valid cuts, we encounter an ambiguity where it is not possible to determine whether it corresponds to Figure 19c or Figure 19d. While extending the algorithm to \mathbb{R}^3 is non-trivial due to these reasons, We anticipate addressing the aforementioned challenges in future work.

8. Conclusion

In this paper, we proposed an Eulerian interface tracking algorithm for unstructured triangle meshes based on the triangle edge cut interface representation. We also designed an area correction algorithm to further improve the accuracy of mass conservation. Our interface representation method features a low level of memory usage that can capture sub-triangle geometric features with 6 DoFs in a triangle. The proposed interface advection algorithm, combined with area correction, can track the dynamic evolution of the interface on a low-cost, fully definitive basis, without any expensive optimization process. Our algorithm handles each triangle cell independently in a parallelization-friendly manner. On different numerical tasks, it's proven that our method outperforms traditional VOF methods, offering a promising



Figure 22: Four possible cases with 0 fluid vertices and $\{4, 5, 6\}$ edges with two valid cuts on them. (a) 4 edges have two valid cuts and the other two adjacent edges have no valid cuts. (b) 4 edges have two valid cuts and the other two non-adjacent edges have no valid cuts. (c) 5 edges have two valid cuts. (d) 6 edges have two valid cuts.

approach for accurate and efficient interface tracking in complex numerical simulations.

The algorithm proposed in this paper requires an unstructured triangle mesh because applying this method to a lattice grid can introduce ambiguities due to the rectangular cells. For example, the two cases in Figure 23 have identical vertex materials and edge cuts, however exhibit different liquid polygons, leading to ambiguity. Another limitation is that the triangle edge cut representations will smooth out sharp corners inside triangles, producing some artifacts, which may be alleviated by adding more vertices within the triangles. Furthermore, the use of CGAL's exact geometry computation slows down runtime and hinders parallelization.

In the future, we plan to address the dimensionality challenges discussed in Section 7 and extend our proposed edge cut-based interface tracking method to 3D space. To enhance efficiency, we aim to adopt faster floating-point geometric algorithms to reduce the computational cost of polygon-polygon intersections and further accelerate the program through parallelization. We also intend to integrate the proposed algorithm with physical simulations to accurately model fluid phenomena with thin



Figure 23: Ambiguities of rectangular cell edge cut reconstruction.

structures.

9. Acknowledgment

Georgia Tech authors acknowledge NSF IIS #2433322, ECCS #2318814, CA-REER #2433307, IIS #2106733, OISE #2433313, and CNS #1919647 for funding support.

References

- [1] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct numerical simulations of gasliquid multiphase flows, Cambridge university press, 2011.
- [2] H. Luo, J. D. Baum, R. Löhner, On the computation of multi-material flows using ale formulation, Journal of Computational Physics 194 (1) (2004) 304–328.
- [3] J. Pan, T. Long, L. Chirco, R. Scardovelli, S. Popinet, S. Zaleski, An edge-based interface tracking (ebit) method for multiphase-flows simulation with surface tension, arXiv preprint arXiv:2309.00338 (2023).
- [4] S. Shin, D. Juric, Modeling three-dimensional multiphase flow using a level contour reconstruction method for front tracking without connectivity, Journal of Computational Physics 180 (2) (2002) 427–470.
- [5] S. Chen, D. B. Johnson, P. E. Raad, D. Fadda, The surface marker and micro cell method, International Journal for Numerical Methods in Fluids 25 (7) (1997) 749–778.
- [6] S. Popinet, S. Zaleski, A front-tracking algorithm for accurate representation of surface tension, International Journal for Numerical Methods in Fluids 30 (6) (1999) 775–793.

