Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's thesis

Curvature-Minimizing Interface Reconstruction

Matthias Bauer

Course of Study:	Informatik, M.Sc.
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Alexander Straub, M.Sc. Prof. Dr. Filip Sadlo
Commenced:	January 07, 2019
Completed:	July 07, 2019

Acknowledgment

Here I would like to take the time and thank everyone who gave their support either mentally or technically over the past preceding years of my life.

Alexander Straub and Prof. Filip Sadlo for supporting this work and giving their technical feedback.

All lecturers of the University of Stuttgart who gave me the opportunity to learn a lot from different topics.

All fellow students who helped me to overcome many study related challenges and with whom I really enjoyed spending time.

And especially my family and friends who are the most important people in my life.

Abstract

The analysis of free surfaces is important to better understand different chemical and physical effects. Free surfaces exist at the interface between two materials (phases) with different densities. Due to that the surface can move almost freely directed by the phase with the higher density.
For reconstruction and tracking of these surfaces different approaches exist. One of these is the Volume-Of-Fluid method in which the domain is separated into cells containing a fractional value. Based on these values algorithms can track volume changes over time by reconstructing a geometrical representation of the surface and deriving shifts of volume fractions between cells.
Often a surface reconstruction based on the volume fractions is therefore necessary to approximate the evolution of the surface using geometrical properties. Only few of the available methods focus on an accurate visualization.
This work introduces a new approach for reconstructing and visualizing interfaces between two phases (e.g. fluid and gas) in 3D on rectilinear grids. The goal is to create a precise mesh representation by combining the Marching Cubes algorithm with the Volume-Of-Fluid approach. First, basic operations that are needed to approximate the fractional values inside the cells and reduce the overall mean curvature of the mesh are defined. After that different schemes to combine these operations as well as necessary extensions are discussed and tested in order to achieve the most accurate result possible. This results in an extended Laplacian method used for mesh smoothing. Cell volume correction is done using a Newton approach which shifts vertices along gradients scaled by volume differences. Conditional refinement of mesh triangles is added to improve the overall result.
It is shown that for simple cases the approach works as intended. Yet, the overall runtime and existing boundary cases necessitates further adaptations so that it can be used on a regular basis in different scientific scenarios.

Freie Oberflächen existieren an der Schnittstelle zwischen zwei Materialien (Phasen) mit unterschiedlicher Dichte, wobei das Verhalten maßgeblich durch die Phase mit höherer Dichte bestimmt wird. Die Analyse von solchen Oberflächen ist notwendig um verschiedene chemische und physikalische Effekte besser zu verstehen.
Für die Rekonstruktion dieser Oberflächen gibt es verschiedene Ansätze. Einer davon ist das sogenannte Volume-of-Fluid Verfahren. Bei diesem wird die zugrundeliegende Domäne in Zellen aufgeteilt. Diese Zellen enthalten Bruchwerte, welche die Volumenanteile der jeweiligen Phase bestimmen. Durch eine geometrische Darstellung der Oberfläche können zeitliche Verschiebungen von Volumenanteilen zwischen Zellen abgeleitet werden. Dabei konzentrieren sich nur wenige der verfügbaren Methoden rein auf eine akkurate Visualisierung des Datensatzes.
Diese Arbeit stellt einen neuen Ansatz zur Rekonstruktion und Visualisierung für zwei Phasen (z.B. Flüssigkeit und Gas) in 3D auf rechtwinkligen Gittern vor. Ziel ist eine präzise Representation der freien Oberfläche durch eine Kombination des Marching Cubes Algorithmus mit dem Volume-Of-Fluid Verfahren. Es werden zunächst grundlegende Operationen definiert, welche für die Korrektur der Volumenanteile in den Zellen und die Minimierung der mittleren Krümmung des Oberflächennetzes erforderlich sind. Danach werden verschiedene Schemata zur Kombination dieser Operationen sowie notwendige Erweiterungen diskutiert und getestet, um ein möglichst genaues Ergebnis zu erzielen. Daraus resultiert die Verwendung einer erweiterten Laplace-Methode, um das Oberflächenetztes zu glätten. Die Korrektur der Zellvolumen erfolgt durch einen Newton-Ansatz, der Knoten entlang von Gradienten verschiebt. Diese werden durch errechnete Volumenunterschiede skaliert. Um das Gesamtergebnis zu verbessern werden Oberflächendreiecke zusätzlich stufenweise verfeinert.
Es wird gezeigt, dass der Ansatz in einfachen Fällen das geplante Verhalten zeigt. Die notwendige Gesamtlaufzeit und einige Randfälle erfordern jedoch weitere Änderungen damit eine Anwendung in verschiedenen wissenschaftlichen Bereichen möglich ist.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CGAL Computational Geometry Algorithms Library. 56

GUI Graphical User Interface. 54

MC Marching Cubes. 17

PLIC Piecewise Linear Interface Calculation. 17

RK Runge-Kutta. 69

SLIC Simple Line Interface Calculation. 19

VOF Volume-Of-Fluid. 17

VOF field Volume-of-Fluid value field. 18

VOF value fractional volume value. 17

VTK Visualization Toolkit. 54

WENO Weighted Essentially Nonoscillatory Scheme. 60

1 Introduction

Free surfaces depict interfaces between so-called phases. Each one consists of a homogeneous mix of molecules leading to volumes with same density and other similar physical and chemical properties [MR74]. Yet, the phases differ strongly with respect to their individual behavior. This is the case for, e.g., a fluid like water and a gas like air [SP00], see figure 1.1. Here the movement of the interface is directed mainly by the fluid phase [Rud97]. Further investigation of interface interactions and their evolution is thus important to understand various physical and chemical phenomena [SEG+15].
For the simulation and visualization of free surfaces different methods exist. Each one with it's own advantages and disadvantages [LZG+08]. One of these methods is the so called Volume-Of-Fluid (VOF) [HN81] method which is often used in practice because it is easy to use and the underlying data structure can be implemented efficiently [LZG+08] [AP91] [DY18].
In VOF methods a fractional volume value (VOF value) defines how much space of a cell is occupied by a certain phase [AP91]. This is further explained in section 3.1. To track surfaces the amount of volume fraction which is shifted between cells can be measured. For that the current surface along with it's geometrical properties must be reconstructed [HF00]. These reconstructions can often be used to visualize the surface. This is the case, e.g., for the well known Piecewise Linear Interface Calculation (PLIC) approach by Youngs [You82] which is explained in section 2.
Like it can be seen in section 2 most existing approaches need further adaptions in order to use them for the visualization of 3D datasets. Cell elements are created independently and a closed mesh representation of the surface is not achieved. In this work a new approach based on the Marching Cubes Marching Cubes (MC) algorithm [LE87] to visualize two-phase 3D VOF datasets is presented and validated. The goal is to use the MC algorithm on two-phase VOF datasets to obtain a closed surface mesh which is then adapted to achieve a close approximation to the VOF values in cells and reduce the overall mean curvature. To do that different operations are executed

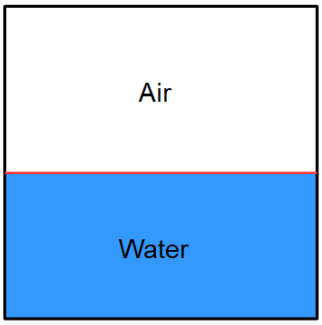


Figure 1.1: The interface between the fluid phase depicted by water and the gas phase depicted by air is a typical example of a free surface. The red line shows the free surface. Both phases have a high difference in density.

iteratively in combination on the mesh. These are mesh smoothing and refinement, cell volume and gradient calculation on the Volume-of-Fluid value field (VOF field). However, surface tracking over multiple timesteps or multi-phase cases are not discussed. For that it can be referred to the approaches presented in section 2.
In the first part of this work basic definitions and operations which are needed to adapt the mesh are introduced and first results are shown. The next part consists of implementation specific details. After that different schemes and extensions which are needed to combine the defined operations are tested. In the end a discussion of the final results and remarks for further necessary improvements are given.
In the next section existing approaches to reconstruct interface surfaces using VOF datasets are discussed and a distinction to the approach presented in this work is given.

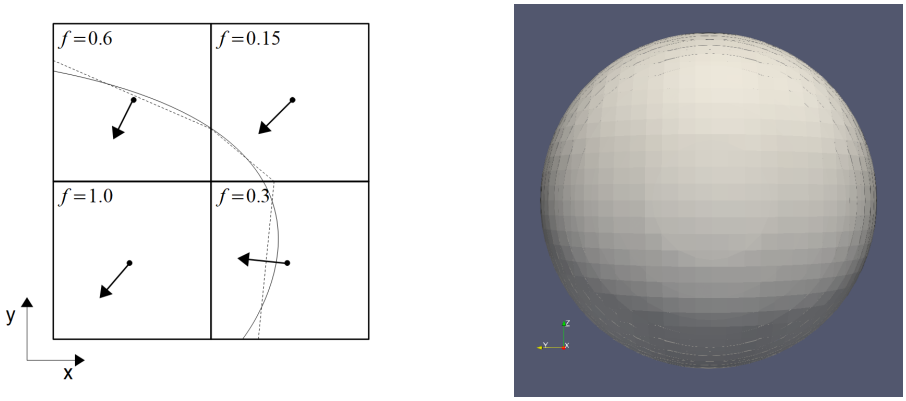
2 Related work

To reconstruct surfaces defined by VOF datasets (see next section) different approaches can be found in the literature. However, some of them are explained only in the case of volume flow between cells in a 2D grid. In general the extension of such 2D schemes to 3D leads to an increase in complexity of the underlying geometry [LRL+06]. Thus, the underlying ideas are depicted as they were presented by their authors in 2D or 3D respectively. An additional generalization to 3D is not given. The goal is to foster the understanding of possible reconstructing techniques for VOF surfaces and how these differ from the one in this work.
Most of the methods are based on the so called PLIC technique in which cell elements consist of linear elements such as lines (2D) or planes (3D). The widely known PLIC approach presented by Youngs [You82] uses a local approximation of the VOF value for each cell individually and is quite accurate and fast [LZG+08]. In the context of this work it is used in order to compare the volume approximation and effects of different operations applied on the mesh visually. It is therefore discussed in more detail.
In the two-phase 3D case of Youngs' [You82] approach a plane is defined at the center of the cell with the normal being equal to the gradient of the respective VOF field[1]. The gradient can be approximated using a finite difference scheme similar to the one depicted in section 3.3. A plane therefore separates the cell into two volumes. It is further shifted along the gradient until the volume created by the plane and the part of the cell lying in one phase, e.g., the fluid, equals the volume fraction depicted by the VOF value. In figure 2.1b a 3D dataset visualized using PLIC can be seen. In 2D a cell centered line is used which is then shifted by a 2D gradient.
A different approach with respect to the orientation of cell elements was presented by Noh and Woodward [NW76] and is called Simple Line Interface Calculation (SLIC). For each cell containing multiple phases (here only two-phases are considered) interfaces are depicted using horizontal and vertical lines (2D case) parallel or orthogonal to the coordinate axes. Each dimension is treated differently which yields multiple outputs for a dataset. PLIC allows arbitrary oriented planes or lines (piecewise linear) while SLIC demands orthogonal or parallel elements related the coordinate axes (piecewise constant) [HF00]. The method presented by Hirt and Nichols [HN81] works similar to SLIC but calculates the surface normal in a neighborhood of nine cells. Depending on the components of the normal the surface is set to be horizontal or vertical. Harvie and Fetcher [HF00] developed the stream scheme in which they use a similar interface reconstruction technique like PLIC. The interface is depicted by points lying on the edge of neighboring cells. In 2D these are connected by a line and shifted using some determined logic with respect to the interpolated gradient of the VOF field until the underlying area fits the VOF value.
Methods to increase the accuracy of the PLIC approach exist as well. Ashgriz and Poo [AP91] introduced the FLAIR (Flux Fine-Segment Model for Advection and Interface Reconstruction) method. It is based on lines (2D case) which are located at cell boundaries. These lines form a

[1] Spatial distribution of all VOF values. See section 3.1.

trapezoid which encloses an area equal to the VOF value of the cell. They further develop a criterion for determining the line's slope depending on the VOF values of neighboring cells. This approach shows a better accuracy as PLIC or SLIC since neighborhood information is included in the slope calculation. The methods presented by Miller and Colella [MC02] and by Liovic et al. [LRL+06] are also based on planar cell (PLIC) elements. Yet, by using cell stencils they achieve a higher degree of accuracy in terms of the PLIC elements' orientation. Both approaches are computational more expensive [LZG+08] since multiple cells are considered for each element's calculation. Lopez et al. [LZG+08] introduced a 3D PLIC approach using cubic-Bezier interpolation in order to increase the accuracy of the plane normals.
In some cases the VOF approach is improved by combining it with other techniques. Scheufler and Roenby [SR18] presented an efficient 2D and 3D PLIC method to reconstruct interfaces based on cell-wise isosurface extraction. After an initial guess of the isovalue a reconstructed distance function is defined over PLIC elements contained in adjacent cells. An iterative scheme is then used to further adapt the isosurface. This is done by altering the element's normal and position with respect to the gradient of the distance function and isovalue to fit the VOF value. Sussman and Puckett [SP00] combined a level set function [2] with the VOF approach in order to improve the accuracy of computed local curvature values.
Further methods to reduce the computation time for surface reconstruction based on PLIC can be found as well. Dai and Tong [DY18] improved an existing analytical 2D algorithm which is based on the separation of polygonal cells into geometric shapes (triangles and trapezoids). For a given interface surface they developed an efficient method for calculating the intersection points of lines parallel to the surface with the polygon edges. These are then used in order to correct the cell volume through defined geometric shapes. Skarysz et al. [SGD18] proposed a 3D method in which the reconstruction is based on the decomposition of cells into tetrahedra. This results in simplified intersection patterns and the final plane position in each tetrahedron is found by a combined iterative solution using Newton (see section 4.1) and a quadratic approach. They have shown that the interface can be calculated faster than it is the case with other existing methods.
Most of the presented approaches approximate the VOF value inside the cell using piecewise linear elements (lines, planes etc.). Connectivity with respect to adjacent cells is not introduced. Further, an exact representation of the underlying VOF field flow cannot always be achieved. This is the case since linear segments are not able to accurately represent curved regions in the dataset. Moreover, complex flow patterns inside cells like, e.g., a bend of the interface cannot be depicted. See figure 2.1a for an example. Only by using a higher domain resolution and therefore smaller cells a more accurate result can be achieved. This is the case for any of the methods presented above.
The main idea of this work is to create a closed surface mesh using 3D two-phase rectilinear VOF datasets which is then evolved iteratively using different operations. To be more precise the mesh is created using the Marching Cubes algorithm and then adapted with respect to the cell volume faction defined by the VOF value and reduced mean curvature. The result is a precise approximation to the flow of the VOF field that depicts the interface. To this end a similar surface reconstruction method yielding mesh connectivity based on the Marching Cubes algorithm could not be found. However, advection calculation or volume transport over cell boundaries is not part of this work.
The basic operations needed to create and adapt the mesh are defined in the next section. These

[2] Here, the level set function is defined as being greater than zero inside the fluid, smaller than zero outside and zero directly on the free surface.



(a) **(b)**

Figure 2.1: Example of PLIC case and result:
(a) Example case of a 2D interface reconstructed using PLIC (dashed line) [You82]. The black arrows indicate the cell centeredVOF field gradients. The curved line depicts the underlying VOF field and it can be seen that the approximation is not exact with respect to the curve. Here, areas in cells are not correct but approximated to foster the understanding of the method. The cell centered VOF values are shown at the top left corner.
(b) Result for a sphere dataset ($51x51x51$ nodes) and PLIC. It can be seen that the created planes (piecewise linear segments) per cell are not connected.

are then tested with different schemes to achieve the best possible result. That is a surface mesh with low mean curvature and precise VOF value approximation.

3 Basic definitions and operations

The goal of this work is to develop an iterative approach based on the MC algorithm for visualization of two-phase 3D VOF datasets. The mean curvature of the surface mesh should be reduced and the correct VOF value with respect to the enclosed volume in cells approximated. To do that different operation and techniques must be applied on the dataset:

1. The Marching Cubes algorithm to get an isosurface approximation of the cell centered VOF field as basic surface mesh. This mesh is the foundation for following operations. See section 3.2.
2. VOF gradient calculation at mesh vertices to further shift them for volume correction. See section 3.3.
3. Calculation of the curvature at surface mesh vertices to verify the result of the smoothing operation. See section 3.4.
4. Laplacian smoothing or a similar technique to minimize the mean curvature on the surface mesh. See section 3.5.
5. Mesh refinement to retrieve a higher mesh resolution and thus more vertices that can be shifted to sample the VOF field. This leads to a preciser interface approximation. See section 3.6.
6. Calculation of the volume which is enclosed by the surface mesh and the part of the cell inside the fluid phase. It is used to determine if the volume inside cells needs further adaption to yield the final VOF value. See section 3.7.

In the following sections of this work the used VOF datasets are defined followed by cases one to six with first results and remarks.

3.1 Volume of Fluid datasets

Datasets based on the VOF method can be created artificially or by running simulations. While the former can be used to test certain conditions or boundary cases the latter can be used to investigate different phenomena in real life [SEG+15]. In this work datasets in which the free surface is artificially shaped as a sphere where used to test the basic functionality and correctness of different operations. This is, e.g., the case for the basic implementation of smoothing in section 3.5 and the adapted Marching Cubes [LE87] algorithm in section 3.2. More complex datasets were created by simulating the behavior of colliding fluid droplets in a gas using Free Surface 3D [SEG+15]. Free Surface 3D is a solver for computational fluid dynamics of incompressible multi-phase flows based on pressure, velocity and a VOF value for each cell. The resulting dataset structure is therefore more complex and it can be used for further convergence, speed and robustness checking as it was

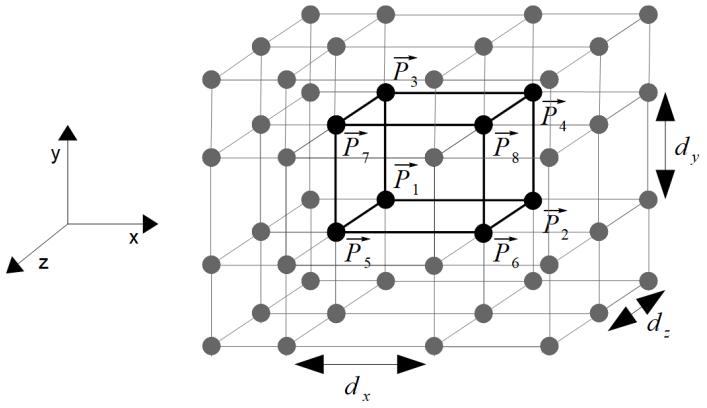
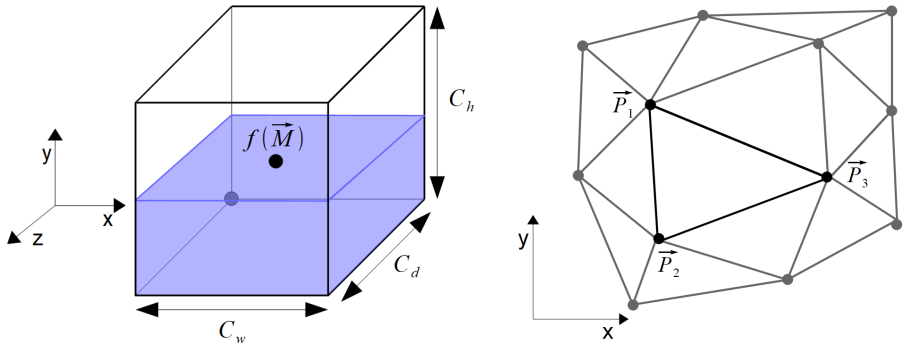


Figure 3.1: Example of a rectilinear grid with cell connection. Each cell inside a grid has eight nodes $\overrightarrow{P_1}$ to $\overrightarrow{P_8}$ (black points) connected by edges (black lines) being parallel to one of the coordinate axis. The origin of the cell lies at $\overrightarrow{P_1}$. The width, height and depth of the cell is defined by the distances d_x, d_y and d_z between connected nodes in each direction.

done in section 4.
In general, to create VOF datasets the underlying domain is separated into discrete elements represented by cells. Like it was mentioned in section 1 fractional values (VOF values) are used to depict how much volume of the cell is occupied by a phase [AP91]. Depending on the source of the data further intersection tests of the surface and the cells are necessary in order to determine the enclosed volume and thus VOF value of the phases inside cells. Yet, the detailed process of creating such datasets is not discussed here.
In this work only two-phase flows are considered which yields two VOF values per cell depicting the fraction of volume V_1 for phase one and V_2 for phase two. That is, e.g., the case for a gas (air) and a fluid (water). Since the VOF value can only have a value between zero and one it is enough to save one VOF value $f = f_2$ per cell to depict the cell's V_2 volume for the fluid phase, see figure 3.2a for an example. The VOF value f_1 can then be calculated using $f_1 = 1.0 - f_2$
The cell's structure is defined by the underlying grid type. A grid consists of connected nodes which are located at the corners of adjacent cells. Different grid types have different topological or geometrical properties regarding the positioning of nodes. In this work only 3D rectilinear and uniform grids are considered. Each node has therefore six connected neighboring nodes. The cells formed by these grids are bounded by eight nodes $\overrightarrow{P_1}$ to $\overrightarrow{P_8}$ which are connected by sixteen edges each one being parallel to one of the coordinate axis, see figure 3.2. In the case of uniform grids the distance d between adjacent nodes along the x,y and z axis are the same hence $d_z = d_y = d_z$. For rectilinear grids these distances can vary which could also yield $d_x \neq d_y \neq d_z$. Yet, the distance is the same for every parallel edge lying in the same segment as the cell, see figure 3.1 for an example. It is further possible to calculate each node position $\overrightarrow{P}$ by summing up the length for each segment



(a) **(b)**

Figure 3.2: VOF cell orientation and mesh example:
(a) Example of a cell inside the VOF dataset. The blue area depicts the volume V_2 occupied by the fluid phase. The VOF value f is stored at the center of the cell (black point) at $\vec{M}$ and is 0.5. This results in V_2 being equal to the cell's half volume defined by $0.5 \cdot C_w \cdot C_h \cdot C_d$. The gray point depicts the cell's origin.
(b) Extract of a mesh in 2D. The mesh consists out of triangles defined by three vertices $\vec{P_1}$, $\vec{P_2}$ and $\vec{P_3}$. Edges and vertices are shared by multiple triangles.

and dimension starting at the origin $(0,0,0)^T$. Here, the position (origin) of each cell C is defined by the node P with lowest x, y and z value

$$C_x = \min \vec{P}_{jx} : j \in [1,8] \tag{3.1}$$

$$C_y = \min \vec{P}_{jy} : j \in [1,8] \tag{3.2}$$

$$C_z = \min \vec{P}_{jz} : j \in [1,8] \tag{3.3}$$

where $\vec{P}_{jk}$ is the component k of the node with index j and C_k is the component k of the cell's origin. See figure 3.1 for an example of a cell and it's node arrangement. The VOF value $f(\vec{M})$ for the cell C is saved at the cell center $\vec{M}$ at the position

$$\vec{M} = \left(C_x + \frac{C_w}{2.0}, C_y + \frac{C_h}{2.0}, C_z + \frac{C_d}{2.0}\right) \tag{3.4}$$

where C_w is the cell's width, C_h is the cell's height and C_d is the cell's depth, hence $C_w = d_x$, $C_h = d_y$ and $C_d = d_z$ at the corresponding segment, see figure 3.2a for a depiction of the cell centered VOF value and resulting volume. Here, the indices x, y and z relate to the VOF value saved at the center $\vec{M}$ of the cell in the respective segments in 3D, thus $f(\vec{M}) = f(x,y,z)$. E.g., $f(0,0,0)$ is the VOF value for the cell that has it's origin located at the domain's origin (first segment for each dimension). Here the spatial distribution of all VOF values is called VOF field and depicts the VOF value for every point in space. Yet, since VOF values are only defined at cell centers interpolation is needed to get the values at arbitrary positions. To depict the free surface the resulting mesh must be fitted to the flow of the VOF field with respect to the cell centered VOF value. Other grid types such as curvilinear grids need further adaptions of the operations listed in section 3. This, however, is not part of this work.
The chosen grid resolution and type depend on the way the dataset is created and the goal of

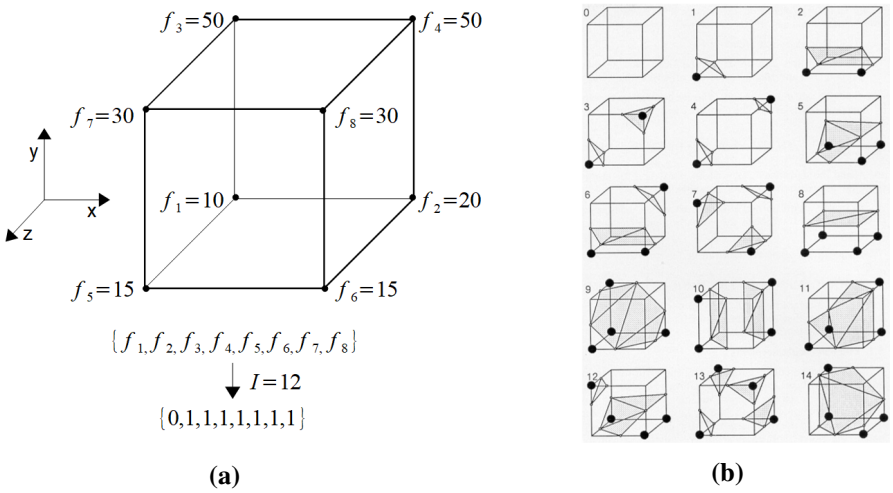


Figure 3.3: MC binary vector and triangle patterns:
(a) Example for the determination of the binary vector. For an isovalue of $I = 12$ the nodes with a value above and below are defined and this information is saved in the binary vector. It has defined positions for every node in the cell.
(b) Possible triangle patterns to approximate the isosurface inside the cell [LE87].

application. Choosing a high grid resolution (number of nodes within the domain) can yield a more accurate reconstruction of the surface but is computationally expensive.
The approach presented in this work is based on operations which are executed on a surface mesh created from VOF datasets. This mesh represents a free surface and consists of triangles which are defined by three vertices $\overrightarrow{P_1}$, $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$ that are connected by edges, see figure 3.2b. Each triangle has multiple adjacent triangles sharing only one vertex or two vertices with an edge.
The generation of the surface mesh along with the mesh based operations are discussed in the following sections. Intermediate results are represented as triangle mesh.

3.2 Surface extraction

In the following section the approach for retrieving the basic surface mesh from the VOF dataset is described. First the surface extraction algorithm is explained followed by details on how the algorithm was adapted to the VOF dataset. After that a method for closing the resulting surface mesh is shown. Results are discussed at the end.

3.2.1 Marching Cubes algorithm

A closed surface mesh must be created from the underlying VOF dataset as basis for further operations. For that the VOF field can be defined as a 3D scalar function $f(x,y,z)$ with $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ that assigns a position in space to a VOF value. This can then be used to define an isosurface S of the VOF field f for a given isovalue I as

$$S = \left\{ \overrightarrow{a} \,|\, f(\overrightarrow{a}) = I \right\} \tag{3.5}$$

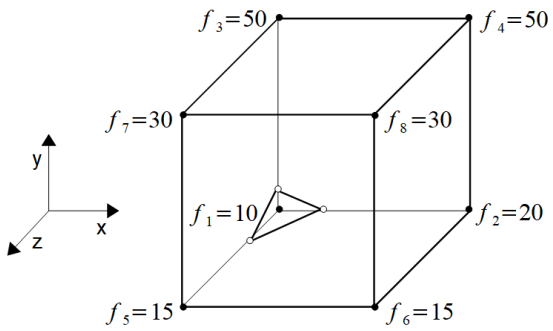


Figure 3.4: Result of the MC algorithm for one cell. For an isovalue $I = 12$ pattern one (see figure 3.3b) is chosen and the points are interpolated along the edges which do contain a point on the isosurface (white points).

Therefore, all points $\overrightarrow{a} = (x, y, z)^T$ in the VOF field which do have the VOF value $f(\overrightarrow{a}) = I$ are part of the isosurface.

Like mentioned in section 3.1 in the underlying dataset VOF values are only defined at the cell center. This results in a discrete data set where values at arbitrary locations have to be interpolated from the cell center. To find the isosurface of a discrete data set the Marching Cubes (MC) algorithm as presented by Lorensen and Cline [LE87] can be used. For a given isovalue the algorithm approximates the isosurface by creating a defined set of triangles for every cell in the dataset. This is done by comparing the VOF value values at cell node to the isovalue. If the VOF value at that node exceeds the isovalue it is considered to lie outside the volume which is bounded by the isosurface. Every node with a value below the isovalue is considered to lie inside the bounded volume. Edges which do have one node inside and one outside of the volume must therefore have a point which lies directly on the isosurface. This information is stored in a binary vector for each cell. Figure 3.3 shows an example of a cell with the calculated binary vector. However, at the beginning the VOF value at each cell node has to be interpolated from the cell centered VOF value. This is explained in the next section.

Using this vector the algorithm loads a triangle out of a set of 256 different patterns which represent the approximated isosurface inside the cell. Due to symmetry the number of these patterns can be further reduced to 14. These can be seen in figure 3.3b. For each binary vector one of the patterns is chosen so that the vertices of the triangles fit on the edges which have points located on the isosurface. Similar to [SR18] the exact position of a point is then found using 3D linear interpolation between the bounding nodes of the edge

$$\overrightarrow{P_n} = (1.0 - w) \cdot \overrightarrow{P_1} + w \cdot \overrightarrow{P_2} \tag{3.6}$$

where $\overrightarrow{P_n}$ is the new point on the edge based on the triangle pattern, $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ are the nodes bounding the edge. And w is the weight for the interpolation which can be retrieved as

$$w = \frac{I - f_1}{f_2 - f_1} \tag{3.7}$$

where f_1 and f_2 are the VOF values at nodes $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ and I is the chosen isovalue. Figure 3.4 shows an example of the final triangle pattern inside a cell.

In this work only two-phase datasets are used. Yet, some of the patterns in figure 3.3b could also

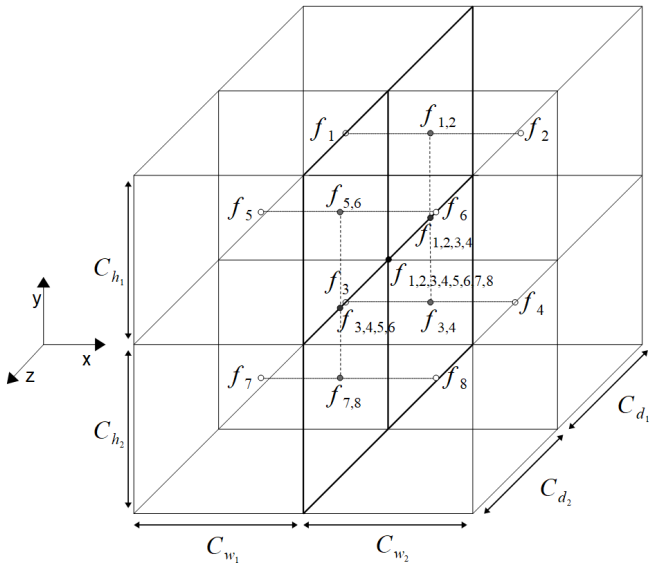


Figure 3.5: Interpolation scheme for Volume-Of-Fluid values at cell nodes. First values $f_{1,2}$, $f_{3,4}$, $f_{5,6}$, $f_{7,8}$ (gray points) are interpolated from the cell centered VOF values (white points). Then these are interpolated to yield the values $f_{1,2,3,4}$ and $f_{5,6,7,8}$ (dark gray points) which are then used to calculate the final point $f_{1,2,3,4,5,6,7,8}$ (black point)

depict multi-phase flows. This is the case for, e.g., pattern four since the cell is intersected by more than one interface. However, for multi-phase flows the dataset must contain multiple VOF values per cell. By providing the data in the way described in the next section such arbitrary cases can only be chosen if the grid has a low node resolution. In this context cases 1, 2, 5, 8, 9, 11 and 14 are considered to depict valid two-phase flow.
There are cases in VOF datasets for which the Marching Cubes algorithm cannot be used directly for two-phase flow to create a closed surface mesh. For this work the algorithm is therefore adapted to use local isovalues on cell edges. This is also explained in the next section.

3.2.2 Adaption to VoF-datasets

The MC algorithm is only executed for cells with a centered VOF value greater than zero and smaller than one since they contain parts of the interface. Depending on the isovalue and it's VOF value each cell node is located either inside or outside the fluid volume bounded by the isosurface. In order to apply the algorithm on the VOF dataset the first step is therefore to determine the VOF values at the cell nodes. In the case of rectilinear grids this can be done using trilinear interpolation of the neighboring cell centered VOF values. For uniform grids averaging the VOF values is sufficient since the width, height and depth of all cells are equal. Other grid types such as curvilinear grids

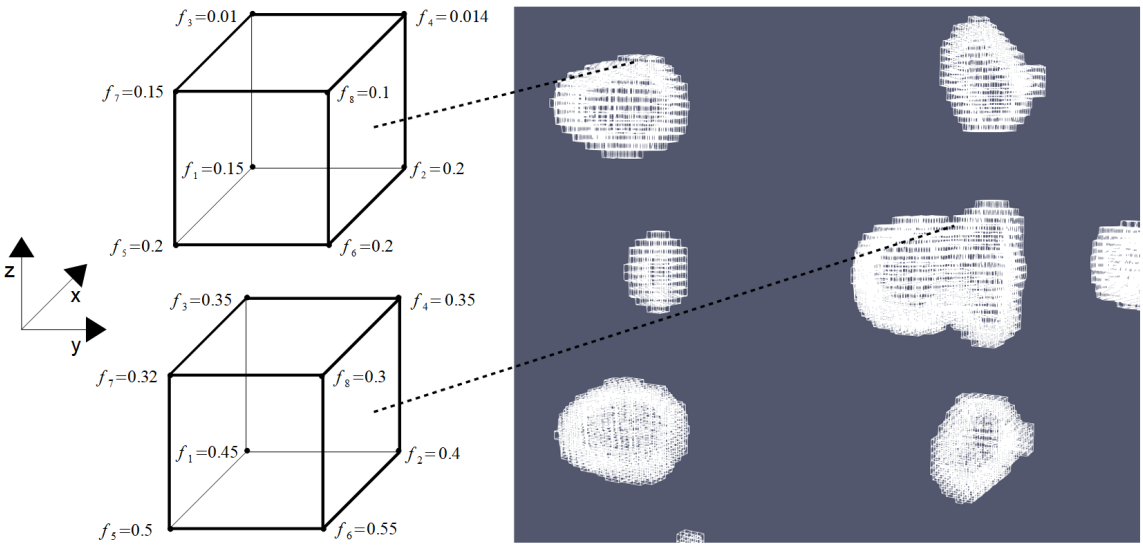


Figure 3.6: Example of cells with non-overlapping VOF value ranges. On the right side a section of a VOF dataset with cell contours can be seen. The biggest VOF value of the top cell is 0.2 while the smallest of the lower cell is 0.3. Choosing an isovalue which marks at least one node inside/outside of the volume in the top cell leads to all nodes of the bottom cell to lie inside the volume and vice versa. For one of this cells the MC algorithm does therefore not produce triangles. A global isovalue cannot be found in this case. This can also happen for cells that are part of the same cluster.

need a different approach by using space transformations and are not discussed here. Each node of a cell has eight adjacent cells with VOF values

$$f_1 = f(x, y, z) \tag{3.8}$$
$$f_2 = f(x+1, y, z) \tag{3.9}$$
$$f_3 = f(x, y-1, z) \tag{3.10}$$
$$f_4 = f(x+1, y-1, z) \tag{3.11}$$
$$f_5 = f(x, y, z+1) \tag{3.12}$$
$$f_6 = f(x+1, y, z+1) \tag{3.13}$$
$$f_7 = f(x, y-1, z+1) \tag{3.14}$$
$$f_8 = f(x+1, y-1, z+1) \tag{3.15}$$

assuming that the node is located at the origin of the cell with indices $(x+1, y, z+1)^T$.
Interpolating these values linearly along the x-direction leads to four interpolated values VOF values $f_{1,2}, f_{3,4}, f_{5,6}, f_{7,8}$

$$f_{1,2} = (1.0 - w_x) \cdot f_1 + w_x \cdot f_2 \tag{3.16}$$
$$f_{3,4} = (1.0 - w_x) \cdot f_3 + w_x \cdot f_4 \tag{3.17}$$
$$f_{5,6} = (1.0 - w_x) \cdot f_5 + w_x \cdot f_6 \tag{3.18}$$
$$f_{7,8} = (1.0 - w_x) \cdot f_6 + w_x \cdot f_8 \tag{3.19}$$

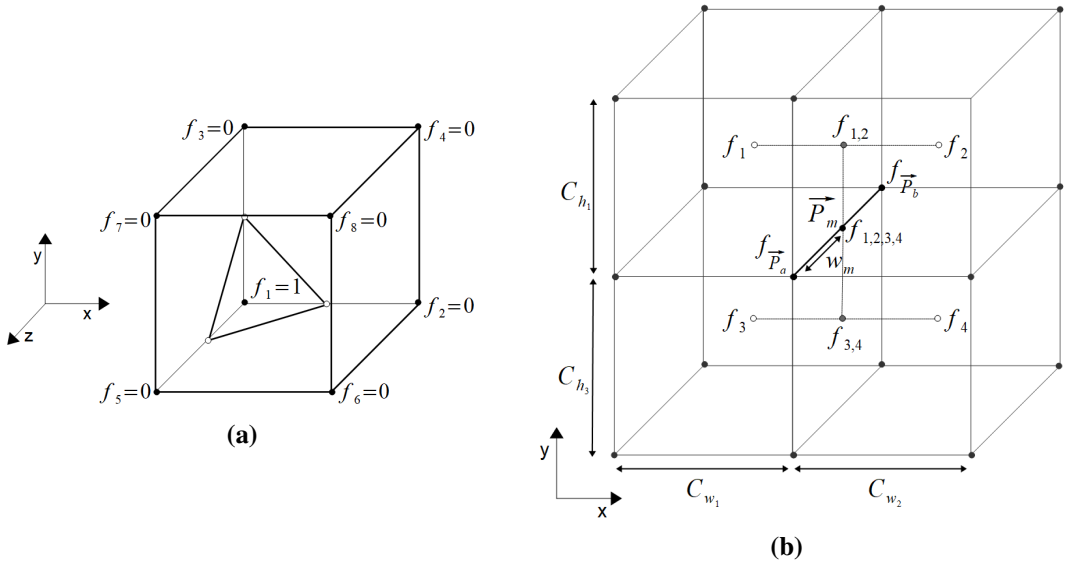


Figure 3.7: Example of local isovalue determination and edge correction:
(a) Cell nodes having an adjacent cell with centered VOF value of zero are assigned a value of zero. Others are set to one. Vertices of the MC triangle pattern are set to the middle of the respective edges using an isovalue of $I = 0.5$.
(b) Further correction of the MC output depicted at the right bottom edge. For cells sharing the edge (black line) the centered VOF values f_1, f_2, f_3 and f_4 are interpolated bilinear to create a new value $f_{1,2,3,4}$ at the middle of the edge. By calculating the weight w_m the point $\overrightarrow{P_m}$ can be shifted relative to the positions of nodes $\overrightarrow{P_a}$ and $\overrightarrow{P_b}$.

where w_x is the weight for the interpolation in x-direction defined as $w_x = \frac{\frac{C_{w_1}}{2}}{\frac{C_{w_1}+C_{w_2}}{2}}$. C_{w_1} is the width of the cell at $(x, y, z)^T$ and C_{w_2} is the width of the cell at $(x + 1, y, z)^T$.
Next the interpolation is done along the y-direction yielding two values $f_{1,2,3,4}$ and $f_{5,6,7,8}$

$$f_{1,2,3,4} = (1.0 - w_y) \cdot f_{1,2} + w_y \cdot f_{3,4} \tag{3.20}$$
$$f_{5,6,7,8} = (1.0 - w_y) \cdot f_{5,6} + w_y \cdot f_{7,8} \tag{3.21}$$

w_y is the weight for the interpolation in y-direction defined as $w_y = \frac{\frac{C_{h_1}}{2}}{\frac{C_{h_1}+C_{h_2}}{2}}$. C_{h_1} is the height of the cell at $(x, y, z)^T$ and C_{h_2} is the height of the cell at $(x, y - 1, z)^T$.
The last step is to interpolate along the z-direction which results in the final VOF value $f_{1,2,3,4,6,7,8}$ at the node

$$f_{1,2,3,4,5,6,7,8} = (1.0 - w_z) \cdot f_{1,2,3,4} + w_z \cdot f_{5,6,7,8} \tag{3.22}$$

with w_z being the weight for the interpolation in z-direction defined as $w_z = \frac{\frac{C_{d_1}}{2}}{\frac{C_{d_1}+C_{d_2}}{2}}$. C_{d_1} is the depth of the cell at $(x, y, z)^T$ and C_{d_2} is the depth of the cell at $(x, y, z + 1)^T$. See figure 3.5 for a depiction of the interpolation scheme.
The interpolation scheme does not consider the real underlying behavior of the VOF field since only linear interpolation is used. Yet, if the interpolation is done for a dataset with a very high node

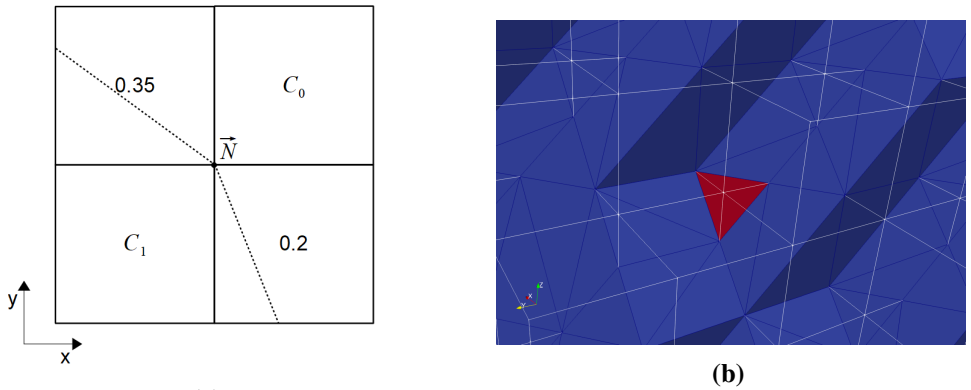
(a) **(b)** **(c)**

Figure 3.8: Result of the edge correction step on a sphere:
(a) Vertices of MC created triangle patterns are located at the middle of edges.
(b) Correction done locally on an edge with respect to VOF values of the bounding nodes $f_{\overrightarrow{P_a}}$ and $f_{\overrightarrow{P_b}}$
(c) Same dataset but visualized with PLIC by Youngs [You82].

resolution the values are close to the real VOF values at cell nodes with respect to a continuous VOF field. The isosurface patch inside the cell is then close to a small linearized segment of the real interface. However, the result of the MC algorithm is a coarse approximation of the isosurface. This already leads to inaccuracies in the mesh creation. Deviations of VOF values due to interpolation are therefore neglectable.
The MC algorithm creates triangle patterns only for cells that have edges with points located on the isosurface. For that one bounding node must be located outside and one inside the volume defined by the isosurface. To determine these edges the interpolated VOF values at the cell nodes are compared to the isovalue. It must therefore be chosen in such a way that for all interface cells (cells with a VOF value above zero or below one) at least one node is inside the volume while the rest is outside or vice versa. Otherwise the cell can not be considered and no triangle pattern is created. The mesh created by the MC algorithm is therefore not fully closed. Moreover, tests have shown that for some datasets a global isovalue to create patterns in all cells cannot be found. This is the case since the VOF value range of some cells at different positions in the grid do not overlap. This can happen in one of the following cases

1. One cell's maximal VOF value is lower than the minimal value of the other cell.
2. One cell's minimal VOF value is higher than the maximal value of the other cell.

See figure 3.6 for an example. In the tested datasets these cases did occur often for different pairs of cells.
To circumvent this problem and to extract a closed surface mesh the global isovalue is changed to a local approach at the edges of adjacent cells. The first step is to determine the edges which have points located on the isosurface. For that one node of the edge has to lie inside the fluid volume while the other one lies outside. If a node has at least one neighbor cell with a VOF value of zero then this node is marked to lie outside by assigning a value of zero. This can be done with respect to the following observations:



(a) **(b)**

Figure 3.9: Example of MC problem case and it's resolution:
(a) Node $\vec{N}$ has adjacent cells C_0 and C_1. The isosurface segments (dashed line) of the other interface cells must therefore connect at node $\vec{N}$.
(b) The red area of the surface shows the additional created triangle inside C_1 to close the mesh. Interface cells with a VOF value are depicted as white cubes.

1. The closed isosurface separates the 3D space into volumes that are completely filled with one of the phases (here fluid or gas). This is also the case for cells which contain parts of the interface.
2. Cells with a VOF value of zero must lie outside the fluid volume since they do not contain fluid.
3. Cells with a VOF value of one must lie inside the fluid volume since they do only contain fluid.

Nodes having an adjacent cell with a VOF value of zero must therefore lie outside the fluid volume or directly on the isosurface (see below).

Now, each node is marked with zero or one. By choosing a global isovalue of 0.5 the vertices of the created triangle patterns are located at the middle point $\vec{P_m}$ of these edges, see figure 3.7a. With this approach all cells that contain a part of the interface are considered by the MC algorithm. Most of the time this results in a closed surface mesh which could not be retrieved by using the MC and a global isovalue. A method to close remaining holes is explained below. However, setting the position of the created triangle vertex to the middle of the edge is not a good approximation of the underlying VOF field. The result can be improved by adding a small correction step per edge using adjacent cells to adjust the position of the triangle vertice. First, the interpolated VOF values $f_{\vec{P_a}}$ and $f_{\vec{P_b}}$ of the bounding nodes $\vec{P_a}$ and $\vec{P_b}$ are determined. Next the VOF values f_1, f_2, f_3 and f_4 of the cells sharing this edge are interpolated bilinear onto it's middle point yielding a new value $f_{1,2,3,4}$

$$f_{1,2} = (1.0 - w_1) \cdot f_1 + w_1 \cdot f_2 \tag{3.23}$$
$$f_{3,4} = (1.0 - w_1) \cdot f_3 + w_1 \cdot f_4 \tag{3.24}$$
$$f_{1,2,3,4} = (1.0 - w_2) \cdot f_{1,2} + w_2 \cdot f_{3,4} \tag{3.25}$$

where w_1 is the weight in one dimension and w_2 in the orthogonal direction depending on the orientation of the edges. For an edge on the bottom right side of a cell this would be $w_1 = \frac{\frac{c_{w_1}}{2}}{\frac{c_{w_1}+c_{w_2}}{2}}$ and $w_2 = \frac{\frac{c_{h_1}}{2}}{\frac{c_{h_1}+c_{h_2}}{2}}$. See figure 3.7b for the scheme on this edge. The calculation at other edges is straightforward. Again, in the case of an uniform grid $f_{1,2,3,4}$ can be determined by averaging. If the edge lies on a boundary of the dataset only a linear interpolation in one of the dimensions is done. The new position of $\overrightarrow{P_m}$ is then set by interpolating between $f_{\overrightarrow{P_a}}$ and $f_{\overrightarrow{P_b}}$ relative to their position

$$w_m = \frac{f_{1,2,3,4} - f_{\overrightarrow{P_a}}}{f_{\overrightarrow{P_b}} - f_{\overrightarrow{P_a}}} \tag{3.26}$$

$$\overrightarrow{P_m} = (1.0 - w_m) \cdot \overrightarrow{P_a} + w_m \cdot \overrightarrow{P_b} \tag{3.27}$$

For a resulting weight of $w_m < 0$ or $w_m > 1$ $\overrightarrow{P_m}$ stays in the middle of the edge. This can happen if the calculated value is lower than $f_{\overrightarrow{P_a}}$ or higher than $f_{\overrightarrow{P_b}}$ due to numerical inaccuracies in the dataset. Additionally, the linearity of the interpolation cannot always depict the underlying flow of the VOF field precisely. However, for the creation of a basic surface mesh this is sufficient since the isosurface approximation created by the MC algorithm introduces inaccuracies by nature. Further, the goal is to improve the result by adapting this mesh iterative to fit the underlying VOF field. In figure 3.8 it can be seen that the correction step produces a preciser isosurface approximation.

There are cases in which a node $\overrightarrow{N}$ has adjacent cells C_0 and C_1 with VOF values of zero and one respectively. Since these cells normally do not contain segments of the isosurface and are not considered by the MC algorithm further corrections are necessary. Otherwise holes in the surface mesh are created because $\overrightarrow{N}$ is considered to lie outside (has neighbor cell with VOF value of zero) the fluid volume while the other nodes of C_1 are inside. This case indicates that the node itself is located on the isosurface if other cells which share this node contain segments of the interface. See figure 3.9a for a 2D example case. It is questionable if such cases can really occur or are due to numerical inaccuracies in the datasets. E.g., in the case of the data created by Free Surface 3D VOF values very close to zero are set to zero (0.000001) and VOF values very close to one to one (0.999999). This threshold could vary depending on the program used to create the datasets and lead to more closely located interface cells.