- [7] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.-J. Jan, A front-tracking method for the computations of multiphase flow, Journal of computational physics 169 (2) (2001) 708–759.
- [8] S. O. Unverdi, G. Tryggvason, A front-tracking method for viscous, incompressible, multi-fluid flows, Journal of computational physics 100 (1) (1992) 25–37.
- [9] X. Chen, J. Lu, S. Zaleski, G. Tryggvason, Characterizing interface topology in multiphase flows using skeletons, Physics of Fluids 34 (9) (2022).
- [10] L. Chirco, J. Maarek, S. Popinet, S. Zaleski, Manifold death: a volume of fluid implementation of controlled topological changes in thin sheets by the signature method, Journal of Computational Physics 467 (2022) 111468.
- [11] S. Osher, R. Fedkiw, K. Piechor, Level set methods and dynamic implicit surfaces, Appl. Mech. Rev. 57 (3) (2004) B15–B15.
- [12] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations, Journal of computational physics 79 (1) (1988) 12–49.
- [13] S. Osher, R. P. Fedkiw, Level set methods: an overview and some recent results, Journal of Computational physics 169 (2) (2001) 463–502.
- [14] F. Gibou, R. Fedkiw, S. Osher, A review of level-set methods and some recent applications, Journal of Computational Physics 353 (2018) 82–109.
- [15] C. W. Hirt, B. D. Nichols, Volume of fluid (vof) method for the dynamics of free boundaries, Journal of computational physics 39 (1) (1981) 201–225.
- [16] J. U. Brackbill, D. B. Kothe, C. Zemach, A continuum method for modeling surface tension, Journal of computational physics 100 (2) (1992) 335–354.
- [17] R. M. Chiodi, Advancement of numerical methods for simulating primary atomization, Cornell University, 2020.
- [18] W. J. Rider, D. B. Kothe, Reconstructing volume tracking, Journal of computational physics 141 (2) (1998) 112–152.
- [19] R. Scardovelli, S. Zaleski, Direct numerical simulation of free-surface and interfacial flow, Annual review of fluid mechanics 31 (1) (1999) 567–603.

- [20] S. Popinet, An accurate adaptive solver for surface-tension-driven interfacial flows, Journal of Computational Physics 228 (16) (2009) 5838–5866.
- [21] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, Interface reconstruction with least-squares fit and split advection in three-dimensional cartesian geometry, Journal of Computational Physics 225 (2) (2007) 2301–2319.
- [22] D. Gueyffier, J. Li, A. Nadim, R. Scardovelli, S. Zaleski, Volume-of-fluid interface tracking with smoothed surface stress methods for three-dimensional flows, Journal of Computational physics 152 (2) (1999) 423–456.
- [23] R. DeBar, Fundamentals of the kraken code, Technical Report UCIR-760 (1974).
- [24] J. Lopez, J. Hernandez, P. Gomez, F. Faura, An improved plic-vof method for tracking thin fluid structures in incompressible two-phase flows, Journal of Computational Physics 208 (1) (2005) 51–74.
- [25] C. Mulbah, C. Kang, N. Mao, W. Zhang, A. R. Shaikh, S. Teng, A review of vof methods for simulating bubble dynamics, Progress in Nuclear Energy 154 (2022) 104478.
- [26] T. Marić, D. B. Kothe, D. Bothe, Unstructured un-split geometrical volume-offluid methods-a review, Journal of Computational Physics 420 (2020) 109695.
- [27] Q. Zhang, P. L.-F. Liu, A new interface tracking method: The polygonal area mapping method, Journal of Computational Physics 227 (8) (2008) 4063–4088.
- [28] J. E. Pilliod Jr, E. G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces, Journal of Computational Physics 199 (2) (2004) 465–502.
- [29] J. López, J. Hernández, P. Gómez, F. Faura, A volume of fluid method based on multidimensional advection and spline interface reconstruction, Journal of Computational Physics 195 (2) (2004) 718–742.
- [30] R. Comminal, J. Spangenberg, J. H. Hattel, Cellwise conservative unsplit advection for the volume of fluid method, Journal of computational physics 283 (2015) 582–608.
- [31] C. B. Ivey, P. Moin, Conservative and bounded volume-of-fluid advection on unstructured grids, Journal of Computational Physics 350 (2017) 387–419.

- [32] V. Dyadechko, M. Shashkov, Moment-of-fluid interface reconstruction, Los Alamos Report LA-UR-05-7571 (2005) 49.
- [33] A. A. Mukundan, T. Ménard, J. C. B. de Motta, A. Berlemont, A 3d moment of fluid method for simulating complex turbulent multiphase flows, Computers & Fluids 198 (2020) 104364.
- [34] H. Anbarlooei, K. Mazaheri, Moment of fluid interface reconstruction method in multi-material arbitrary lagrangian eulerian (mmale) algorithms, Computer methods in applied mechanics and engineering 198 (47-48) (2009) 3782–3794.
- [35] M. Shashkov, E. Kikinzon, Moments-based interface reconstruction, remap and advection, Journal of Computational Physics 479 (2023) 111998.
- [36] M. Cutforth, P. T. Barton, N. Nikiforakis, An efficient moment-of-fluid interface tracking method, Computers & Fluids 224 (2021) 104964.
- [37] H. T. Ahn, M. Shashkov, Adaptive moment-of-fluid method, Journal of Computational Physics 228 (8) (2009) 2792–2821.
- [38] J. Nocedal, S. J. Wright, Numerical optimization, Springer, 1999.
- [39] Q. Zhang, P. L.-F. Liu, Hypam: A hybrid continuum-particle model for incompressible free-surface flows, Journal of Computational Physics 228 (4) (2009) 1312–1342.
- [40] Q. Zhang, A. Fogelson, Fourth-order interface tracking in two dimensions via an improved polygonal area mapping method, SIAM Journal on Scientific Computing 36 (5) (2014) A2369–A2400.
- [41] P. Karnakov, S. Litvinov, P. Koumoutsakos, A hybrid particle volume-of-fluid method for curvature estimation in multiphase flows, International Journal of Multiphase Flow 125 (2020) 103209.
- [42] L. Chirco, S. Zaleski, An edge-based interface-tracking method for multiphase flows, International Journal for Numerical Methods in Fluids 95 (3) (2023) 491– 497.
- [43] T. C. Hales, Jordan's proof of the jordan curve theorem, Studies in logic, grammar and rhetoric 10 (23) (2007) 45–60.

- [44] J. R. Shewchuk, Triangle: Engineering a 2d quality mesh generator and delaunay triangulator, in: Workshop on applied computational geometry, Springer, 1996, pp. 203–222.
- [45] J. R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, Computational geometry 22 (1-3) (2002) 21–74.
- [46] K. Crane, Discrete differential geometry: An applied introduction, Notices of the AMS, Communication 1153 (2018).
- [47] M. Owkes, O. Desjardins, A mass and momentum conserving unsplit semilagrangian framework for simulating multiphase flows, Journal of Computational Physics 332 (2017) 21–46.
- [48] S. T. Zalesak, Fully multidimensional flux-corrected transport algorithms for fluids, Journal of computational physics 31 (3) (1979) 335–362.
- [49] M. Rudman, Volume-tracking methods for interfacial flow calculations, International journal for numerical methods in fluids 24 (7) (1997) 671–691.
- [50] B. Xie, F. Xiao, Toward efficient and accurate interface capturing on arbitrary hybrid unstructured grids: The thinc method with quadratic surface representation and gaussian quadrature, Journal of Computational Physics 349 (2017) 415–440.
- [51] P. K. Smolarkiewicz, The multi-dimensional crowley advection scheme, Mon. Wea. Rev 110 (12) (1982) 1968–1983.
- [52] F. Xiao, Y. Honma, T. Kono, A simple algebraic interface capturing scheme using hyperbolic tangent function, International journal for numerical methods in fluids 48 (9) (2005) 1023–1040.
- [53] P. Alliez, A. Fabri, Cgal: the computational geometry algorithms library, in: ACM SIGGRAPH 2016 Courses, 2016, pp. 1–8.
- [54] T. Bartels, V. Fisikopoulos, Fast robust arithmetics for geometric algorithms and applications to gis, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 46 (2021) 1–8.
- [55] Y. Wang, C. Jiang, C. A. Schroeder, J. Teran, An adaptive virtual node algorithm with robust mesh cutting., in: Symposium on Computer Animation, Vol. 77, 2014, p. 85.

- [56] M. de Matos Menezes, S. V. G. Magalhães, W. R. Franklin, M. A. de Oliveira, R. E. B. Chichorro, Accelerating the exact evaluation of geometric predicates with gpus, 28th International Meshing Roundtable, Buffalo, NY, USA 16 (2019).
- [57] R. Chandra, Parallel programming in OpenMP, Morgan kaufmann, 2001.
- [58] M. Voss, R. Asenjo, J. Reinders, Pro TBB: C++ parallel programming with threading building blocks, Vol. 295, Springer, 2019.