Moving the vertices of edges sharing $\overrightarrow{N}$ onto the node in order to close the mesh can lead to further inaccuracies. The volume in adjacent cells is altered and less vertices are present to adapt the mesh. To solve this problem C_1 is set to be an interface cell. The MC algorithm can therefore create a triangle pattern to close the mesh, see figure 3.9b. Yet, since the cell has a VOF value of one further operations should move the created patch out of the cell to correct the volume.

Figure 3.11 shows the final output of the adapted MC algorithm and PLIC for two different datasets. It can be seen that the presented adaption is already approximating the results achieved with PLIC. Further, in figure 3.10a a 2D cell arrangement similar to the one found in Sadlo et al. [KSM+13] can be seen. It leads to artifacts (stacked planes) when PLIC is used. In this case the adapted MC algorithm closes the mesh for all connected cells. This behavior can also be observed in the 3D case, see in figure 3.10d.

For some cell arrangements no triangles can be produced. Such a case is depicted in figure 3.10b. All cell nodes are marked as being outside the volume. The VOF field can thus behave arbitrarily inside the cell and this cannot be depicted by the MC algorithm. It would demand the introduction

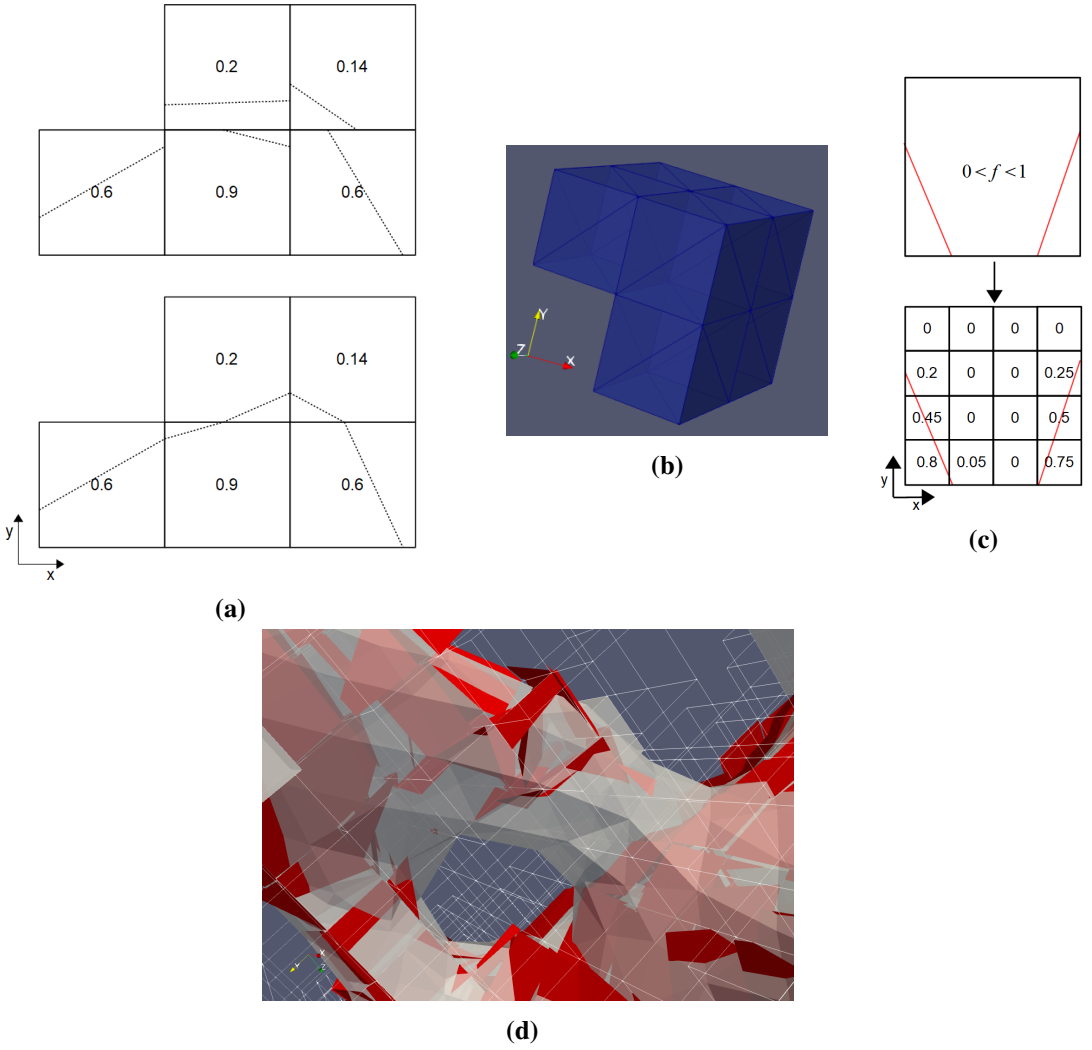


Figure 3.10: Example of PLIC artifacts resolved by the MC algorithm and problematic arrangement of cells:
(a) In the upper image a 2D cell arrangement can be seen that leads to stacked planes (dashed lines) when PLIC is used [KSM+13]. In the bottom image it is shown that the adapted MC algorithm can treat such cases right by providing a closed surface mesh. The dashed lines are created by stepping through the described algorithm (node inside/outside selection and pattern creation).
(b) Cell arrangement which leads to complex or arbitrary behavior inside the cells since all nodes are marked as being outside the volume. Here, a higher node resolution can resolve such problems, see also figure 3.10c.
(c) Example of a cell intersected by multiple interfaces. Again, a higher cell resolution must be used to treat such cases correctly.
(d) Example of mesh connectivity introduced by the MC algorithm. The red patches depict the planes created using PLIC. A connection of the resulting surface patches cannot be seen. With the adapted MC algorithm the mesh (transparent surface) is connected over adjacent cells. Interface cells are depicted as white cubes.

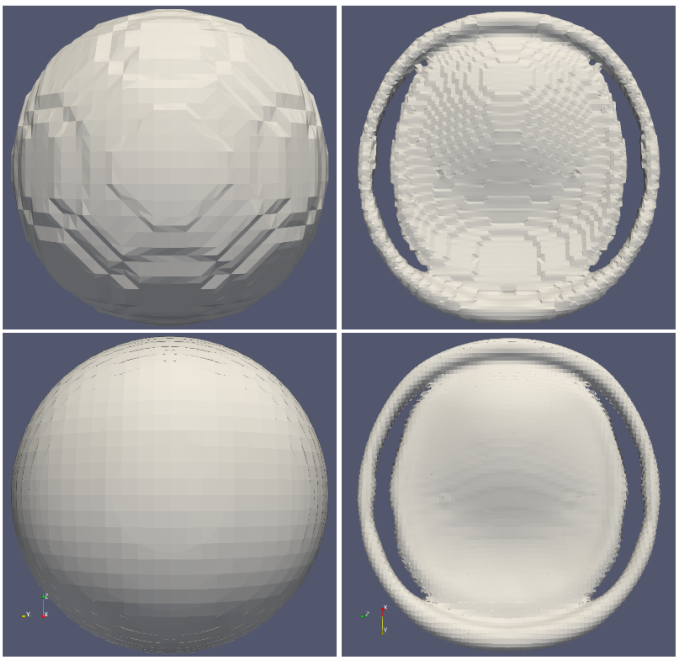


Figure 3.11: Output of the adapted MC algorithm compared to PLIC:
In the upper row results for the adapted Marching Cubes on different datasets can be seen. In the lower row the same dataset is shown using PLIC. The created mesh already approaches the visualization created with PLIC.

of closed concave triangle patterns inside the cell if such arrangements are not due to numerical inaccuracies. This issue can be solved by providing a higher node resolution so that complex VOF field behavior is distributed over smaller cells. If multiple interfaces should intersect a single cell a higher node resolution is needed as well. See figure 3.10c for an example and the solution using a higher node resolution.
It is still necessary to adapt the mesh in order to correct the enclosed volume inside the cell and reduce mean curvature. In the next sections the necessary operations are explained in detail.

3.3 Gradient calculation

For volume correction (see section 4.1) the vertices of the surface mesh must be movement according to some constraints. Yet, this should be done along the normalized VOF field gradient since it contains information about the shape of the underlying VOF field. This is not the case for the normals of the surface mesh and hence no adaption to the real interface can be done. Gradients are calculated at cell nodes and then interpolated for each component to the vertex location $\overrightarrow{P}$ within the cell. Generally, the gradient of a 3D scalar function $f(x,y,z)$ with $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ can be defined as

$$\nabla f(x,y,z) = \left(\frac{df}{dx}(x,y,z), \frac{df}{dy}(x,y,z), \frac{df}{dz}(x,y,z)\right)^T \tag{3.28}$$

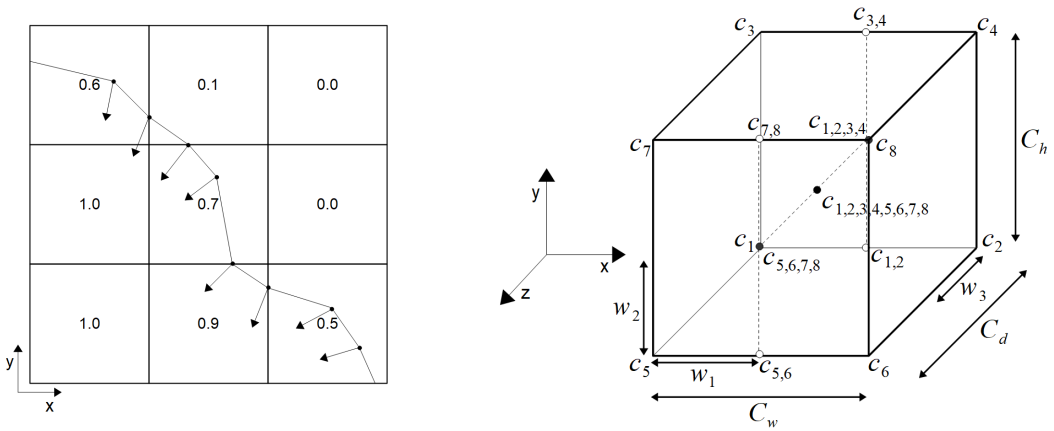


Figure 3.12: Basic gradient interpolation scheme and 2D example of the gradient direction:
(a) For each vertex on the surface mesh the VOF gradient points inside the fluid volume (directions are approximated). Cells are depicted as squares with the VOF value stored at the center.
(b) For each gradient component at the nodes $\overrightarrow{P_1}$ to $\overrightarrow{P_8}$ interpolation of the component c (c_1 to c_8 at the nodes respectively) is done along the x-direction yielding $c_{1,2}$, $c_{3,4}$, $c_{5,6}$ and $c_{7,8}$ (white points). Then along the y-direction yielding $c_{1,2,3,4}$ and $c_{5,6,7,8}$ (gray points). After that the final value $c_{1,2,3,4,5,6,7,8}$ (black point) is interpolated along the z-direction.

where the component $\frac{df}{dk}$ is the partial derivative of f in direction k. Here, the scalar function under consideration is the VOF field where x, y and z are cell indices. f is the VOF value stored at the cell center.

The interpolation of the gradient components from cell nodes is done using trilinear interpolation inside the cell similar to the scheme in section 3.2.2. First, the component is interpolated linearly along the x-direction

$$c_{1,2} = (1.0 - w_1) \cdot c_1 + w_1 \cdot c_2 \tag{3.29}$$
$$c_{3,4} = (1.0 - w_1) \cdot c_3 + w_1 \cdot c_4 \tag{3.30}$$
$$c_{5,6} = (1.0 - w_1) \cdot c_5 + w_1 \cdot c_6 \tag{3.31}$$
$$c_{7,8} = (1.0 - w_1) \cdot c_7 + w_1 \cdot c_8 \tag{3.32}$$

where c is the gradient component (x, y or z) at the cell nodes that is to be interpolated and $w_1$is the weight of the interpolation in x-direction. It relates to the local x-position $\overrightarrow{P}_x$ of the vertex inside the cell, hence $w_1 = \frac{\overrightarrow{P}_x - C_x}{C_w}$ where C_x is the cell's origin x-position and C_w is the width of the cell. This first interpolation step yields values $c_{1,2}, c_{3,4}, c_{5,6}, c_{7,8}$ on the edges parallel to the x-axis. The procedure is then repeated for the y-direction

$$c_{1,2,3,4} = (1.0 - w_2) \cdot c_{1,2} + w_2 \cdot c_{3,4} \tag{3.33}$$
$$c_{5,6,7,8} = (1.0 - w_2) \cdot c_{5,6} + w_2 \cdot c_{7,8} \tag{3.34}$$

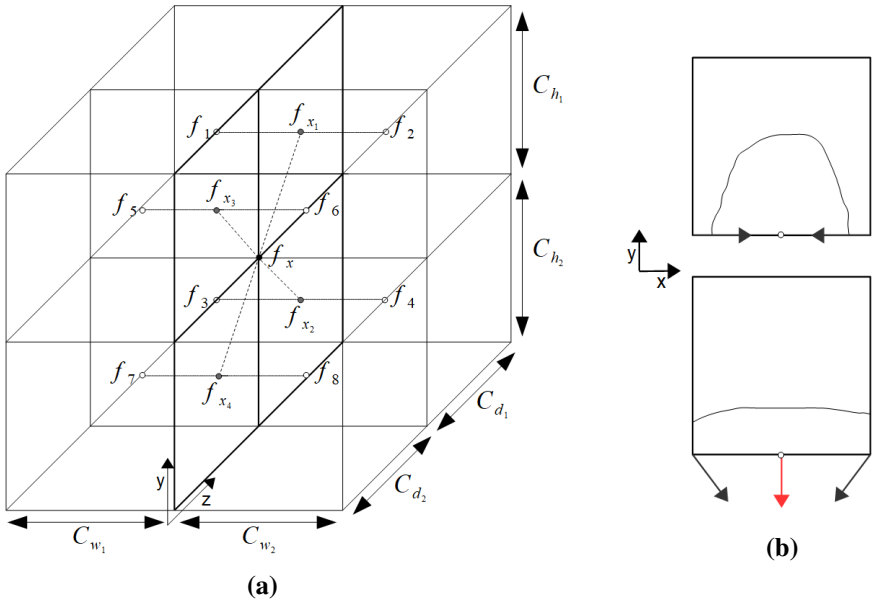
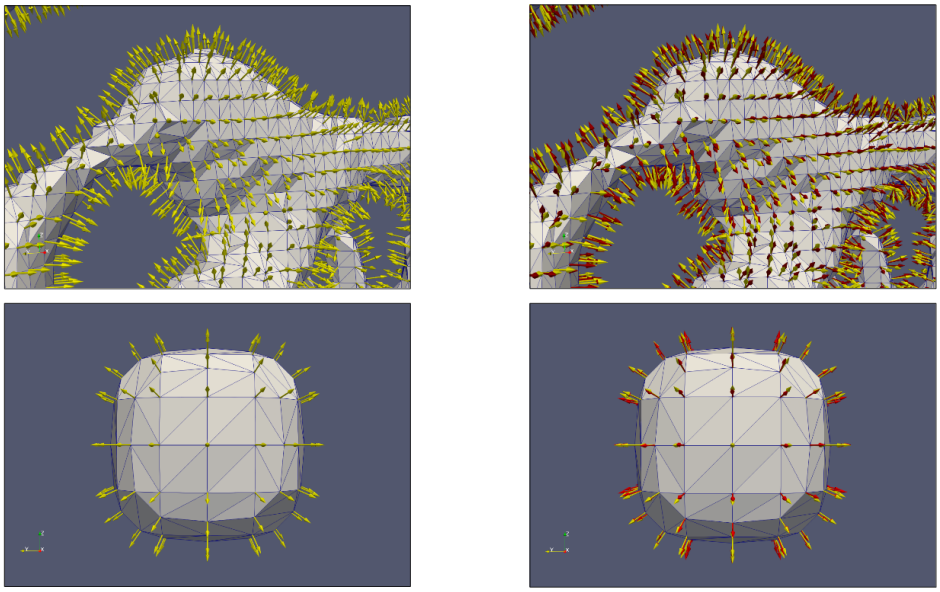


Figure 3.13: Example of the calculation of the partial derivative in x-direction and opposite directed gradients:
(a) For each adjacent cell the partial derivatives in x-direction are calculated at the shared xz-plane (f_{x_1} to f_{x_4}, gray points) using the cell based VOF values (white points). These are then interpolated to yield the final partial derivative in x-direction (f_x, black point) at the node.
(b) Top: low cell resolution is leading to a complex VOF field behavior in the cells and opposite directed gradients (gray arrows) at nodes. During interpolation the x or y-components could cancel each other out.
Bottom: higher cell resolution leads to smaller cells and a linear approximated interface. Complex VOF field behavior inside the cell cannot occur and interpolation leads to a valid gradient (red arrow) at the white point.

where $w_2$is the weight of the interpolation in y-direction. It relates to the local y-position $\overrightarrow{P}_y$ of the vertex inside the cell, hence $w_2 = \frac{\overrightarrow{P}_y - C_y}{C_h}$ where C_y is the cell's origin y-position and C_h is the height of the cell. The values $c_{1,2,3,4}$ and $c_{5,6,7,8}$ are located on the front and back planes of the cell. The last interpolation step in the z-direction yields the final component value $c_{1,2,3,4,5,6,7,8}$ at the vertex location

$$c_{1,2,3,4,5,6,7,8} = (1.0 - w_3) \cdot c_{1,2,3,4} + w_3 \cdot c_{5,6,7,8} \tag{3.35}$$

where w_3 is the weight of the interpolation in z-direction. It relates to the local z-position $\overrightarrow{P}_z$ of the vertex inside the cell, hence $w_3 = \frac{\overrightarrow{P}_z - C_z}{C_d}$ where C_z is the cell's origin z-position and C_d is the depth of the cell. The scheme can be seen in more detail in figure 3.12b. After interpolation of each component normalization of the combined vector yields the final gradient. It points in the direction of the highest value change. Therefore, it is pointing inside the fluid volume since the cells there have a higher VOF value, see figure 3.12a for a two-dimensional example.
To calculate the gradients at the cell nodes the approach presented by Sadlo et al. [KSM+13] is used. Every partial derivative is based on the interpolation of four plane based partial derivates



(a) **(b)**

Figure 3.14: Visualization of VOF field gradients at surface mesh vertices:
(a) Normalized VOF field gradients are depicted using yellow color.
(b) A direct comparison between normalized mesh normals (red - averaged from triangle normals) and normalized VOF gradients can be seen. The gradient direction was reversed for visualization purposes.

which are calculated using finite differences between the VOF values of adjacent cells. For the x-direction, e.g., these four partial derivatives $f_{x_1}, f_{x_2}, f_{x_3}, f_{x_4}$ are located at the center of the shared xz-planes. For the node at the origin of a cell with indices $(x+1, y, z+1)$ they can be defined using expressions 3.8 to 3.15 as

$$f_{x_1} = \frac{f_2 - f_1}{\frac{C_{w_2}}{2} + \frac{C_{w_1}}{2}} \tag{3.36}$$

$$f_{x_2} = \frac{f_4 - f_3}{\frac{C_{w_2}}{2} + \frac{C_{w_1}}{2}} \tag{3.37}$$

$$f_{x_3} = \frac{f_6 - f_5}{\frac{C_{w_2}}{2} + \frac{C_{w_1}}{2}} \tag{3.38}$$

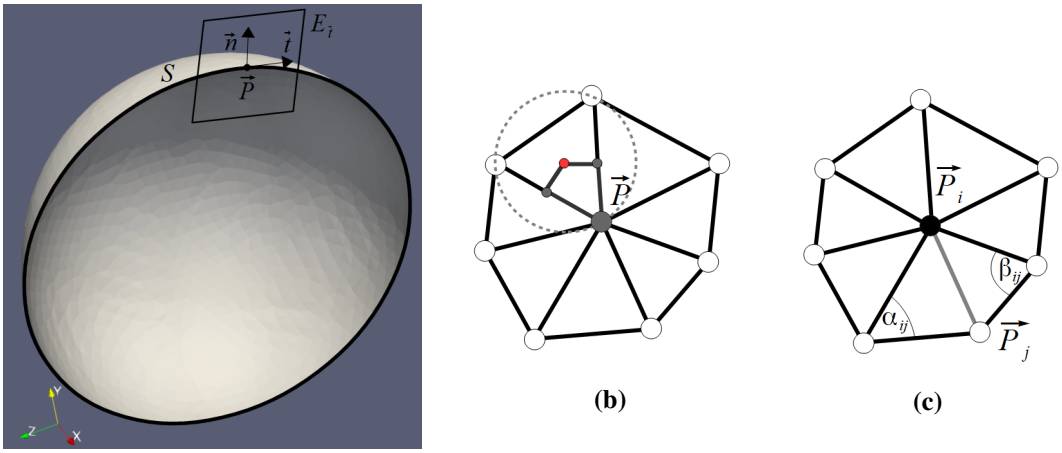
$$f_{x_4} = \frac{f_8 - f_7}{\frac{C_{w_2}}{2} + \frac{C_{w_1}}{2}} \tag{3.39}$$

where C_{w_1} relates to the width of the cell with indices (x, y, z) and C_{w_2} relates to the width of the cell with indices $(x+1, y, z)$. The calculation of the final x-derivative is then achieved using bilinear interpolation

$$f_{x_{1,2}} = (1.0 - w_1) \cdot f_{x_1} + w_1 \cdot f_{x_2} \tag{3.40}$$

$$f_{x_{3,4}} = (1.0 - w_1) \cdot f_{x_3} + w_1 \cdot f_{x_4} \tag{3.41}$$

$$f_x = (1.0 - w_2) \cdot f_{x_{1,2}} + w_2 \cdot f_{x_{3,4}} \tag{3.42}$$



(a) **(b)** **(c)**

Figure 3.15: Example of Voronoi area and tangent plane:
(a) Section of the tangent plane $E_{\vec{t}}$ (black square) depicted on a half sphere. The curve (black line) lying inside this plane and on the surface S can be seen.
(b) One-ring-neighborhood around $\vec{P}$ and Voronoi area (gray) for one triangle. It is created by using the circumcenter (red point) and the middle points of the edges sharing $\vec{P}$.
(c) Example of edge $\overrightarrow{P_iP_j}$ (gray) and angles α_{ij} and β_{ij}.

where $w_1 = \frac{\frac{c_{h_1}}{2}}{\frac{c_{h_1}}{2}+\frac{c_{h_2}}{2}}$, $w_2 = \frac{\frac{c_{d_1}}{2}}{\frac{c_{d_1}}{2}+\frac{c_{d_2}}{2}}$ and $f_x = \frac{df}{dx}(x,y,z)$. See figure 3.13a for a depiction of the scheme in x-direction. The other directions are treated analogously (xy-plane for the z-direction, xz-plane for the y-direction). Normalization of the combined vector yields the final gradient. To allow the calculation of the gradient at the boundary of the dataset artificial cells can be created by mirroring cells first in x then in y and lastly in z-direction.
For opposite oriented gradients it is possible that components cancel each other out during interpolation. However, due to the nature of the datasets these cases can only occur if the node resolution is very low. This introduces inaccuracies and the interface depicted by flow of the VOF field inside cells could be have an arbitrary shape. However, a higher node resolution of the datasets leads to an approximation of the interface with small planar surface patches in the cell, see figure 3.13b. Therefore, opposite directed gradients do not occur in the datasets under normal circumstances.
In figure 3.14a the calculated and normalized VOF gradients at mesh vertices are visualized. In figure 3.14b it can be seen that the mesh normals differ from the gradients of the VOF field since they do not contain information about the flow of the interface. Verification was done using simple meshs and checking each gradient for correct direction. After that this process was repeated using meshs of higher complexity.
In the next section the basic idea to calculate the surface curvature at mesh vertices is explained.

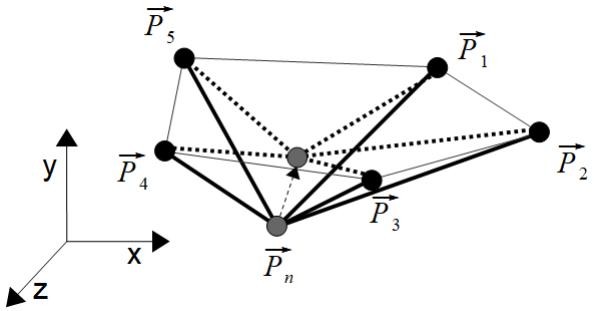


Figure 3.16: Visualization of a vertex shift using Laplacian smoothing. The mesh consists of six vertices ($N = 5$). The position of $\overrightarrow{P_n}$ is slightly lower which introduces a peak and hence area with higher local mean curvature (strong concave shape, κ_1 and κ_2 are high) on the mesh. After one step of Laplacian smoothing for $\overrightarrow{P_n}$ it is shifted slightly (dashed arrow) in direction of $\overrightarrow{P_1}$ to $\overrightarrow{P_5}$. This shrinkens the peak and reduces the mean curvature in this area (flatter mesh, $\kappa_H \rightarrow 0$). The vertex connections leading to $N = 5$ are marked as thick black lines.

3.4 Curvature calculation

One goal of the approach presented in this work is to minimize the curvature of the mesh that represents the interface. To be more precise the mesh's local mean curvature κ_H should be minimized hence $\kappa_H \rightarrow 0$. This results in a reduction of the principal curvature components of the surface (see formula 3.45) and therefore an overall flatter mesh since curvature can be generally described as the amount of local bend of a surface [MDSB01]. Thus, a minimal surface [CM04] with minimal surface area is approximated and a better adaption to the flow of the underlying VOF field as well as a visually more appealing result is achieved. Along with the method to reduce the mean curvature this is further discussed in section 3.5. Yet, it's effectivity must be verified in comparison to other operations (see section 4). This can be done by plotting and visual depiction of the mesh's mean curvature over time, see also figure 3.18.

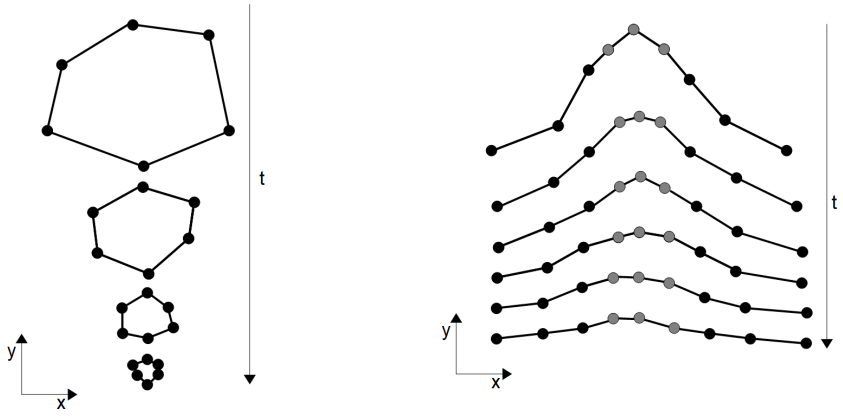
To calculate mean curvature values on a mesh the method presented by Meyer et al. [MDSB01] can be used. The mean curvature κ_H at a vertex $\overrightarrow{P}$ on the surface S can be defined [MDSB01] as

$$\kappa_H = \frac{1}{2\pi} \int_0^{2\pi} \kappa_N(\theta) d\theta \tag{3.43}$$

where $\kappa_N(\theta)$ is the normal curvature. For the tangent plane $E_{\overrightarrow{t}}$ defined by a surface normal $\overrightarrow{n}$ at $\overrightarrow{P}$ and the tangent $\overrightarrow{t_\theta}$ at angle θ the normal curvature expresses the curvature of a curve [1] lying on S and in $E_{\overrightarrow{t}}$, see figure 3.15a. κ_1 and κ_2 are the principal curvatures with tangents $\overrightarrow{t_{\theta_1}}$ and $\overrightarrow{t_{\theta_2}}$. The curve lying in $E_{\overrightarrow{t_{\theta_1}}}$ has the lowest and the one in $E_{\overrightarrow{t_{\theta_2}}}$ the highest normal curvature. $\kappa_N(\theta)$ can be expressed in terms of the principal curvatures κ_1 and κ_1 using Euler's Theorem [Ale78] as

$$\kappa_N(\theta) = \kappa_1 \cos(\theta)^2 + \kappa_2 \sin(\theta)^2 \tag{3.44}$$

[1] For a curve lying in a plane curvature is equal to the speed of directional change.



(a) **(b)**

Figure 3.17: Cases for problematic smoothing behavior:
(a) Positional information is distributed over a simple connected mesh (here in 2D). Laplacian smoothing tends to shift the mesh vertices to a single position. t depicts time.
(b) 2D extract of a mesh which shows a vertex cluster (gray points) at a peak. Over time it can be seen that the area around the cluster is getting flattened faster due to connected vertices which are not depicted here. Using a coarser mesh resolution can reduce this effect. t depicts time.

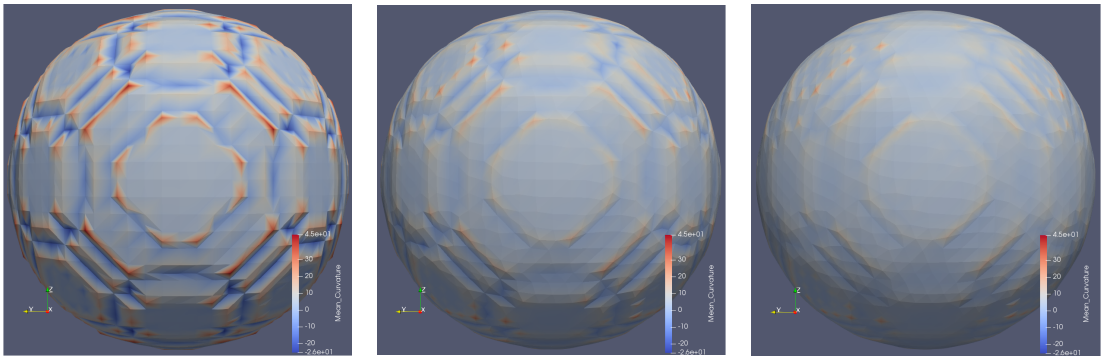
insertion into formula 3.43 with integration further yields

$$\kappa_H = \frac{(\kappa_1 + \kappa_2)}{2} \tag{3.45}$$

However, formula 3.43 needs to be discretized in order to be used at the surface mesh vertices. For that properties of mesh vertices are expressed in term of spatial averages over adjacent triangles (one-ring neighborhood).
For that Meyer et al. [MDSB01] redefine the integral boundary in terms of a discrete area over adjacent triangles with conformal parameters u, v and the normal curvature using a Laplacian $\overrightarrow{P}_{uu} + \overrightarrow{P}_{vv}$ for a vertex $\overrightarrow{P}$ at $(u,v)^T$. Using Gauss's Theorem they further redefine the Laplacian in terms of vertices and angles of the adjacent triangles. Averaging by the spatial volume area A_m (see below) yields the final value of the mean curvature κ_H at a mesh vertex $\overrightarrow{P_i}$ as

$$\kappa_H(\overrightarrow{P_i}) = \left\| \frac{1}{2A_m} \sum_{j \in N_1(i)} (\cot(\alpha_{ij}) + \cot(\beta_{ij}))(\overrightarrow{P_i} - \overrightarrow{P_j}) \right\| \cdot 0.5 \tag{3.46}$$

where $\overrightarrow{P_j}$ is the vertex lying on the same edge like $\overrightarrow{P_i}$. α_{ij} and β_{ij} are the opposite angles of the triangles sharing edge $\overrightarrow{P_iP_j}$. $N_1(i)$ is the number of triangles in the one-ring-neighborhood of $\overrightarrow{P_i}$. See figures 3.15b and 3.15c for a depiction. The resulting value is positive. Yet, it is sufficient to



(a) **(b)** **(c)**

Figure 3.18: Visualization of mean curvature reduction using Laplacian smoothing. Starting with a mesh created by the MC algorithm it can be seen that multiple iterations (from left to right) of Laplacian smoothing decrease the local mean curvature at vertices (peaks are reduced, blue/red color depict areas with high mean curvature). Here the smoothing operation has been scaled slightly to better depict the change of mean curvature over time.

investigate the evolution of the mesh's mean curvature (see section 4) over time.
The angle α between two vectors $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ can be determined as

$$\alpha = \arccos\left(\frac{\overrightarrow{P_1} \bullet \overrightarrow{P_2}}{\left\|\overrightarrow{P_1}\right\| \cdot \left\|\overrightarrow{P_2}\right\|}\right) \tag{3.47}$$

taken from the geometrical definition of the dot product.
A_m is the sum over partial triangle areas in the one-ring-neighborhood

$$A_m = \sum_{i=1}^{N} A_i \tag{3.48}$$

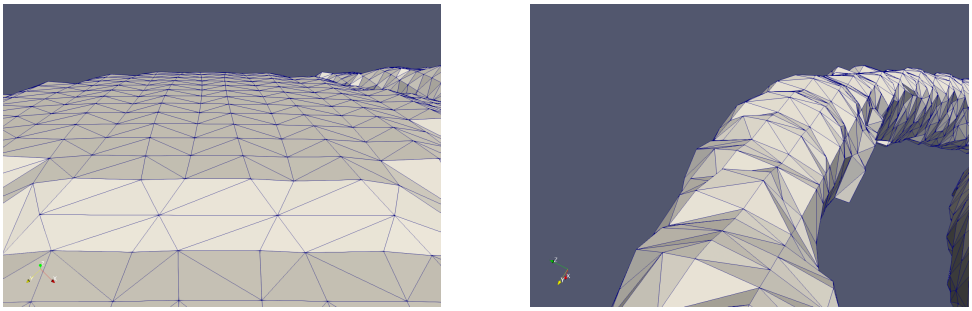
where A_i is the partial area of triangle i and N is the number of adjacent triangles of $\overrightarrow{P_i}$. For obtuse triangles (one angle is bigger than $90°$) A_i is the area of triangle i divided by two or four depending on the position of the obtuse angle [MDSB01]

$$A_i = \begin{cases} \frac{g \cdot h}{2} \cdot \frac{1}{2}, & \text{angle at } \overrightarrow{P_i} \text{ is obtuse} \\ \frac{g \cdot h}{2} \cdot \frac{1}{4}, & \text{else} \end{cases} \tag{3.49}$$

where g is the length of the triangle base and h is the height. If the triangle is not obtuse A_i is the Voronoi region [MDSB01] inside the triangle

$$A_i = \frac{1}{8} \sum_{j \in N_1(i)} (\cot(\alpha_{ij}) + \cot(\beta_{ij})) \left\|\overrightarrow{P_i} - \overrightarrow{P_j}\right\|^2 \tag{3.50}$$

That is the area defined by the circumcenter of the triangle connected to the middle points of the edges sharing $\overrightarrow{P_i}$ and $\overrightarrow{P_i}$ itself. See figure 3.15b for a depiction of the Voronoi area. Using the Voronoi region for each triangle leads to a better numerical estimate. Yet, for obtuse triangles the



(a) **(b)**

Figure 3.19: Example of the MC algorithm leading to a low resolution of triangles:
(a) Flat interface areas can be approximated with a low number of triangles.
(b) Complex VOF field behavior (curved surface) demands a higher local number of triangles. Otherwise peaks are introduced and the interface cannot be approximated precisely.

formula does not hold and therefore a simpler triangle area calculation is used instead [MDSB01]. Other approaches than the one presented here can also be used for different reasons. Yet, this approach has been tested against analytically known curvature values on different meshs [MDSB01]. It was shown that the error rate is close to the one attained by using more complicated schemes which are based on second-order finite differences. For a mesh with infinite resolution formula 3.46 should yield the correct analytical mean curvature values at every vertex [MDSB01]. Refer to [MDSB01] for more information on this topic.
To reduce the mean curvature a smoothing technique such as Laplacian smoothing can be used. This is explained in the next section.

3.5 Laplacian smoothing

The mesh within each cell should yield a precise approximation of the interface surface depicted by the VOF field. Yet, a mesh with high local curvature and hence positional fluctuations between vertices in cells is more likely to deviate from the real interface. This is the case since, e.g., phases have similar chemical properties [MR74] and hence the forces between molecules should lead to consistent surfaces.
Positional fluctuations are due to peaks and thus strong local bends on the surface mesh created by the MC algorithm, see, e.g., figure 3.8b. In the last section it was mentioned that curvature can be defined as the amount of local bend [MDSB01] of a surface. A local peak therefore leads to a high or low local mean curvature κ_H. This is the case since at these positions the mesh is strongly convex or concave shaped and both principal curvature directions are either high or low. However, a flat surface region results in principal curvatures and hence a mean curvature κ_H of zero[2]. The mesh must therefore be made flatter to remove local curvature fluctuation and achieve a better approximation of the interface defined by the VOF field.
To flatten the mesh Laplacian smoothing can be used. It is a commonly applied technique [OBB01a]

[2]By removing local bends the surface mesh area is reduced and a minimal surface ($\kappa_H \rightarrow 0$) [CM04] is approached.

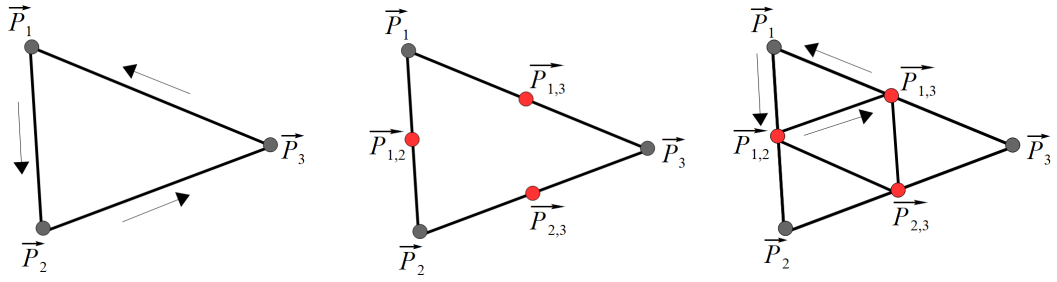


Figure 3.20: Refinement scheme for triangles:
(a) Surface mesh triangle with counter-clockwise connection order $\overrightarrow{P_1} \rightarrow \overrightarrow{P_2} \rightarrow \overrightarrow{P_3} \rightarrow \overrightarrow{P_1}$.
(b) Additional vertices (red - $\overrightarrow{P_{1,2}}$, $\overrightarrow{P_{1,3}}$ and $\overrightarrow{P_{2,3}}$) are added at the middle of each triangle edge.
(c) New vertices are connected counter-clockwise with old vertices to form a new triangle, e.g., $\overrightarrow{P_1} \rightarrow \overrightarrow{P_{1,2}} \rightarrow \overrightarrow{P_{1,3}} \rightarrow \overrightarrow{P_1}$.

and works by altering each vertice's position depending on the connected mesh vertices [VMM99], hence

$$\overrightarrow{P_n} = \frac{1}{N} \sum_{i=1}^{N} \overrightarrow{P_i} \tag{3.51}$$

where $\overrightarrow{P_n}$ is the new position of the vertex, $\overrightarrow{P_i}$ is a directly connected vertex and N is the total number of connected vertices. Using this formula vertices are shifted to each other which yields an overall flatter mesh [OBB01a]. See figure 3.16 for a depiction of the vertex movement. Here, every vertex has the same weight $\frac{1}{N}$ applied. Other weights like the inverse vertex distance can also be used. Yet, having the same weight for each connected vertex already yields good results [Tau99].
The effect of Laplacian smoothing is especially noticeable when, e.g., the mesh is visualized using different lighting models. Some of them like the well known Phong model [Pho75] depend on the interpolation of the normal within mesh triangles to a point at which the light intensity is calculated. A flatter mesh results in a slower change of the gradient's direction between adjacent triangles. This creates regions with similar lighting properties and therefore an overall more appealing visualization. On a closed mesh vertices are connected over multiple edges. Repeated steps of smoothing lead to attraction of far distant vertices since positional information is distributed over edges, see figure 3.17a. Therefore, vertices can be pulled to a single position and this shrinks the mesh's volume over time [OBB01a]. This effect is stronger on a mesh with low vertex resolution. Also, due to the isosurface approximation with simple triangle patterns the created surface mesh has many peaks. Vertices lying at such peaks have a smaller euclidean distance and form dense regions (clusters [OBB01a]). This is especially the case for a mesh with a high vertex resolution. Many smoothing steps are necessary until such clusters are flattened because distant positional information is distributed slowly, see figure 3.17b. To avoid the formation of clusters smoothing can be done on a coarse mesh (low number of vertices to depict the same surface) at first. This way positional information is distributed more rapidly between distant vertices at the beginning leading to faster smoothing. Yet, like mentioned above on a coarse mesh the shrinkage effect is stronger and this

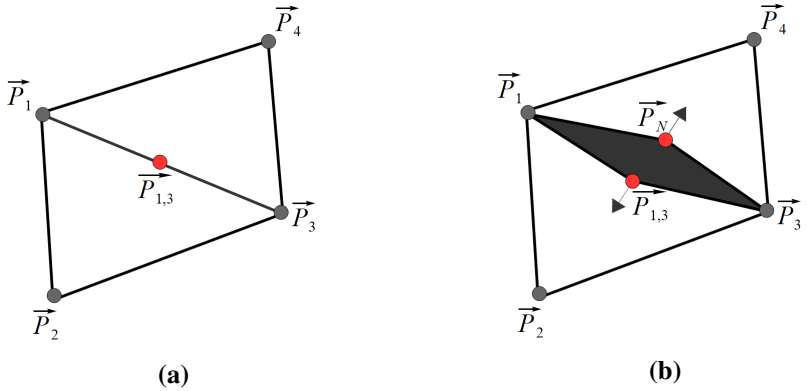
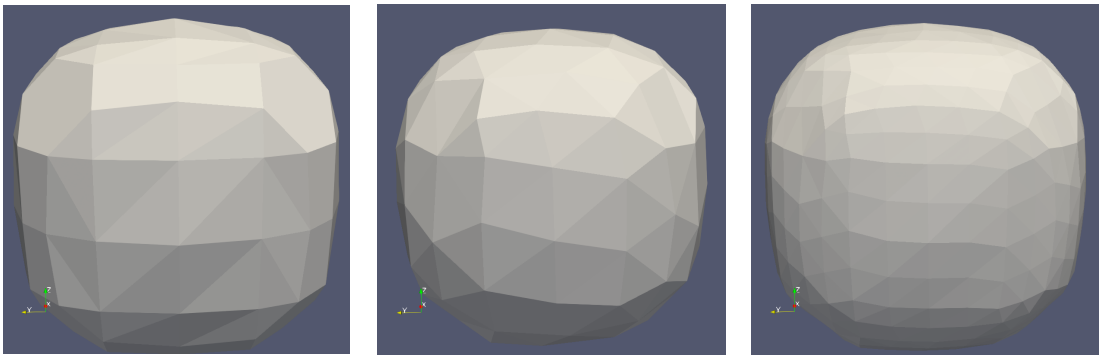


Figure 3.21: Example of mesh holes due to missing vertex tracking:
(a) Two triangles sharing an edge (gray line) with a newly added vertex $\overrightarrow{P_{1,2}}$ (red point) in the middle.
(b) Vertices have not been tracked. For an adjacent triangle a new vertex $\overrightarrow{P_N}$ was created on the same edge. A mesh operation (gray arrows) moves the vertices $\overrightarrow{P_{1,2}}$ and $\overrightarrow{P_N}$ differently. This leads to a hole in the mesh (gray area).

could counteract the interface approximation by altering the volume in cells. A trade-off between a coarse and fine mesh is therefore necessary, see section 4.3.
These properties can influence the results of other operations such as volume correction inside the cell if, e.g., the positional change created by the smoothing is pointing in a different direction. To solve this problem additional schemes are thus needed to combine the operations presented in this section. This is further discussed in section 4.
The effect of Laplacian smoothing is visualized over multiple time steps on a sphere ($51x51x51$ nodes) in figure 3.18. It can be seen that the local mean curvature at vertices is decreasing.
Like mentioned above the mesh resolution can impact the effect of smoothing. In the next section it is explained how the resolution of the mesh can be increased using triangle refinement.

3.6 Refinement

The basic surface mesh is created using the MC algorithm. Therefore, the number of initial triangles per cell to approximate the interface depicted by the VOF field is low. This is not a problem if the interface is a flat area, see figure 3.19a. However, mesh vertices on the basic surface mesh have a high distance and hence regions in which the interface bends or behaves more complicated cannot be approximated precisely. This can be seen in figure 3.19b and leads to deviations of the cell volumes and areas with a high mean curvature (peaks). Moreover, succeeding operations could introduce more peaks on the surface mesh. This can happen if, e.g., a cell contains only one vertex that can be shifted in order to correct the volume (see section 4.1). Also, the shrinkage effect of Laplacian smoothing is higher on a mesh with lower vertex resolution since distant positional information is distributed faster.
The impact of these problems can be reduced by having a mesh with higher triangles and thus vertex resolution. Distances between vertices are smaller and smoothing is done more locally without altering the coarse mesh structure, see figure 3.22. Further, in formula 3.56 it can be seen that the



(a) **(b)** **(c)**

Figure 3.22: Example of information distribution at different refinement steps:
(a) Original mesh after the MC algorithm.
(b) After multiple smoothing steps without refinement the coarse mesh structure does change significantly (mesh shrinkage). Information from distant connected vertices can be distributed faster which leads to a greater impact on the new vertices' positions.
(c) Two cycles of refinement. After multiple smoothing steps the coarse mesh structure has not change much. Yet, the mesh is overall flatter (lower mean curvature) and peaks have been reduced.

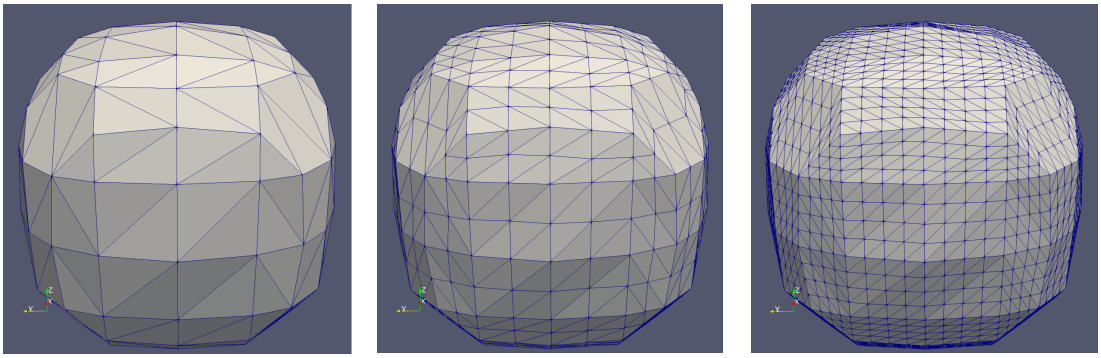
size of a triangle relates directly to the size of the corresponding volume element (tetrahedron) used for volume calculation. With smaller mesh triangles a more precise cell volume correction is therefore possible, see also section 4.3.
To introduce more triangles on the mesh a new vertex is created in the middle of each edge

$$\overrightarrow{P_{1,2}} = \frac{\overrightarrow{P_2} - \overrightarrow{P_1}}{2} \tag{3.52}$$

$$\overrightarrow{P_{1,3}} = \frac{\overrightarrow{P_3} - \overrightarrow{P_1}}{2} \tag{3.53}$$

$$\overrightarrow{P_{2,3}} = \frac{\overrightarrow{P_3} - \overrightarrow{P_2}}{2} \tag{3.54}$$

$\overrightarrow{P_{x,y}}$ is the new vertex which lies in the middle of the triangle edge defined by the vertices $\overrightarrow{P_x}$ and $\overrightarrow{P_y}$. The newly created vertices are connected with the original vertices in counter-clockwise direction to yield the new triangles. In figure 3.20 this can be seen in more detail and the result on a sphere ($10x10x10$ nodes) is shown in figure 3.23. This refinement scheme has been chosen because the created triangles are more similar in terms of area and vertex angles. Other methods tested lead to thin triangles and closely connected vertices.
The normals of new triangles point in the same direction as the normal of the original triangle if the vertices have already been connected in counter-clockwise direction before. If that is not the case the connection order or resulting normal vector of the new triangles must be flipped.
Adjacent triangles with same edges share the same created vertex $\overrightarrow{P_{x,y}}$. Hence, it is necessary to keep track of already created vertices and remove redundant ones. Otherwise operations which work on mesh vertices could lead to gaps within the surface mesh, see figure 3.21. Tracking can be done by checking adjacent cells for identical vertices, see section 3.8.



(a) **(b)** **(c)**

Figure 3.23: Example of refinement on a sphere:
(a) No refinement. Surface mesh is created by the Marching Cubes algorithm.
(b) Mesh after one refinement cycle.
(c) Mesh after the second refinement cycle.

Arbitrary many refinement cycles are possible but tests have shown that more than two are not practical. The obtained benefits do not justify the increasing computational costs of different operations, see section 4.3. Moreover, a higher resolution of the dataset helps to reduce the number of refinement cycles since cells and hence created mesh triangles are smaller. This already yields a more accurate surface mesh with respect to the real interface.
In the next section a method to calculate the mesh volume and hence VOF value inside cells is explained.

3.7 Volume calculation

One goal of the approach presented in this work is to keep the VOF value inside interface cells correct. In order to steer further corrections a method to measure the current VOF value of a cell is needed. The VOF value depicts how much volume of a cell is occupied by the fluid phase (see section 3.1). Therefore, to calculate the current VOF value the volume enclosed by the surface mesh and the part of the cell inside the fluid must be determined, see figure 3.24a for an example.
For volume calculation the approach by Zhang and Chen [ZC01] can be used. It is based on the sum of signed tetrahedra volumes. For each triangle that makes up the surface area of the enclosed volume a tetrahedron with one vertex located at the origin $\overrightarrow{O} = (0,0,0)^T$ is created. The signed volume V_s of this tetrahedron can then be calculated as [ZC01]

$$V_s = \frac{1}{6}\overrightarrow{OP} \bullet \overrightarrow{N_t} \tag{3.55}$$

where $\overrightarrow{OP}$ is a vector from the origin $\overrightarrow{O}$ to a vertex $\overrightarrow{P}$ of the triangle and $\overrightarrow{N_t}$ is the normal of the triangle. The sign of the volume is determined by using the dot product $\bullet$ in the formula. Hence, if the normal is pointing to the same triangle side as the vector $\overrightarrow{OP}$ the dot product is positive. This yields a positive volume V_s, see figure 3.25 for an example.

Assuming that the triangulated surface of the enclosed volume (see figure 3.24b for an example) is known the volume V can then be calculated [ZC01] using

$$V = \left|\sum_i V_{s_i}\right| = \left|\sum_i \frac{1}{6}(-x_{i,3}\cdot y_{i,2}\cdot z_{i,1} + x_{i,2}\cdot y_{i,3}\cdot z_{i,1} + x_{i,3}\cdot y_{i,1}\cdot z_{i,2} - x_{i,1}\cdot y_{i,3}\cdot z_{i,2} - x_{i,2}\cdot y_{i,1}\cdot z_{i,3} + x_{i,1}\cdot y_{i,2}\cdot z_{i,3})\right| \tag{3.56}$$

where V_{s_i} is the signed volume of the tetrahedron related to surface triangle i and $l_{i,k}$ is the l (x, y or z) component of vertex k for triangle i on the volume's surface. The winding order must be consistent between neighboring triangles to yield the correct result.
In order to apply formula 3.56 a triangulated surface that encloses the fluid volume like it can be seen in figure 3.24b is needed. To get this surface the intersection points of the mesh triangles and cell must be calculated. Then the resulting intersection pattern at the cell side planes and mesh triangles must be triangulated to yield a closed surface.
The first step is therefore to find all of the mesh's surface triangles which intersect the cell or are located inside of it. This can be done by testing the whole surface mesh or caching a subset of triangles for each cell as it is explained in section 3.8. Yet, the extracted mesh patch must overlap the cell entirely, see figure 3.26a. Otherwise holes are introduced and a correct triangulation at the cell sides is not possible. Depending on the orientation of the missing triangle's normal this could lead to a lower or higher overall volume, see figure 3.26b for an example case.
The next step is to find intersection points between the cell and patch triangles as it can be seen in figure 3.26a. The following cases are possible:

1. Intersection of mesh triangle edges and cell sides defined by bounded planes, see 3.7.
2. Intersection of cell edges and mesh triangles, see 3.7.

Each one of this cases leads to one or more intersection points on the cell's sides.
The plane at the side of the cell is defined by a normal vector $\overrightarrow{N} = (n_1, n_2, n_3)^T$ pointing out of the cell and one point $\overrightarrow{O}$ on the plane defining it's position in space. All points $\overrightarrow{P}$ which lie on the plane must then fulfill

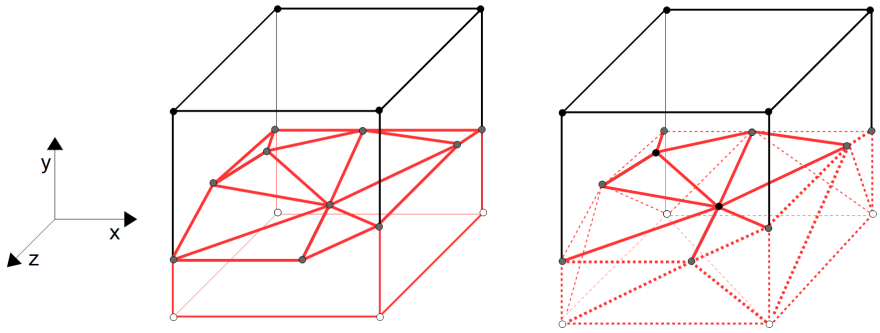
$$\overrightarrow{N} \bullet \overrightarrow{OP} = 0 \tag{3.57}$$

For the left side of the cell this would yield the normal $\overrightarrow{N} = (-1, 0, 0)^T$ and $\overrightarrow{O} = (C_x, C_y, C_z)^T$ being the cell's origin.
A triangle can be defined in terms of three vertices $\overrightarrow{P_1}$, $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$. Every point inside the triangle can be defined by using barycentric coordinates $\alpha_1, \alpha_2, \alpha_3$

$$\overrightarrow{P} = \alpha_1 \cdot \overrightarrow{P_1P_2} + \alpha_2 \cdot \overrightarrow{P_1P_3} + \alpha_3 \cdot \overrightarrow{P_2P_3} \tag{3.58}$$

with $\overrightarrow{P}$ lying on the triangle if the conditions $\sum_{i=1}^{3} \alpha_i = 1$ and $0 \leq \alpha_i \leq 1$ are met.
The edge of a cell lies on a line defined as

$$\overrightarrow{P}(t) = \overrightarrow{P_1} + t \cdot \overrightarrow{P_1P_2} \tag{3.59}$$



(a) (b)

Figure 3.24: Volume calculation example on cell sides:
(a) For each cell the volume enclosed by the surface mesh inside the cell (triangulated gray points) and the part of the cell lying inside the fluid (white points) must be calculated. The volume bounded by the surface with red lines must thus be considered for calculation.
(b) Cell nodes inside the volume (white points) and intersection points of the mesh triangles with the cell planes (gray points) have been triangulated (red dashed lines). All triangles depicted by red lines are needed in order to calculate the enclosed volume. The bottom side is triangulated by introducing two new triangles.

$\overrightarrow{P}$ is a point on the line and the parameter $t \in \mathbb{R}$ defines it's position. Yet, for the cell edge only $0 \leq t \leq 1$ is valid. $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ are the cell's nodes bounding the edge. An triangle edge can hence be defined in a similar way by setting $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ to be vertices of the triangle.

Case 1:

For each plane being coplanar with one of the cell's sides all intersection points with the patch triangles must be found. Intersections between cell planes and triangle edges are found by inserting formula 3.59 into 3.57. This yields

$$t = \frac{\overrightarrow{P_1 O} \bullet \overrightarrow{N}}{\overrightarrow{P_1 P_2} \bullet \overrightarrow{N}} \tag{3.60}$$

where $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$ are the triangle vertices for the specific edge. If $\overrightarrow{P_1 P_2} \bullet \overrightarrow{N} = 0$ the edge is parallel to the plane hence no intersection point can be found. For $t < 0$ and for $t > 1$ the intersection point is not located on the triangle edge. For $0 \leq t \leq 1$ the intersection point $\overrightarrow{K}$ on the plane can be calculated using t and formula 3.59. However, since the planes defined by formula 3.57 extend infinitely an additional two dimensional boundary check using the position of the respective nodes must be done to make sure that $\overrightarrow{K}$ really lies inside of the cell's side. E.g., if $\overrightarrow{P_1}$, $\overrightarrow{P_2}$, $\overrightarrow{P_3}$ and $\overrightarrow{P_4}$ are the nodes defining the cell's front side the following conditions have to be fulfilled

$$\overrightarrow{K}_x \leq \overrightarrow{P_2}_x \tag{3.61}$$

$$\overrightarrow{K}_x \geq \overrightarrow{P_1}_x \tag{3.62}$$

$$\overrightarrow{K}_y \leq \overrightarrow{P_1}_y \tag{3.63}$$

$$\overrightarrow{K}_y \geq \overrightarrow{P_3}_y \tag{3.64}$$

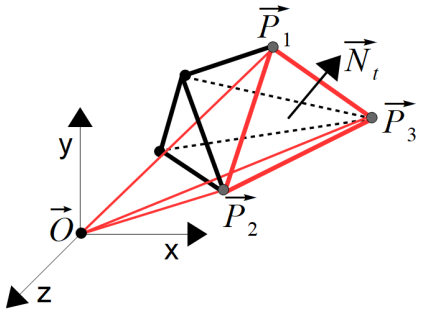


Figure 3.25: Example of a positive signed volume. For the triangle defined by the vertices $\overrightarrow{P_1} = (x_1, y_1, z_1)$, $\overrightarrow{P_2} = (x_2, y_2, z_2)$ and $\overrightarrow{P_3} = (x_3, y_3, z_3)^T$ a tetrahedron is created (red lines). Since the vector $\overrightarrow{OP_1}$ is pointing in the same direction as the normal $\overrightarrow{N_t}$ (positive dot product) the volume is positive. The winding-order is $\overrightarrow{P_1} \rightarrow \overrightarrow{P_2} \rightarrow \overrightarrow{P_3} \rightarrow \overrightarrow{P_1}$. Image is based on [ZC01].

where $\overrightarrow{j}_l$ depicts component l of the point $\overrightarrow{j}$. The scheme is also shown in figure 3.27a. The adaption to other cell sides is straightforward. This process must be repeated for each edge of the patch's triangles and cell sides. The intersection points can then be saved separately for each triangle and cell side.

Case 2:

Next, the intersection points between the cell's edges and triangles must be found. This is done by using formula 3.59 in combination with 3.57 while the plane and triangle are set to be coplanar. The plane and triangle normals are thus equal and defined as

$$\overrightarrow{N_t} = \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3} \tag{3.65}$$

$\overrightarrow{P_1}$, $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$ are the vertices of the triangle. After normalization this yields the final normal. Again, it is only checked for cases where $0 \leq t \leq 1$, hence the intersection point $\overrightarrow{K}$ must be located on the cell edge. If that is the case the location of the intersection point can be calculated using t and formula 3.59. Yet, an additional boundary check must be done to ensure that the point really lies within the triangle, see figure 3.27b. This is done using barycentric coordinates and yields

$$\overrightarrow{K} = \alpha_1 \cdot \overrightarrow{P_1P_2} + \alpha_2 \cdot \overrightarrow{P_1P_2} + \alpha_3 \cdot \overrightarrow{P_2P_3} \tag{3.66}$$

which leads to a system of linear equations

$$\left(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}, \overrightarrow{P_2P_3}\right) \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \overrightarrow{K} \tag{3.67}$$

The intersection point lies within the triangle if the conditions of formula 3.58 with respect to the calculated barycentric coordinates hold. However, if there is one i so that $\alpha_i = 0$ the point lies directly on a edge of the triangle. This point was already found by case one (see 3.7) since it is the intersection of a triangle edge with a cell side at it's border. Additionally, for a point $\overrightarrow{K} = (0,0,0)^T$ the trivial solution $\alpha_1 = 0, \alpha_2 = 0, \alpha_3 = 0$ is not valid and should be treated separately. Intersection

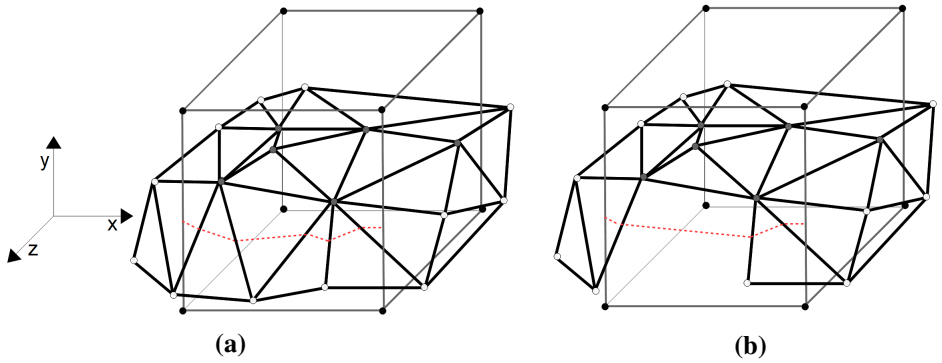


Figure 3.26: Example of extracted mesh for volume calculation:
(a) The mesh fully overlaps the cell and no holes are introduced. White points are outside and gray points inside the cell. The resulting intersection pattern on the cell's front side plane is depicted as red dashed line.
(b) The mesh does not fully overlap the cell. Holes are introduced at the front plane. A different intersection pattern can be seen which leads to an incorrect volume.

points can be saved for each cell edge.

To calculate the fluid volume using formula 3.56 all participating surface triangles must be determined. First, for each side of the cell the found intersection points and the nodes inside the fluid volume must be triangulated. The latter have been determined in section 3.2.2. For simple convex shaped intersection patterns points and nodes can be connected one after another. This is done by choosing one node inside the fluid volume as reference and connecting it with the closest point or node. Both are then connected with the next closest point or node. This yields the first triangle. After that the last connected point or node and the reference node are connected with the next closest point or node. This is repeated until all points and nodes are connected and the cell side is triangulated completely, see figure 3.28a.

Yet, not all segments of the surface mesh and hence intersection patterns have a convex shape. For the triangulation of such cases the ear clipping algorithm can be used which is based on a method proposed by G.H. Meisters [Mei75]. It works by cutting off triangles (ears) at the border of a polygon. Here, this polygon is defined at the cell sides using the intersection points and the cell nodes inside the fluid volume. Due to the connectivity of the mesh the boundary of the polygon is already given [3]. One point $\overrightarrow{P_1}$ on the polygon border along with it's two connected neighbors $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$ is chosen. It is then checked if $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$ can be connected without intersecting the boundary of the polygon. If that is the case $\overrightarrow{P_1}$ is removed along with the created triangle and the connection between $\overrightarrow{P_2}$ and $\overrightarrow{P_3}$ is added as new boundary segment, see figure 3.28b. This process is then repeated and the polygon is cut into multiple triangles. The normal of these triangles is equal to the one of the cell side plane. Intersection testing can be done by defining all border segments between connected points or nodes using formula 3.59 with insertion and additional boundary check. Other methods to triangulate such convex shaped polygons exist as well. Yet, the calculated volume must be the same in any case.

[3] Same intersection points on adjacent triangles or points and nodes lying on the same cell edge can be connected as a segment of the border.

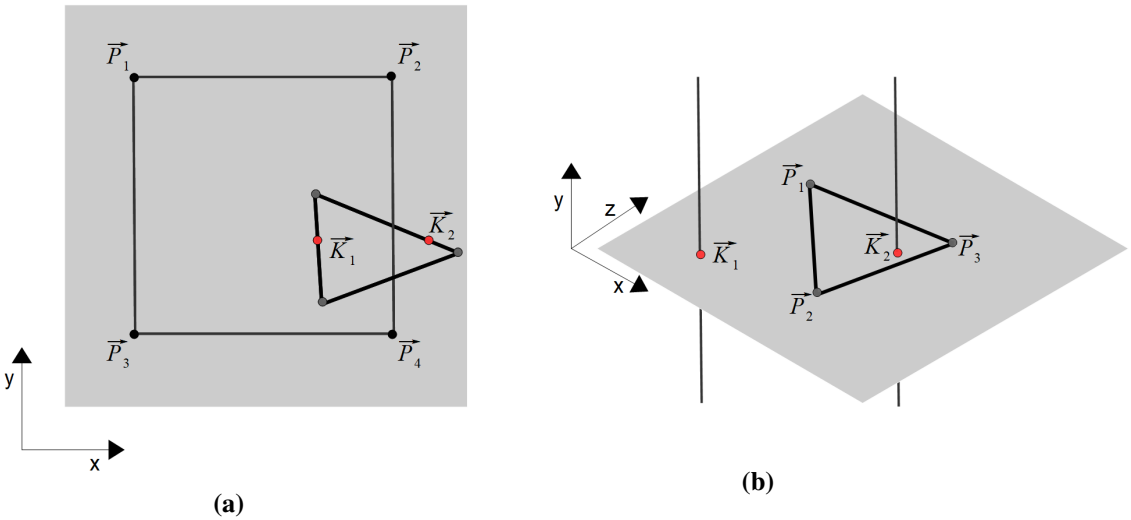


Figure 3.27: Example of boundary checking on triangles and cell sides:
(a) Two triangle edges intersect the infinite plane (gray area) at $\vec{K_1}$ and $\vec{K_2}$. Only $\vec{K_1}$ lies within the cell side define by $\vec{P_1}$, $\vec{P_2}$, $\vec{P_3}$ and $\vec{P_4}$.
(b) Two cell edges (gray lines) intersect the infinite plane at $\vec{K_1}$ and $\vec{K_2}$ but only $\vec{K_1}$ lies within the triangle defined by $\vec{P_1}$, $\vec{P_2}$ and $\vec{P_3}$.

There are cell sides which do not have intersection points with the surface mesh but nodes which are located in the fluid volume. These cases need to be treated additionally by introducing two new triangles, see figure 3.24b. The normal of these created triangles is set to be equal to the normal of the plane at the cell side.
Next, all triangles which do intersect the cell need to be cut at the side planes by adding new triangles. Since all possible intersection patterns are convex (see figure 3.28c) a simple scheme like depicted in figure 3.28a can be used. For that all intersection points of the triangle and it's vertices inside the cell must be known. The former have already been calculated in cases 1 or 2 and the latter can be found using a boundary check similar to expressions 3.61 to 3.64 using the cell's origin, width, height and depth as

$$C_x \leq \vec{T}_x \tag{3.68}$$
$$C_y \leq \vec{T}_y \tag{3.69}$$
$$C_z \leq \vec{T}_z \tag{3.70}$$
$$(C_x + C_w) \geq \vec{T}_x \tag{3.71}$$
$$(C_y + C_h) \geq \vec{T}_y \tag{3.72}$$
$$(C_z + C_d) \geq \vec{T}_y \tag{3.73}$$

C_x, C_y and C_z are the cell's x,y and z origin. $\vec{T}_l$ is the l component of triangle vertex $\vec{T}$ and C_w, C_h, C_d are the cell's width, height and depth respectively. The normal of each new triangle is equal to the one of the triangle before intersection.
At last triangles that are part of the mesh surface inside of the cell and do not intersecting the cell must be found. This is done by checking all triangles of the surface patch if all of their vertices

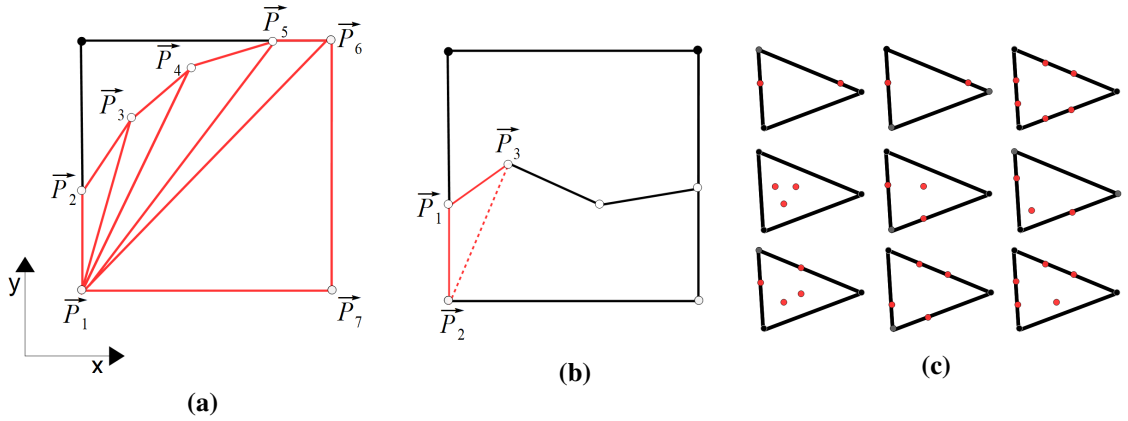
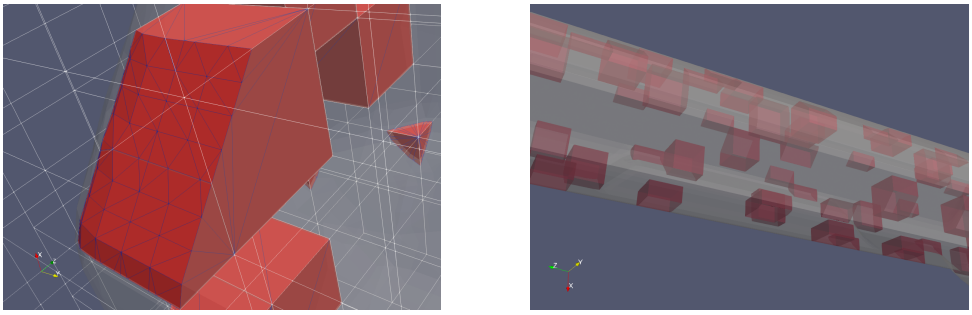


Figure 3.28: Triangulation examples of intersection patterns on the front side of a cell:
(a) After choosing a reference node $\vec{P_1}$ a convex intersection pattern is created by choosing the closest points or nodes iteratively. Here, $\vec{P_2}$ to $\vec{P_5}$ are calculated intersection points of triangle edges. $\vec{P_1}$, $\vec{P_6}$ and $\vec{P_7}$ are cell nodes that are located inside the fluid volume.
(b) First ear clipping step on a partially concave shaped intersection pattern. Starting with point $\vec{P_1}$ it is checked if the adjacent points or nodes $\vec{P_2}$ and $\vec{P_3}$ can be connected without intersecting the border of the polygon. The red dashed line is part of the new boundary of the polygon. The created triangle is enclosed by red lines and removed for the next step. Black points are located outside of the fluid volume.
(c) Possible triangle intersection patterns. Due to the geometry of the cell these patterns have a convex shape and can be triangulated using a simple scheme like it was used in **(a)**. The red dots depict intersections with cell sides or edges. Vertices lying inside the cell are depicted as gray points. They can be determined using a simple boundary check.

are inside the cell. Again, this can be done by using boundary checks with the cell's origin and dimensions with expressions 3.68 to 3.73 for each vertex of a triangle.
Having all needed triangles (triangles inside cell, triangulated cell sides and intersected triangles) the enclosed fluid volume can now be calculated using formula 3.56. However, a smaller cell with the same intersection patterns yields a lower enclosed volume. This is due to the fact that the calculated volume depends on the size of the triangles created by the MC algorithm and these are directly related to the cell size.
In figure 3.29 triangulated surface areas which are used for calculating the enclosed fluid volume can be seen. It directly relates to the VOF value f of the cell. If V_m is the measured enclosed fluid volume and V_v is the volume of the cell $(C_w \cdot C_h \cdot C_d)$ then f can be defined as

$$f = \frac{V_m}{V_v} \tag{3.74}$$

The current volume should be used to steer the correction of the VOF value inside cells. This is further explained in section 4.
In the next section implementation specific details with respect to volume calculation and other operations of this section are explained.



(a) **(b)**

Figure 3.29: Final triangulated intersection patterns of two different meshs:
(a) The resulting triangulation pattern at the surface of the fluid volume (red) inside a cell can be seen as blue lines. Interface cells are depicted as white cubes.
(b) Visualization example of cell fluid volumes (red). The interface is shown as transparent surface.

3.8 Implementation details

This section contains details about the implementation used for the operations in section 3.
The goal is to give a general idea of different design choices which speed up the computation. Further, it is explained which programming environment has been used and why certain decisions were made. Bottlenecks and problem cases are discussed as well.
However, a detailed explanation of the used libraries and programs as well as their internal implementation is not given. Instead it is referred to the original authors for further details.
The programs and libraries used here only represent a subset of possibilities to implement the operations mentioned in section 3. However, they have been chosen for different reasons as it is explained below. Yet, it is possible to choose other implementation techniques if they give an advantage in terms of simplicity or computational effort.

Rendering engine and graphical user interface

The operations defined in section 3 were implemented in the context of a Paraview[4] plugin. Paraview is an open source multi-platform application and widely used in the scientific community. It provides a lot of tools for loading, analyzing and visualizing big datasets. The rendering engine of Paraview is based on the Visualization Toolkit (VTK)[5] and the Graphical User Interface (GUI) on QT [6]. VTK is an open source multi-platform software system providing various algorithms for image processing and rendering which can be accessed in the context of a Paraview plugin. QT is a cross-platform framework and toolkit for creating GUIs.
The plugin is used as a data filter. Applied on a dataset Paraview automatically passes the dataset as VTK object to the plugin. After that all operations which are defined in section 3 are executed

[4] 5.6.0 RC3 64bit - `https://www.paraview.org/`
[5] As contained in Paraview 5.6.0 RC3 64bit - `https://vtk.org/`
[6] `https://www.qt.io/`

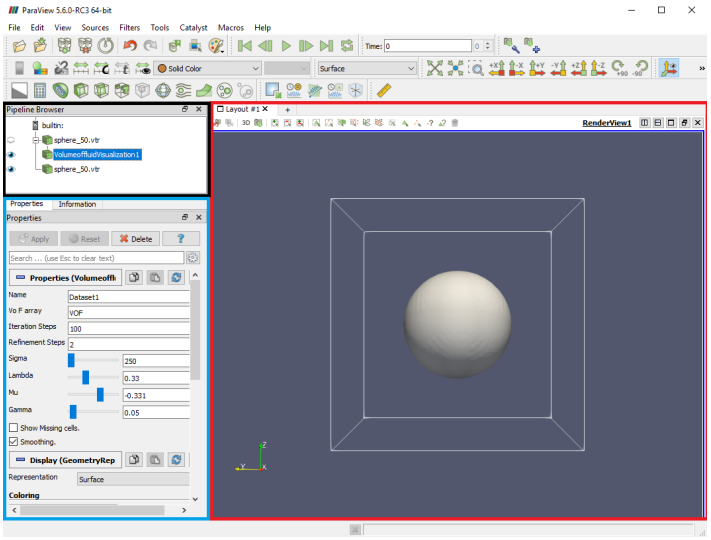


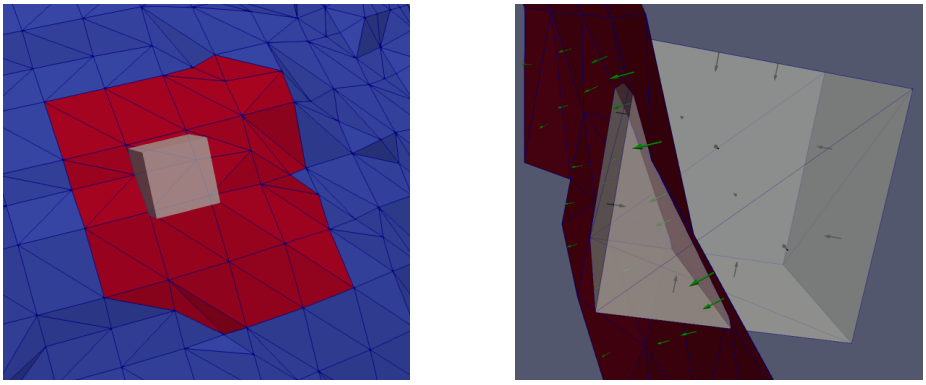
Figure 3.30: Overview of the Paraview GUI. The red rectangle depicts the render view. The blue rectangle depicts the properties of the chosen filter with available GUI parameters. The black rectangle depicts the pipeline browser view with a hierarchy tree consisting of the loaded datasets (top level) and applied filters.

depending on the parameters given in the plugin GUI. However, the order of execution depends on the schemes defined in section 4. The final output mesh as well as custom vertex and triangle related information are then passed back to Paraview and visualized in the render view. See figure 3.30 for a GUI overview.
Most libraries such as libigl[7] demand additional programming effort for loading, visualizing and processing data. Choosing a solution that already provides all needed functionality like Paraview saves a lot of development time.
The plugin as it was used at the end of section 4 can be found at `https://github.com/MaBa90/VOFVis`. In the next section it is explained how the different operations in section 3 are implemented within the plugin.

Operation implementation

Some of the operations defined in section 3 are implemented using external libraries or functions provided by VTK. This results in a slight computational overhead due to conversions which have to be done between different datastructures. For operations needed in multiple iterations data is therefore cached in simpler datastructures (`C++ vector`) and only updated if necessary. This the case for cell data such as the origin, dimension, node gradient and neighborhood information as well as vertex positions, connections and the containing cells. The latter is necessary for, e.g., interpolation of the gradient inside the cell. Additionally, a list with contained vertices is stored for each cell. To update the cell-vertex connections only the cell neighborhood is considered. Otherwise all cells have to be checked for each vertex which is computational expensive for a big dataset.

[7] `https://libigl.github.io/`



(a) **(b)**

Figure 3.31: Example of cached mesh surface patches and fluid volume with normals:
(a) For the cell depicted with yellow color the red surface patch has been cached. Only the vertex positions have to be updated in further iterations to calculate the volume.
(b) Normal direction for cell (yellow triangles, black arrows for normal) and surface patch (red triangles, green arrows for normal) along with the resulting fluid volume (left side behind the surface).

Since the MC algorithm is widely known using an already existing solution like the one provided by VTK saves development time. Therefore, the MC algorithm is based on the implementation provided by VTK[8]. To retrieve a closed surface mesh the algorithm is called for each cell separately after defining which nodes of the cell are lying inside or outside the fluid volume. See section 3.2.2 for more details. Like it is mentioned in section 3.6 vertices have to be tracked in order to maintain a closed surface mesh after refinement. Here, tracking is done by checking the neighboring cells for redundant vertices and reusing the corresponding vertex id (vector position). This is faster since for every cell only fifteen other cells with their containing vertices have to be checked. This number can be reduced even further if only interface cells are considered. Using the MC implementation from VTK requires a `vtkFloatArray`[9] datastructure containing node value information (inside or outside fluid volume) along with cell specific data such as the origin and dimensions. Creating this data for every cell introduces overhead. Yet, since the MC algorithm is only needed at the beginning to create a closed surface mesh it is neglectable.
After the triangle patterns for the cell are created the vertices can be shifted along the edges to yield a better approximated isosurface, see section 3.2 for further details.
The calculation of the cell volume is done using the Computational Geometry Algorithms Library (CGAL)[10] which provides different geometrical algorithms. To be more precise the `Clip` [11] function is used. Test have shown that the results were the same as planned in section 3.7 with respect to the resulting triangulation pattern and cell volumes. Additionally, choosing the `Clip` function from CGAL saves a lot of development time since the process of calculating the volume consists of many different steps and is therefore prone for errors. The `Clip` function cuts a mesh at an arbitrary surface into two separate volumes V_1 and V_2. One of these volumes along with the surface inside the mesh

[8] `https://vtk.org/doc/nightly/html/classvtkMarchingCubes.html`
[9] `https://vtk.org/doc/nightly/html/classvtkFloatArray.html`
[10] 4.13 - `https://www.cgal.org/`
[11] `https://doc.cgal.org/latest/Polygon_mesh_processing/index.html#title17`

can then be triangulated to yield a new closed mesh. This is done by first defining each cell with respect to it's position and dimensions. The normal at each side is further set to point into the cell. Next the surface patch which should intersect the cell is chosen and it's normals are set to point into the fluid volume. The latter is done automatically by the `Clip` function if the right winding order for the mesh vertices is chosen. The function then triangulates the enclosed fluid volume, see 3.31b. The final triangulated mesh can be used directly to calculate the volume with formula 3.56.
To pass data into the `Clip` function a conversion into the `Surface_mesh<>` [12] class is necessary. To reduce overhead cells and intersecting surface patches are cached once and only a few operations to update vertex positions are necessary in each iteration. The surface patches are created by determining the area of the mesh which intersects the cell and it's adjacent cells, see figure 3.31a. This is done by traversing the connected vertices outgoing from the ones inside the respective cell. The assumption is that such a surface patch will always intersect the respective cell since vertices are most likely moved along a vector similar to the VOF field gradient, see section 4 for more details. Using such patches further leads to a reduced number of triangles which have to be checked for intersection. Otherwise the whole mesh with all of it's triangles has to be checked for intersection with a single cell. Additionally, the volume calculation is parallelized using OpenMP [13]. Multiple threads are used to distribute the overall number of necessary cell calculations which speeds up the calculation process
Due to it's internal implementation the `Clip` function cannot be executed for some mesh configurations and thus produces errors. This is the case if, e.g., the cell is intersected multiple times by the surfaces which could be the case in a multi-phase flow application. Further, such mesh arrangements can indicate a low mesh resolution leading to an incorrect depiction of the underlying VOF field, see figure 3.10c or 5.6a. Possible remedies could be to separate the volume calculation of these cells into multiple runs one for each intersecting patch or use a higher node resolution. Yet, the implementation used in this work ignores such ambiguous cases.
Up to now the volume calculation is an expensive operation (see table 4.1). This is due to the fact that the mesh changes over time and for each cell and triangle new intersection points must be determined. It further results in a high number of triangulations for different convex and concave patterns (section 3.7).
For mean curvature calculation the approach presented by Meyer et al. [MDSB01] (section 3.4) was implemented at first. Yet, tests have shown that the results compared with the known mean curvature of a spherical dataset were not as expected. Since Meyer et al. [MDSB01] have shown that their approach approximates analytically known mesh curvatures the issue had to be located in the used implementation but could not be found during further investigations. Therefore, the solution provided by VTK in terms of the `vtkCurvatures` [14] class was chosen. The curvature calculation is based on the method presented by Hormann et al. [DHKL01]. It works by replacing each mesh edge with a cylinder with tangents lying in the plane of adjacent triangles sharing this edge. Blending these cylinders yields an approximated smooth surface on which the curvature can be retrieved by

[12] `https://doc.cgal.org/latest/Surface_mesh/classCGAL_1_1Surface__mesh.html`
[13] `https://www.openmp.org/`
[14] `https://vtk.org/doc/nightly/html/classvtkCurvatures.html`

means of integration [DHKL01]. Using differential geometry [DHKL01] along with additional area normalization the mean curvature κ_H at a vertex $\overrightarrow{P}$ can be defined as

$$\kappa_H(\overrightarrow{P}) = \frac{1}{4A_v} \sum_{i=1}^{N} \left\| \overrightarrow{e_i} \right\| \cdot |\beta_i| \tag{3.75}$$

where N is the number of directly connected vertices, $\overrightarrow{e_i}$ is the edge of $\overrightarrow{P}$ and the connected vertex with index i. β_i is the angle between the normals of the triangle sharing $\overrightarrow{P}$ and the vertex with index i. A_v is the Voronoi area (see formula 3.50) of the triangles sharing $\overrightarrow{P}$. Tests in the context of this work have shown that this approach yields the correct mean curvature while being faster than the previously used erroneous code. To use the mesh data inside the `vtkCurvatures` class a conversion to the `vtkDataObject` [15] is necessary. Thus, for each iteration the basic mesh connection is saved and only vertices need to be shifted to their new position before the curvature calculation is applied.
Other operations such as the calculation of the gradient, refinement and Laplacian smoothing were implemented using the approaches presented in their respective section. For gradient calculation only cell data is needed. Like mentioned above this data is created at the beginning and reused in later iterations. Further, the gradient at the cell nodes is calculated only once for each cell and cached. This can be done since the underlying grid does not change over time. Saving vertex to cell connections like mentioned above further allows a faster lookup for the corresponding cell and it's node based gradients. Refinement and Laplacian smoothing are executed on the created surface mesh. The refinement operation assigns newly created vertices to containing cells and tracks them over a cell neighborhood like it was mentioned above for the MC algorithm. For Laplacian smoothing each new vertex position is calculated by averaging over the directly connected vertices (see section 3.5). Using OpenMP multiple new vertex positions are calculated in parallel.
Other implementation specific details are mentioned in section 4. Yet, caching and tracking techniques are always used in combination with OpenMP if possible.
Within the next sections a scheme to correct the cell volume along with minimization of the mean curvature is developed step by step and tested in the end.

[15] `https://vtk.org/doc/nightly/html/classvtkDataObject.html`

4 Operational schemes and extensions

The goal of this work is to create an accurate representation of the real interface. In section 3 the necessary operations to create and adapt the surface mesh were defined. The next step is thus to find a scheme that uses these operations in order to correct this volume by shifting the mesh vertices. This scheme must further be combined with smoothing and mesh refinement to lower the overall mean curvature and make the correction more precise. Tests were done on a simple mesh (spherical dataset, $51x51x51$ nodes) at first. In section 5 more complex meshs are used to verify the generally applicability.
In the following section different approaches to correct the volume inside cells were investigated.

4.1 Volume correction

After an isosurface has been attained using the MC algorithm the volume enclosed by the mesh and cells deviates from the real volume defined by the VOF value. In theory different approaches to correct this volume exist.
One of these is to calculate a linear shift for each vertex along a certain direction, e.g., the gradient in such a way that the volume can be corrected directly. The volume of a mesh directly relates to the size of the surface triangles, see formula 3.56. However, shifting a vertex does not change the area of the triangle and hence the volume of the tetrahedron sharing this vertex in a linear matter, see figure 4.1. Regarding the possible cell arrangements, mesh-cell intersections and vertex connection a

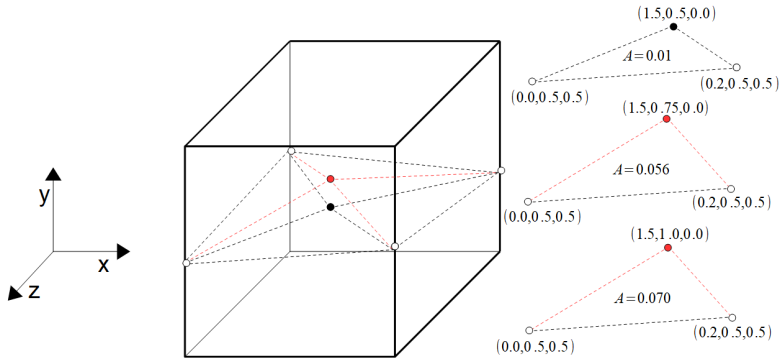


Figure 4.1: Example of vertex shifts and influence on triangle areas. A linear shift of $(0.0, 0.25, 0.0)$ is added two times on the vertex depicted by the black point which leads to the red point. Here, the mesh vertices are perfectly aligned with the cube edges. However, this is not the case for an arbitrary mesh. Even in such a simple example the change of the resulting triangle area A is not linear.

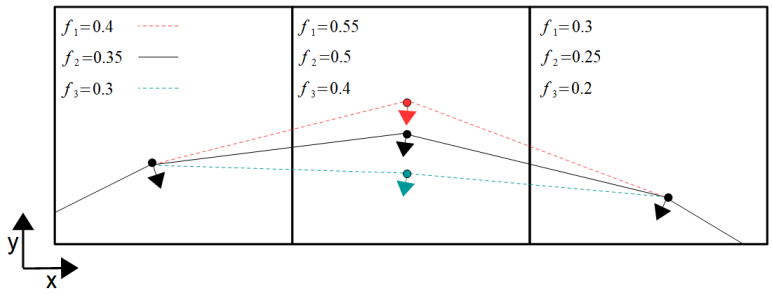


Figure 4.2: Example of vertex shifts and influence on the volume in 2D. The correct volume in each cell is achieved by choosing the mesh defined by the black vertices. Moving the vertices in the middle cell to the red position increases the volume in each cell. Moving the vertices to the green position decreases the volume in each cell. VOF field gradients are depicted at the cell vertices using arrows.

correction of the volume inside a cell by using a calculated shift is thus only possible in very simple cases. Further, a shift of a vertex changes the closed mesh and therefore the volume in adjacent cells. A succeeding shift of the vertices in adjacent cells is then necessary which again alters the volume in the original cell, see figure 4.2 for an example. Due to this behavior a history of the volume changes related to preceding shifts cannot be used to derive extrapolate a final shift. Therefore, it is not possible to find a perfect shift for each vertex in advance.

Other approaches could be used to shift the vertices on a surface that is already smooth and fulfills the underlying property of the cell volume. Such a surface could in theory be created by methods like the Weighted Essentially Nonoscillatory Scheme (WENO) [KSW+12]. It is based on the construction of cell interfaces by fitting a higher order polynomial function over adjacent cells. This polynomial is defined with respect to the integrated fluid volume of the cell. Due to the connection over adjacent cells and an used weighting scheme the resulting surface should already yield a low mean curvature which makes additional smoothing unnecessary. However, the extension of this approach to 3D and especially to the VOF method is not straightforward. The orientation of a surface described by a polynomial function $S(u,v)$ must be chosen for each cell so that it fits the underlying cell arrangement. The integration over the coordinates (u,v) must then relate to the volume of the cell defined by the VOF value and existing mesh vertices have to be shifted to the closest point on the surface. Due to the mentioned difficulties further work has not been done to adapt WENO or a similar method to be used with VOF datasets. It also contradicts the goal of this work which is to fit a surface mesh created by the MC algorithm iteratively to the underlying VOF field since vertices could also be created directly on the surface.
Since a direct shift of the vertices and reconstructions using higher order polynomials cannot be used an iterative solution must be chosen.
Some approaches use geometric primitives in order to depict the enclosed volume. In the case of PLIC by Youngs [You82] a plane defined by the VOF field gradient is used. To yield the correct volume one possibility is to shift this plane in an iterative matter until the volume enclosed by the plane and cell inside the fluid is correct. Yet, this approach only corrects the current cell's volume and does not introduce connectivity between adjacent cells. Further, it works with piecewise linear segments (planes) which is not case for the mesh created by the MC algorithm.
With respect to the approaches discussed above and the planned combination with other operations a technique to correct the volume should have the following properties

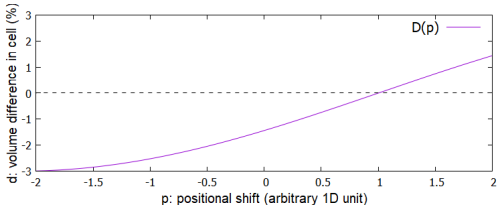


Figure 4.3: Example of the volume difference function. Graph depicts the volume difference of the containing cell in relation to the positional shift (here in 1D) of a vertice. Shifting the vertex into the positive x-direction approaches a volume difference of zero. This is a simple example to show the underlying principle of the iterative scheme. The extension to 3D is straightforward.

1. Reduction of the total volume difference with respect to the volume defined by the VOF value (see formula 3.74) in each cell converging a stable value close to zero. This can be done iteratively since smoothing also needs multiple iterations to reduce the mean curvature, see section 3.5.
2. Converging of the volume difference within few steps. This reduces the number of volume calculations since this is an expensive operation, see also table 4.1.
3. Lowering of the overall curvature to yield an overall smoother mesh. This way the smoothing operation (section 4.2) should have less impact on the volume correction and a faster and more stable convergence behavior is achieved. This is the case since for a flatter mesh the shifts created by the smoothing operation are smaller as well.
4. Containment of the mesh within all interface cells so that they can be used in the volume calculation. Additionally, vertices should not be moved out of cells. Otherwise it may not be possible to use them for volume correction in further steps. Both cases will thus increase the final deviation of the volume.

The idea is to relate the volume correction within each interface cell to the problem of finding the zero of a function $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ which assigns the position of the vertex $\overrightarrow{P} \in \mathbb{R}^3$ in space to a volume related value $d \in \mathbb{R}$ hence $D(\overrightarrow{P}) = d$. For d the current VOF value or volume of the cell containing $\overrightarrow{P}$ cannot be used since these value are only zero if the mesh is outside interface cells. A better choice is to use the volume difference of the cell containing $\overrightarrow{P}$ as d. Assuming that only one vertex is located inside each cell the goal is therefore to find the position $\overrightarrow{P}$ that yields a zero volume difference $D(\overrightarrow{P}) = 0$, see figure 4.3. This volume difference can be calculated by subtracting the measured volume (section 3.7) from the one defined by the VOF value ($f \cdot C_w \cdot C_h \cdot C_d$). A negative difference therefore depicts a volume that is too high. See figure 4.2 for a depiction in 2D[1] of a volume that is too high or too low.

An iteration scheme is thus needed to shifted the vertices into the direction for which the function D is zero. To approximate the position of the zero value of a function Newton iteration can be used [SG19]. In 3D the it can be defined as [SG19]

$$\overrightarrow{P_{i+1}} = \overrightarrow{P_i} - \nabla f(\overrightarrow{P_i}) \cdot f(\overrightarrow{P_i}) \tag{4.1}$$

[1] 2D is generally used to depict certain properties in a more understandable way. The transfer to 3D is straightforward.

where f is the function of which the zero value is to be found and $\overrightarrow{P_i}$ is the current position of point $\overrightarrow{P}$ in 3D.
Under normal conditions the Newton iteration converges quadratically adding two digits of precision for every iteration [SGD18]. The idea is therefore that a fast convergence of the correct volume in each cell can be achieved with this method. Further, by choosing appropriate vertex shifts the influence of the volume of adjacent cell should be minimized yielding a faster reduction of the overall volume difference. Also, results could prove that other iterative schemes to correct the volume inside cells might work as well or not work at all.
To use the Newton iteration the idea is to first set $\nabla f(\overrightarrow{P_i})$ to the normalized gradient of the VOF field. Like it was mentioned in section 3.3 this is a better choice than using the interpolated normals of the mesh since these do not contain any information about the flow of the VOF field. For $f(\overrightarrow{P_i})$ the volume difference inside the cell is used hence $f(\overrightarrow{P_i}) = D(\overrightarrow{P_i})$. Formula 4.1 then becomes

$$\overrightarrow{P_{i+1}} = \overrightarrow{P_i} - \nabla f(\overrightarrow{P_i}) \cdot D(\overrightarrow{P_i}) \tag{4.2}$$

However, for a volume which is too low the vertex should be moved out of the fluid yielding a higher volume inside the cell, figure 4.2. Therefore, a slight variation of the method is used in which the gradient is inverted and formula 4.2 can be rewritten to

$$\overrightarrow{P_{i+1}} = \overrightarrow{P_i} + \nabla f(\overrightarrow{P_i}) \cdot D(\overrightarrow{P_i}) \tag{4.3}$$

which is the final iterative scheme that to correct the volume inside cells.
The value returned by D should be used independently for every cell size. For very small cells having the normalized gradient scaled with a high value could lead to vertices that are moved out of interface cells. Unlike a scaling based on a percentual value (figure 4.3) the idea is to use the calculated volume difference. This way the gradient is scaled by a fraction of the cell dimension which should lead to a more stable convergence behavior and less influence on the volume of adjacent cells.
In figure 4.4a a depiction of the shift operation in 2D can be seen. The 3D case works analogously. The goal is to keep vertices inside cells and not further increase the mean curvature on the mesh. In each iteration the vertices should therefore be moved only once along the scaled gradient. Correcting the volume for each cell separately could lead to a mesh with a higher mean curvature. If, e.g., the volume inside a cell is too low the vertex is shifted along the negative gradient multiple times to correct the volume. These multiple shifts highly alter the volume in adjacent cells. The contained vertices then need to be moved for a higher distance in order to compensate the created volume change. This introduces peaks on the mesh and increases the curvature, see figure 4.4b. Further, it could lead to vertices being moved out of cells over multiple iterations. Having only one correction step per vertex alters the volume of adjacent cells only slightly. This should lead to a more stable and precise volume correction. Moreover, vertices should be shifted freely inside cells. Moving the vertices only along the cell edges used by the MC algorithm could shrink the volume difference for a few iterations. Yet, this constraints the shift direction and could lead to an increasing mean curvature since the vertices reside at cell boundaries. Further, volume corrections cannot be done if the gradient direction is orthogonal to the edge.
Over multiple iterations vertices can be shifted out of cells. Yet, like mentioned above vertices should reside inside cells that contain parts of the interface. To achieve that volume correction can also be done in non-interface cells if they contain a vertice. Yet, more expensive volume calculations are necessary. A direct shift along the gradient direction back into the closest interface cell could

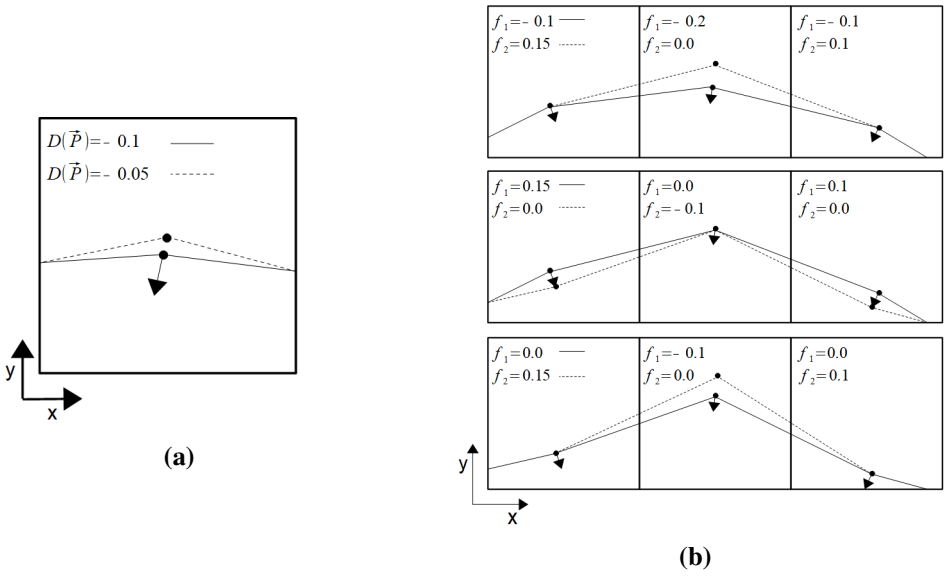
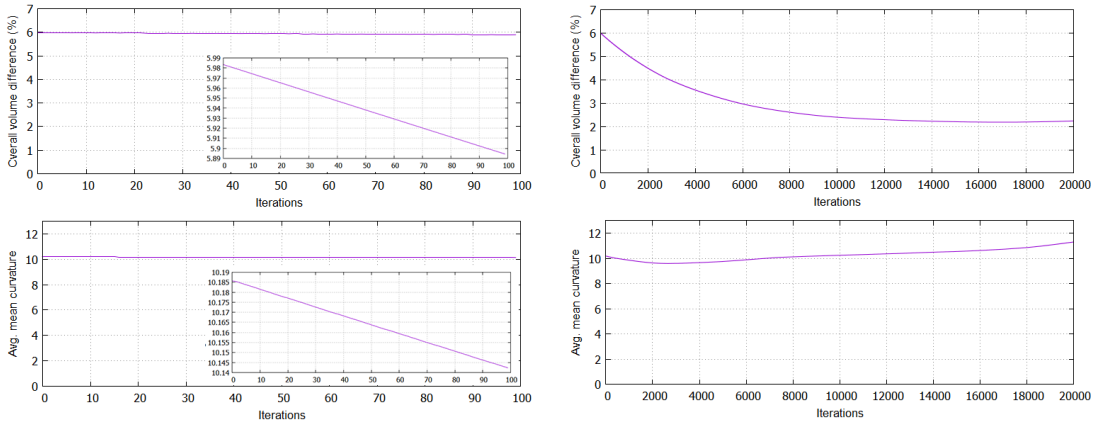


Figure 4.4: Example of shift operation and increase of curvature in 2D:
(**a**) 2D depiction of one step of the iterative scheme which is used to correct the volume. The fluid volume inside the cell enclosed by the mesh is too low. Therefore, the vertex (black point) is shifted along the reversed gradient direction. This reduces the total volume difference inside the cell.
(**b**) Example of higher curvature by correcting the volume in each cell over multiple iterations. In the first row the volume (area in 2D) in the middle cell was too low and has been corrected. This alters the volume in the adjacent cells. As soon as the volume is corrected in these cells the volume in the middle cell changes again. Over multiple iterations this behavior leads to peaks on the mesh. This effect is especially strong if vertices are shifted over a high distance.

also be possible, see figure 4.6b. However, allowing vertices to generally leave interface cells can lead to a similar behavior as described in figure 4.4b. Therefore, an additional check is done to avoid that vertices leave interface cells. If a vertex cannot be shifted the shift vector is divided and added iteratively onto the vertex position, see figure 4.6b. This way a slight adaption can still be made even if the original shift would move the vertex out of the cell.
To reduce the iteration steps the volume correction is executed on the mesh created by the MC algorithm at first to approximate the coarse structure of the interface. This mesh has less vertices and overall bigger surface triangles. Changes on these triangles have a higher impact on the cell volume, see formula 3.56. The algorithm should therefore converge to a stable volume difference in each cell within fewer iterations. Later the correction can be done one a finer mesh in combination with smoothing, see section 4.3. This mesh has smaller triangles and the volume can be adapted more precisely. Thus, after refinement the mesh volume difference should be even lower.

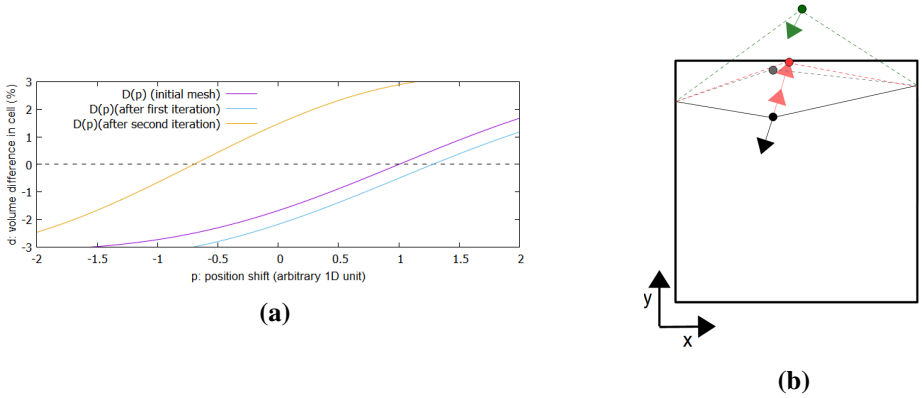


(a) **(b)**

Figure 4.5: Results of the scheme based on Newton iteration:
(**a**) For 100 iterations the volume difference and average absolute mean curvature seem to convergence (big plot). Yet, in the small plot the y-axis is not scaled and it can be seen that for this number of iterations no convergence behavior is present even though the volume difference is decreasing as expected.
(**b**) For 20000 iterations the volume difference is decreasing at first. Yet, after a high number of iterations it increases again. This is due to a similar behavior as is can be seen in 4.4b. Additionally, vertices are not moved outside interface cells but are stopped. This leads to vertices that cannot move further to correct the volume. The curvature increases also due to the effect described in figure 4.4b.

For validation the summed volume difference as percentual value[2] and the average mean curvature [3] over all cells is plotted. This way the evolution of the mesh over multiple iterations can be observed in detail. With respect to the properties of the used iteration scheme a stable volume difference value should be achieved after a reasonable amount of iterations.
For a sphere within a grid of $51x51x51$ nodes the evolution of the volume difference using formula 4.3 and the average absolute mean curvature can be seen in figure 4.5a. The property of the Newton iteration with respect to the quadratic convergence behavior cannot be observe. The volume difference does decrease slowly over many iterations. After a high number of iterations the volume difference does not converge to a stable value but does increase again, see figure 4.5b. This is also the case for the average mean curvature. For a more complex mesh (droplet collision) with higher node resolution this effect is even stronger. See figure 4.7 for a comparison.
One explanation for the slow decrease of the volume difference are small vertex shifts. The gradient is scaled using the volume difference and depending on the cell dimension this results in a short shift vector. An uniform grid in which each cell dimension has a length of about 0.02 units [4] leads to a cell volume of 0.000008 cube units. For the complex mesh with a cell dimension length of

[2] It is retrieved by summing up the current cell volumes divided by the ones defined using the VOF value and cell dimensions.
[3] It is retrieved by summing up the absolute mean curvature for every vertex divided by the number of vertices. The absolute value is taken since only the tendency to shrink the curvature ($\kappa_H \rightarrow 0$) and not the curvature direction is investigated.
[4] Here, an unit is referred to $1.0/$((#domain cells)-1)

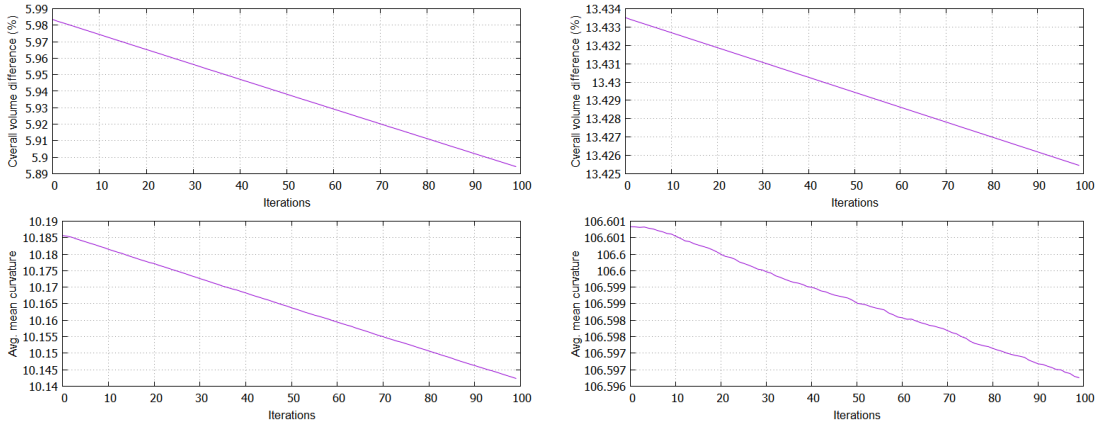


(a) **(b)**

Figure 4.6: Example of multiple difference functions and vertex movement correction:
(**a**) After each iteration the volume difference function D of a vertex has the zero value on a different position. Therefore, the shift direction can fluctuate from iteration to iteration. This results in a higher runtime.
(**b**) If the new position of a vertex (black point) lies outside interface cells (green point) the movement is divided into multiple parts (red arrows) yielding a new valid position. This is done instead of a shift back along the new gradient direction at the vertex (gray point). Both approaches result in similar positions.

about ~0.004 this yields a volume of 0.000000064 cube units. Even though the cell dimensions differ by a factor of 5 the volumes differ by a factor of 125. Since the gradient is further scaled by a fraction of the volume the shift in each iteration has only little influence on the volume inside a cell. Another explanation for the general slow decrease of the volume difference is the used iteration scheme. It is based on finding the zero of a function defined by the position of the vertices. Like mentioned above the shift of a vertex influences the volume of adjacent cells. After one iteration this function changes for each vertex so that the new zero position is not the same anymore, see figure 4.6a. Together, both effects can explain the slow decrease of the volume difference.
A complex mesh has a general higher volume deviation in the beginning which necessitates more iterations to reduce the volume difference significantly. One reason for that are numerical inaccuracies that result in cells with very low[5] or high[6] VOF value. The correction step of the the MC approach presented in section 3.2 therefore leads to inaccurate representations of the VOF value at cell edges. Further, due to a low node resolution some cell arrangements approximate the interface depicted by the underlying VOF field only coarsely. This leads to strong deviations of VOF values in neighboring cells and can also affect the final result of the interpolation. See figure 4.8a for an example. The starting value of the average mean curvature for different meshs further depends on the cell structure of the dataset and the mesh created by the MC algorithm. For a complex mesh this value can thus be arbitrarily high and is only used as measurement for the overall smoothness of the mesh.
Like mentioned above the volume difference does increase again over time. Vertices are kept inside interface cells in order to further use them in following iterations. Yet, this creates vertices which are stuck at cell boundaries and hence leads to an increased deviation of the volume in some of the

[5] E.g., 0.0060992 in case of the tested datasets.
[6] E.g., 0.999968 in case of the tested datasets.



(a) **(b)**

Figure 4.7: Example volume correction over 100 iterations on different meshs:
(**a**) Volume correction on a topologically simple mesh (sphere, $51x51x51$ nodes). It can be seen that the volume difference is reduced by 0.089%.
(**b**) For a more complex mesh (droplet collision, $251x251x251$ nodes) with more cells the volume difference is only reduced by 0.008%.

cells. Moreover, the average mean curvature drops at the beginning but increases again over time. The mesh created by the MC algorithm has a lot of peaks and thus areas with a high mean curvature (see section 3.4). When the vertices are shifted in direction of the gradient the volume difference in each cell is getting lower. Further, the mesh is adapted more and more to the underlying VOF field which yields a lower mean curvature. However, due to the low resolution of the mesh peaks are introduced again after a a high number of iterations. Both observations can be explained by a similar behavior as depicted in figure 4.4b (alternating volume corrections over adjacent cells). It results in an increasing curvature and volume difference. The idea is therefore to combine smoothing with volume correction. This is discussed in section 4.2.

In figure 4.5b it can be seen that many iteration steps are necessary in order to shrink the volume difference to the lowest possible value. In every iteration a calculation of the current enclosed volume inside cells is necessary. With respect to the average duration of an iteration (see table 4.1 [7]) the accumulated runtime to reduce the volume difference can be problematic. A higher number of cells and therefore triangles increases the duration of the volume correction even more. Using other methods or libraries to calculate the volume may result in an overall lower runtime. However, another reliable technique which is not based on finding intersection patterns and triangulation was not found in the context of this work. Further, the high number of iterations which is needed to shrink the volume and the increase in runtime by adding more triangles should persist for any other method to calculate the volume.

Like mentioned above one reason for the slow reduction of the volume difference and the thus needed high number of iterations is the scaling of the gradient. To reduce the number of iterations and therefore the overall runtime an additional scaling factor $\sigma > 0$ can be used to compensate small cell volumes. The idea is that by using this factor each vertex is shifted by a longer distance

[7] Here, runtime is measured based on the implementation used for testing.

Mesh (nodes per dimension)	#Triangles of MC mesh	#Cells used by MC	Duration (s)
Sphere (21)	968	488	0.095
Sphere (51)	5864	2936	0.525
Sphere (101)	23528	11768	2.007
Droplet Collision #2 (257)	30464	15184	2.450
Droplet Collision #3 (257)	40940	20436	4.115

Table 4.1: Durations[8] of the volume calculation using the test implementation [9] on meshs created by the MC algorithm. Independent of the underlying complexity of the dataset a higher number of cells and therefore triangles results in a higher runtime. The calculation was parallelized using OpenMP as it was mentioned in section 3.8.

and therefore has a higher impact on the cell's volume. The scheme used in 4.3 can thus be rewritten to

$$\overrightarrow{P_{i+1}} = \overrightarrow{P_i} + \sigma \cdot \nabla f(\overrightarrow{P_i}) \cdot D(\overrightarrow{P_i}) \tag{4.4}$$

The size of the factor depends on the used dataset. If the cell resolution is high a bigger factor is needed in order to shrink the volume difference faster. This is due to the basic scaling of the gradient with respect to a fraction of the cell volume (see above). In figure 4.8b the impact of different factors on the volume difference can be seen.

As expected the volume difference does decrease faster until it reaches a global minimum. Yet, for some factors a sudden increase of the volume difference can be observed. This effect is created by empty interface cells that do not contain parts of the surface mesh. Therefore, no volume is enclosed in these cells and the overall volume difference increases [10]. Test have shown that the affected interface cells have a VOF value close to zero or one and are intersected by only a small MC surface patch at the beginning. If contained vertices are shifted into adjacent cells the mesh can thus leave the interface cell. See figures 4.9a and 4.18c for examples. For some cell configurations this has also been observed without adding an additional factor σ. Yet, the effect is stronger when a high factor is used. In some cases the mesh is shifted back into empty interface cells automatically if the volume correction alters the mesh in adjacent cells. However, in order to generally circumvent empty interface cells vertices must be shifted back manually before the volume is calculated. This can be done, e.g., by choosing the closest vertex on the cell's corresponding surface patch (see section 3.8) and shifting it in the direction of the empty cell's center. However, the problem with this approach is that artificial peaks are created on the surface mesh leading to a higher mean curvature and volume deviations in adjacent cells which increases the overall volume difference. These peaks are especially visible if the mesh has been smoothed before like it is done in the next section. Further, the mesh can be shifted out of the cell again in any succeeding iteration. Another

[8] Test system:
Processor: AMD Ryzen 5 2600 Six-Core Processor, 3400 Mhz
Memory: 16 GB DDR4
Graphics card: NVIDIA GeForce GTX 1050 Ti
Operating system: Windows 10 64bit

[9] It can be found at `https://github.com/MaBa90/VOFVis`.

[10] Cells which are added to close the mesh are not considered in this case since they should not contain parts of the interface after multiple iterations, see section 3.2.

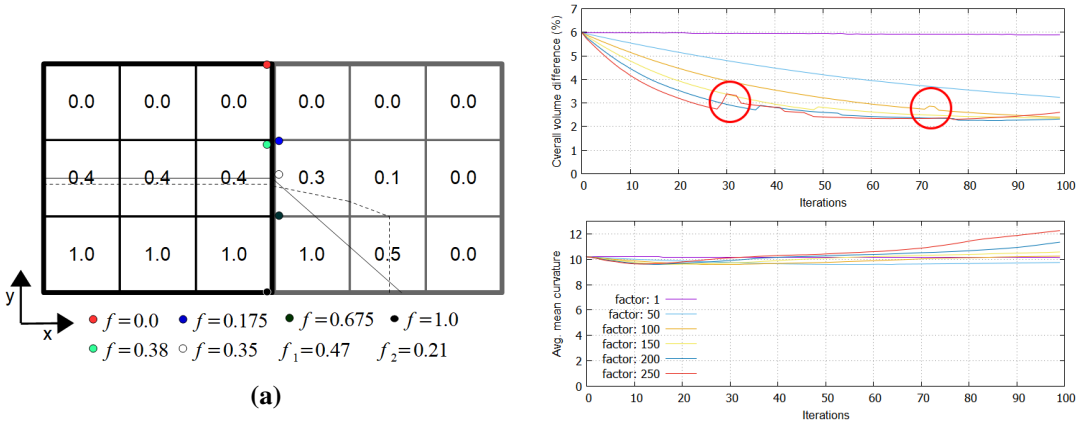
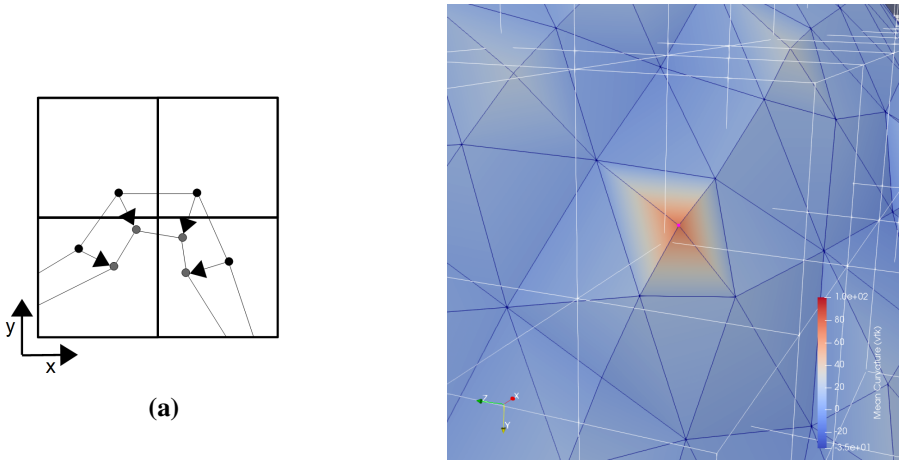


Figure 4.8: 2D example of volume deviation due to a low node resolution and volume correction using a factor:
(**a**) For a low node resolution two adjacent cells have VOF values f_1 and f_2 that deviate strongly from each other. Here, the white point depicts the interpolated position depending on the VOF values at the blue and dark green nodes (high resolution) averaged over four adjacent cells. The light green point depicts the interpolated position depending on the VOF value at the red and black nodes (low resolution) averaged over four adjacent cells. It can be seen that the white point is closer to the interface defined by the higher node resolution (dashed line). In relation the light green point is further away from the interface defined by the lower node resolution (continuous line). This thus leads to a greater deviation from the real interface.
(**b**) For an additional scaling factor σ the volume difference does decrease faster (sphere dataset, $51x51x51$ nodes). Yet, over time the volume increases again like it was the case without σ. Additionally, the mesh can leave cells. This leads to a sudden increase of the volume difference (red circles). This effect is stronger with a higher factor. However, the mesh can be moved back into cells due to the volume correction within adjacent cell. A larger shift of the vertices can also magnify the effect explained in figure 4.4b. This leads to an increasing average mean curvature.

approach is to keep at least one vertex inside cells. Yet, this also leads to peaks on the surface mesh if the surrounding area of the mesh changes. To this end no satisfying solution could be found that can prevent this kind of behavior and no additional empty cell correction is done. Only volume differences of cells which are intersected by the mesh are considered. This is also the case for the schemes explained in the next sections. This way It leads to an overall better visual result due to a lower mean curvature. Further, small cell patches that are created due to numerical errors are ignored automatically if the patch is moved out of the cell. Otherwise they would lead to an increasing overall volume difference.
It can be seen that for a high factor the mean curvature increases. This can be explained by the behavior mentioned in figure 4.4b. It leads to additional peaks in the mesh. Having a bigger shift magnifies this effect and further mesh processing is necessary. This is done by adding smoothing (see section 4.2) to flatten the mesh locally. Yet, in order reduce the creation of peaks and maintain a lower mean curvature during volume correction an interpolation scheme between cell volume

differences was tested as well. It was based on inverse distance weighting using the distances of mesh vertices from the cell centers of surrounding cells. This way a vertex close to the border of one cell is shifted also with respect to the volume difference of the adjacent cell. The idea was that this could prevent high alternations of shifts distances at cell borders. However, for narrow cell structures like, e.g, tubes this is problematic since the volume difference in opposing cells influences the new vertex positions as well. This can lead to undefined behavior of the volume correction.
An automatic method to choose an appropriate factor with respect to stability (less empty cells, stable volume difference and mean curvature) and speed (faster reduction of volume difference) has not been found in the context of this work. Up to this end multiple tests of the same mesh are necessary in order to determine a good factor σ.
As mentioned above the other reason for a slow decrease of the volume difference is the influence of vertex shifts on the volume in adjacent cells. During the correction step each vertex is shifted into the direction of zero volume difference in it's containing cell. This results in a change of the zero position in each adjacent cell for the next iteration. Slight oscillations of the vertices positions can occur slowing down the decrease of the volume difference. To make sure that such a behavior is not solely due to the nature of the applied iterative scheme another method was tested as well. From a starting position the vertex is steered to it's final position[11] along the scaled gradient over multiple iterations. This positional change can therefore be defined as an initial value problem. Hence, a scheme such as the fourth-order Runge-Kutta (RK) [Kut01] can be used. The difference function (here the scaled gradient) is sampled at multiple mesh deviations leading to a final positional shift for the current iteration. Yet, the problem of alternating cell volumes persists and a stable and fast volume reduction could only be achieved on an already refined mesh with a high step size. Further, multiple volume corrections (four in the case of fourth-order RK) are necessary to calculate the weights. This results in an highly increased runtime for the algorithm due to the additional triangles of the mesh. In the context of this work Newton iteration thus stays the basic method to correct the volume. Only a single volume calculation per cell is necessary and no other iterative method that has a different underlying assumption to solve the mentioned problems could be found. The problem of vertex shifts influencing the volume in adjacent cells persists. Yet, using refinement (see section 4.3) smaller triangles reside inside cells. This reduces the influence area of vertices.
In figure 4.10 results of multiple iterations volume correction with an additional factor and a comparison to PLIC can be seen. In the case of the tested dataset (sphere on a $51x51x51$ nodes grid) PLIC shows an accurate representation without artifacts. The overall volume difference is lowered as expected. This is especially the case for areas that have a high volume deviation at the beginning due to the MC approximation. Yet, for some cells the volume difference begins to deviate more with an increasing factor. Such a behavior can be explained if the volume correction was done in the adjacent cell at the end or if vertices are stuck at cell borders (see above). Also, like mentioned above having a high factor can introduce artificial peaks on the mesh. It leads to a bumpier surface with higher mean curvature, see figure 4.8b. This can also be observed as deviation from the PLIC depiction in figure 4.10. The idea is to avoid such a behavior by combining smoothing with volume correction (see section 4.2).
In this section an iterative approach to correct the volume inside cells was presented. It is capable of decreasing the overall volume difference in cells and can in some cases reduce the local mean curvature of the mesh. Yet, some of the properties defined at the beginning of this section such as containment of the mesh inside interface cells or convergence of the volume difference to a stable

[11] Here, in the best case this position is defined at which the volume difference in each cell is zero.



(a) **(b)**

Figure 4.9: 2D example of empty cells and manually created surface bumps in 3D:
(**a**) The black points are shifted (black arrows) due to volume correction. This leads to empty cells in the top row. Yet, the result is valid since the bottom cells have a VOF value between zero and one.
(**b**) The red vertex has been shifted back into an empty interface cell manually. It can be seen that the surface is getting bumpier and hence the mean curvature does increase (see color scale). In the next iteration the volume of this and adjacent cells must be corrected again. This, however, could lead to the cell being empty again. Cells are depicted as white boxes.

value within few steps cannot be fulfilled. A manually chosen scaling factor σ can be used to speed up the reduction of the volume difference. However, this comes with the cost of an increasing mean curvature and empty interface cells. A shift of the mesh back into these cells is not done in favor of keeping the mean curvature low. Up to this end the algorithm must be stopped after a fixed number of iterations or if the volume difference starts to increase again. Due to the mentioned problems additional improvements are necessary:

1. Further reduce the mean curvature of the created mesh. For that smoothing should be used in combination with volume correction. This could also reduce the overall runtime by removing high volume deviations (due to peaks) between adjacent cells.
2. Make the volume correction more precise to fit the coarse mesh better to the underlying VOF field. For that mesh refinement is necessary. It creates smaller triangles with closer located vertices. A change of these triangles has less impact on the cell volume.

These improvements, their combination as well as the practicability of the resulting schemes are discussed in the following sections.

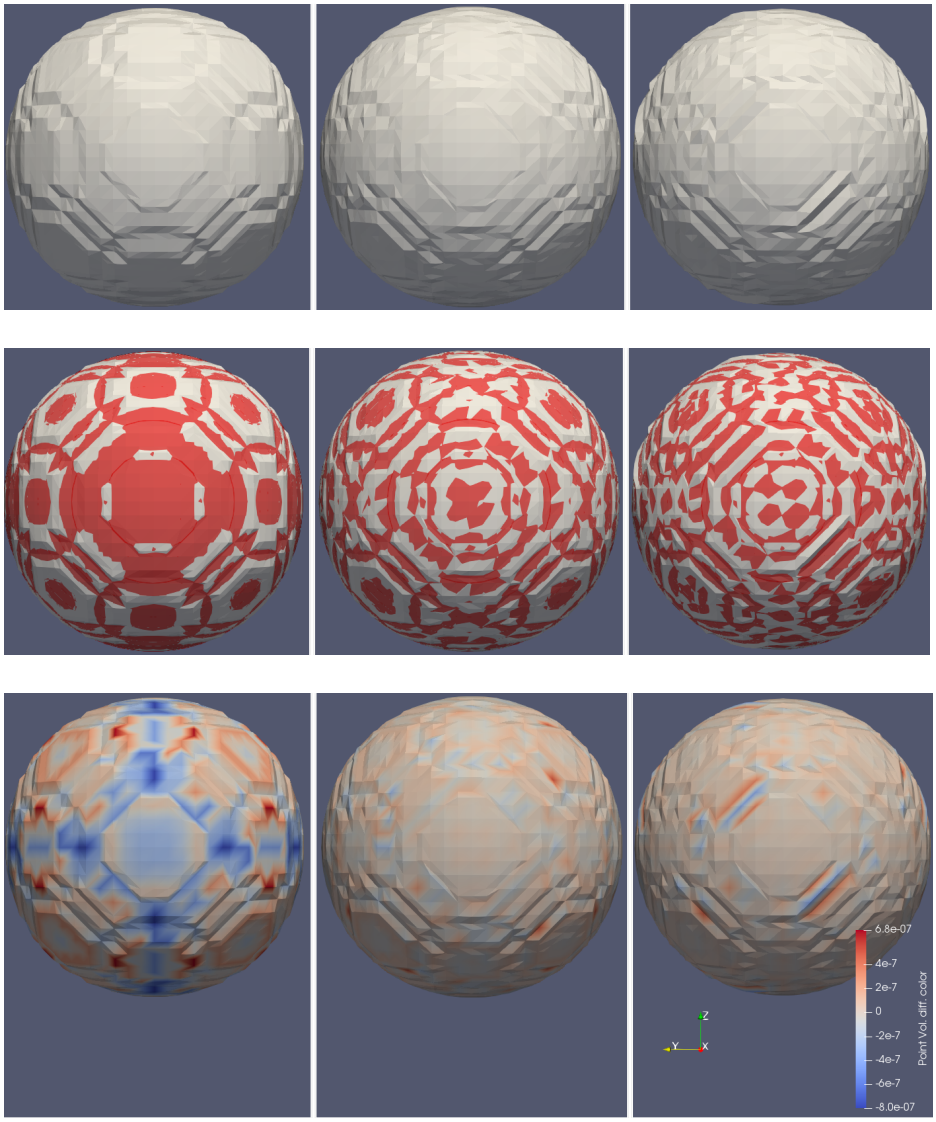


Figure 4.10: Comparison of results achieved using the described volume correction approach on a sphere ($51x51x51$ nodes) with 100 iterations. For mesh in the columns a different factor σ (1, 100, 250) was used to scale the gradient. The first row shows the final corrected mesh. It can be seen that a higher scale factor also leads to a bumpier surface (right column). In the second row a direct comparison to PLIC (red transparent surface) is shown. The mesh created without an additional factor has less peaks and it's surface is therefore closer to the PLIC elements. In the bottom row the volume difference of the containing cell at each vertex is shown. The deviation of the volume at certain areas is reduced for the meshs with a factor of 100. For a factor of 250 the volume difference starts to increase again (more red and blue areas) for some of the cells. The dataset has been chosen since the PLIC visualization does not contain artifacts. Hence, it can be user as visual reference.

Algorithm 4.1 Execution order of a single iteration with smoothing.

```
procedure ITERATION(M, C, σ)        // M = mesh data, C = cell data, σ = correction factor
    SMOOTH(M, C)
    CALCULATEVOLUME(M, C)
    CORRECTVOLUME(M, C, σ)
end procedure
```

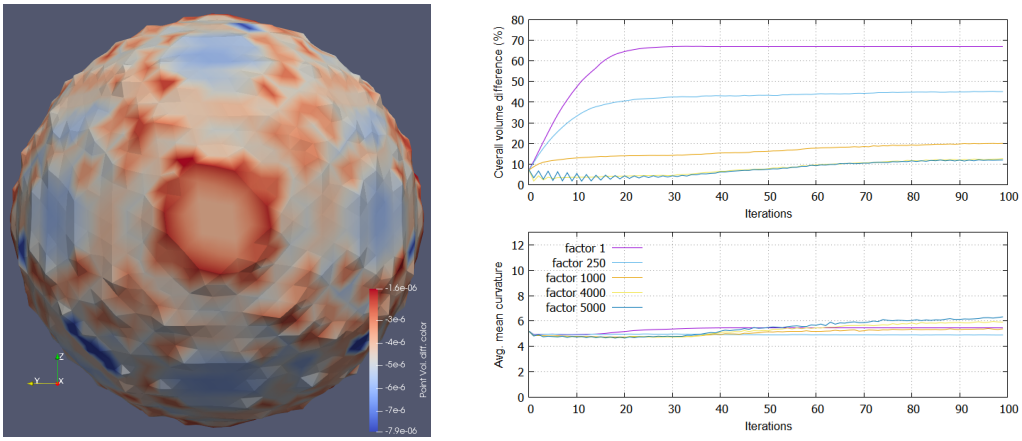
4.2 Combination with smoothing

In order to have a more accurate representation of the actual free surface (see section 3.5) the volume difference in cells along with the mean curvature of the mesh must be reduced. In the last section an approach to decrease the volume difference within cells has been presented. Yet, the mean curvature is reduced only slightly in some cases. The next step is therefore to combine volume correction with Laplacian smoothing so that the mean curvature is reduced while a low volume difference is maintained. With respect to the results of the previous section a combination should have the following properties:

1. Converge to a minimal mean curvature. For datasets with analytically known mean curvatures this should be close to the actual one[12].
2. A low overall volume difference. Further, the deviation of the volume should decrease along with the mean curvature. In the best case both should approach the minimal value at the same time.
3. Keep or decrease the number of iterations which are needed to achieve the minimal volume difference.

Smoothing should be used before volume correction is done. This way the iteration always stops with the corrected mesh and no further volume deviation is introduced. The execution order of one iteration can be seen in algorithm 4.1.
In figure 4.11a a first result of added Laplacian smoothing on a spherical dataset is shown. Due to it's shrinking behavior (see section 3.5) Laplacian smoothing significantly alters the coarse mesh structure with each single iteration. The mesh is thus shifted into the direction of the sphere center. Since vertices are only allowed to stay inside interface cells the movement is stopped at cell boundaries. This leads to a maximal negative volume difference after a few iterations, see figure 4.11a. In the case of this spherical dataset the shrinkage effect is especially strong since almost no flat surface regions exist. Due to the negative volume difference vertices are shifted back into cells during volume correction. This also results in an increasing mean curvature for a high factor σ, see figure 4.11b.
In order to maintain a reduction of the volume difference the impact of the smoothing operation must be lowered. In section 4.1 it was mentioned that an additional factor can be introduced to speed up the volume correction. Yet, counteract mesh shrinking by using a very high factor σ results in an effect similar as the one described in figure 4.4b. Vertices are shifted along higher distances which is especially the case if the volume deviates strongly inside the cell. Thus, peaks on the surface are created and the volume in adjacent cells is altered. The smoothing operation now

[12] Small deviations can persist, e.g., due to numerical errors or the number of used iterations.



(a) **(b)**

Figure 4.11: Result of the first smoothing approach:
(**a**) Since the smoothing tends to shrink the mesh the surface is shifted into the direction of the sphere center and stopped at cell boundaries. This results in an increasing negative volume difference in each cell (color scale).
(**b**) For low factors the smoothing operation is stronger than the volume correction. The mesh is shifted into the direction of the cell center but kept inside interface cells. The mean curvature is thus decreasing significantly after the first smoothing step[13].For high factors the volume correction has a higher impact and the overall volume difference is lowered. Yet, with a high factor the volume correction locally leads to peaks and the volume in adjacent cells is altered. The overall mean curvature as well as the volume difference thus increase again over time. Smoothing tends to flatten such peaks and in combination both approaches can yield oscillations and a bumpier surface.

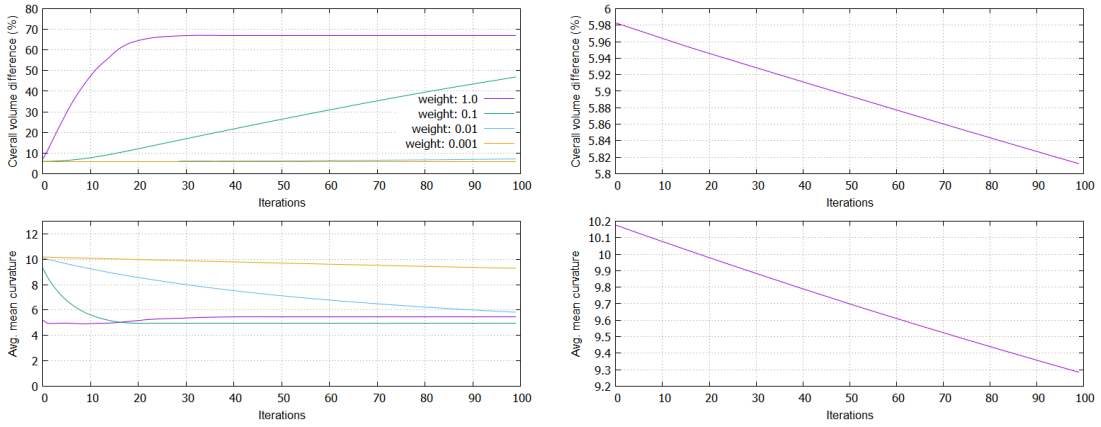
shrinks the mesh especially at such peaks which again changes the volume in adjacent cells. As it can be seen in figure 4.11b this leads to an increasing mean curvature and volume difference as well as oscillations of the latter.

An appropriate combination of smoothing and volume difference is therefore necessary. For that an additional weight $0 < \lambda \leq 1.0$ [Tau99] is introduced to scale the shift vector created by smoothing. Formula 3.51 is thus rewritten to

$$\overrightarrow{P_n} = \lambda \left(\frac{1}{N} \sum_{i=1}^{N} \overrightarrow{P_i} \right) \tag{4.5}$$

The effect of such a weight with $\sigma = 1$ can be seen in figure 4.12a. Only for a small weight both the volume difference and curvature decrease over time, see figure 4.12b. In figure 4.13 it is shown that for an appropriate combination of factor σ and weight λ the desired behavior can be achieved in the case of a simple spherical dataset ($51x51x51$ nodes). Both the mean curvature and volume difference are maintained at a stable low value and the resulting mesh structure is close to an approximation achieved using PLIC. The mean curvature also approaches the analytical known

[13]The mean curvature is measured after the smoothing step. In the case of the tested dataset 10.18 was the measured mean curvature of the base mesh, see, e.g., figure 4.8b.



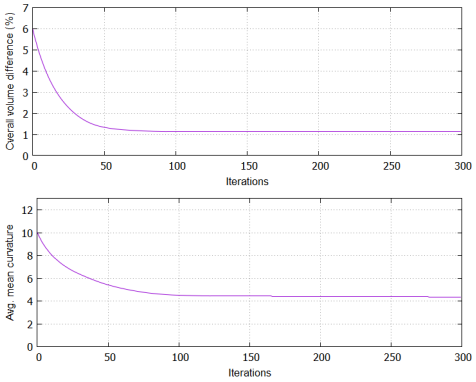
(a) **(b)**

Figure 4.12: Test of the weighted smoothing operation:
(**a**) Different values for λ have been chosen to show the behavior over multiple timesteps with a factor of $\sigma = 1$. For a lower weight the mean curvature does decrease slower but the final volume difference is closer to zero. The average mean curvature value of the base mesh at the beginning is 10.18. Yet, it's measurement is done after the first smoothing step. For $\lambda = 1$ it can be seen that the mesh is flattened significantly stronger.
(**b**) By choosing a small weight (e.g. $\lambda = 0.001$) both the volume difference and average mean curvature can be decreased over time. However, in comparison to figure 4.7a it can be seen the mean curvature decreases faster since additional smoothing is done in each iteration.

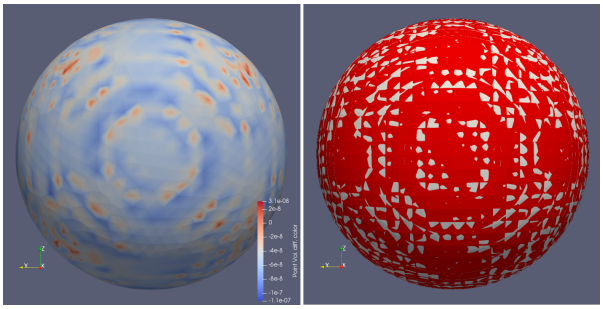
value of the sphere ($\kappa_H = 4.0$). Yet, since the shifts created by smoothing alter the volume inside cells the number of iterations needed to decrease the volume difference is higher. Choosing a higher factor σ in order to circumvent this problem as it was done in the last section can lead to the behavior shown in figure 4.11b. This again necessitates a higher smoothing weight λ and can introduce instabilities (permanent increase of the volume difference) and oscillations. A tradeoff between reduction speed and accuracy is thus necessary by choosing appropriate parameters. However, the best combination of weight σ and factor λ differs from mesh to mesh due to cell arrangements and dimensions. Up to this point no method was found to choose the best fitting values automatically depending on the underlying dataset structure. Finding a combination which yields a stable reduction hence demands multiple tests with different parameters. With respect to the runtime of the volume calculation (see table 4.1) this can be problematic. A smoothing method with a mesh independent weight could thus help to create stable results.

To do that one idea is to associate the volume correction and smoothing with a fixed vector length. The shift vector created by the volume correction is weighted so that it's length is always greater than the one created by smoothing. This way smoothing is involved in the process while the impact of the volume correction on the mesh is greater. Yet, an appropriate weight between the two vectors could not be found during multiple tests. In any case the mean curvature is not reduced to the same extend as it can be seen in figure 4.13a.

In literature extensions to Laplacian smoothing can be found. One of these is, e.g., the mean curvature flow [OBB01b] where smoothing is done with respect to the mesh normal and curvature.



(a)



(b)

Figure 4.13: Results achieved using weighting smoothing and volume correction:
(**a**) Chosen weight λ of 0.018 and a factor σ of 210. Both the volume difference and smoothing are stable for the tested amount of iterations. Additionally, the mean curvature approaches the analytical value $\kappa_H = 4.0$.
(**b**) The overall volume difference decreases further. Bumps on the mesh are reduced and the result is close to the one achieved using PLIC (right side, red surface).

Smoothing is therefore stronger at curved regions, e.g., peaks created by the MC algorithm. Yet, this approach still leads to mesh shrinking [OBB01b].

The surface mesh can be seen as a basic smooth surface with additional high frequencies (noise)[Tau95]. These frequencies are created by alternations in the mesh's geometry and therefore curvature. The idea is to remove these high frequencies and thus avoid mesh shrinkage by keeping the coarse mesh structure defined by low frequencies in place. This should lead to consistent volume differences inside cells and avoid the choice of different weights for each mesh. In Taubin's method [Tau95] the smoothing operation is therefore defined in terms of a low pass filter. [14] Defining the vertices' positions as a signal consisting of different frequencies a transfer function is used in order to attenuate high frequencies above a chosen passband frequency k_P. See figure 4.14a for a depiction of such a transfer function. This leads to another weight $\mu < 0$ with $\mu < -\lambda$. In one iteration both weights are used in alternating order to achieve the properties of a low pass filter. With respect to formula 3.51 this yields

$$\overrightarrow{P_n} = \lambda \left(\frac{1}{N} \sum_{i=1}^{N} \overrightarrow{P_i} \right) \tag{4.6}$$

$$\overrightarrow{P_n} = \mu \left(\frac{1}{N} \sum_{i=1}^{N} \overrightarrow{P_i} \right) \tag{4.7}$$

[14] The basic Laplacian smoothing operation is not a low pass filter since a curved mesh can be reduced to a single point over multiple iterations. Therefore, all frequencies are removed during the smoothing process.

Algorithm 4.2 New iteration scheme based on Taubin's method.

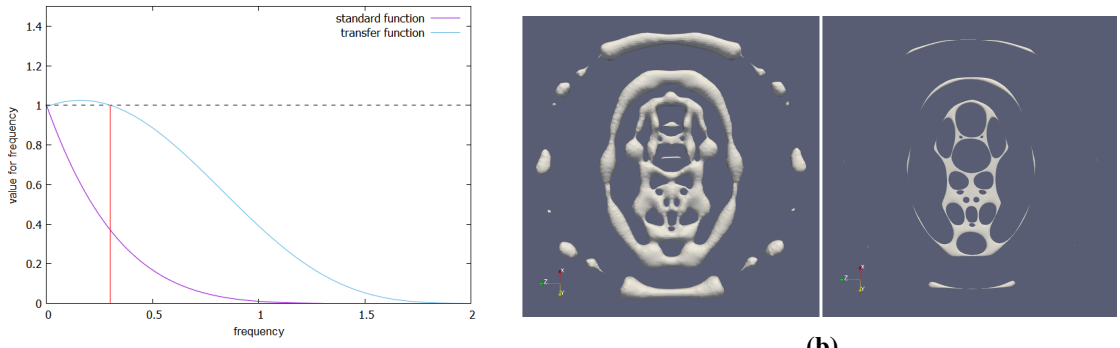
procedure ITERATION(M, C, σ, λ, μ) // μ and λ are the scale factors as defined in [Tau95].
  SMOOTH(M, C, λ)
  SMOOTH(M, C, μ)
  CALCULATEVOLUME(M, C)
  CORRECTVOLUME(M, C, σ)
end procedure

The new scheme for one iteration can be seen in algorithm 4.2.
By choosing k_P[15] and λ the following equation can be used to determine μ [Tau99]

$$k_P = \frac{1}{\mu} + \frac{1}{\lambda} > 0 \tag{4.8}$$

With Taubin's method no shrinkage effect should be present, see figure 4.14b. The smoothing process can therefore be done in an more controllable matter with less influence on the volume correction. Further, in [Tau99] it can be seen that the same parameters set can be used to achieve good results for different meshs. If $\mu = -\lambda$ is chosen the smoothing scheme is called bilaplacian smoothing and can additionally reduce the creation of low-frequency surface waves [OBB01b]. To create the results shown in the context of this work values λ and μ are based on the ones used in [Tau99] and [Tau95] and yield $\mu \sim (-\lambda)$.
In figure 4.15 results of Taubin's method along with volume correction can be seen. However, to achieve a faster reduction of the volume difference an additional factor σ is still needed. Yet, it can be set higher than it was the case with the approaches tested before. This leads to an overall faster and more stable volume reduction. Further, the same smoothing parameters can be used independently of σ while achieving similar results, see figure 4.15a. Yet, if the factor is chosen too high shifts created during volume correction introduce peaks on the mesh. This again leads to an overall increasing mean curvature and volume difference in adjacent cells. See figure 4.4b for a depiction of this behavior. The mesh is flattened by smoothing and over multiple iterations this can lead to oscillations and an even higher volume difference, see also figure 4.15a. Up to this end no method was found which can define the maximum factor for a set of smoothing parameters in advance. Hence, finding the highest possible factor σ still demands multiple test runs.
To sum up, smoothing is needed in order to further reduce the mean curvature and volume deviation. For different cell arrangements the shrinking behavior of Laplacian smoothing can lead to instability which cannot be corrected by choosing an arbitrary high factor σ. Using Laplacian smoothing therefore necessitates the appropriate choice of a weight λ along with a factor σ for each mesh individually. Yet, finding the right combination of these parameters is difficult with respect to the runtime of one iteration. In order to avoid the choice of an additional weight which strongly depends on the mesh geometry, further reduce the volume difference and mean curvature Taubin's method [Tau95] is used. It leads to better results in terms of a faster and stable mean curvature and volume difference reduction. Further, the same parameter sets as defined in [Tau95] and [Tau99] can be used to yield good results on different mesh geometries.
However, other problems beside mesh shrinking are not solved by using Taubin's method. One of these is the possible creation of vertex clusters as it was explained in section 3.5. In literature

[15] In [Tau99] it is mentioned that a value between 0.01 to 0.1 leads to good results.



(a) (b)

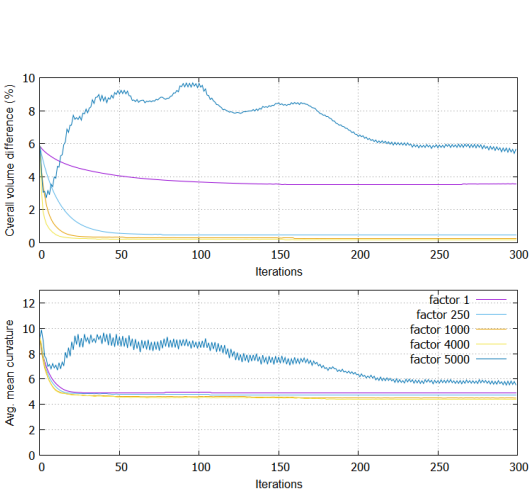
Figure 4.14: Comparison of Taubin smoothing and Laplacian smoothing:
(**a**) For multiple iterations Laplacian smoothing tends to remove all frequencies (standard function). Using the transfer function defined by Taubin [Tau95] all frequencies below k_p (red line) are kept while the others are removed over time.
(**b**) Comparison of Taubin (left, $\mu = -0.331$ and $\lambda = 0.33$) and Laplacian (right, $\sigma = 1$) smoothing. On the left side it can be seen that even after multiple iterations the mesh does not shrink significantly. However, this is not the case for Laplacian smoothing.

further extension exist that address such problems by adding additional weighting schemes. One of these is, e.g., presented by Ohtake et al. [OBB01b] and reduces the irregularity of the surface mesh by having more uniformly distributed vertices. Additionally, they introduced a method to enhance crease-like structures which are normally flattened by other smoothing schemes [OBB01b]. Due to the stable results achieved on the test dataset using Taubin's method other extensions have not yet been implemented.

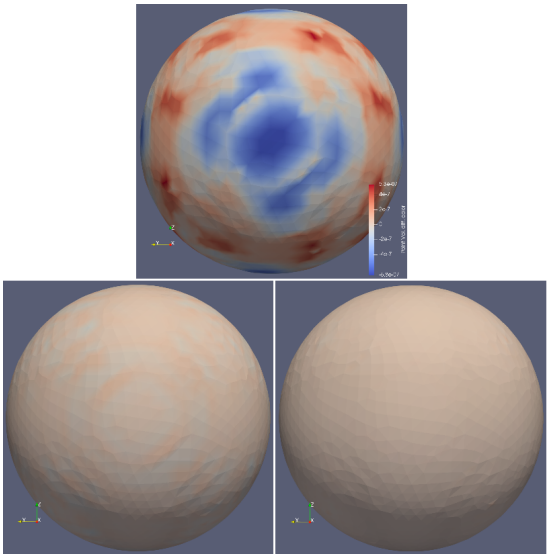
Even though the approach created in this section yields good results for the test mesh further improvements can be done. By adding mesh refinement (see section 3.6) a better adaption of the mesh to the VOF field and thus an even lower volume difference and mean curvature is possible. A method of combining refinement with the approach presented above is explained in the next section.

4.3 Combination with refinement

The approach defined in the last section showed the desired behavior with respect to volume difference and mean curvature reduction on the test mesh. Yet, it is executed on the base mesh created by the MC algorithm. Like mentioned in section 4.1 this allows a faster volume correction since bigger triangles have a higher impact on the volume difference in cells, see formula 3.75. However, inaccuracies are introduced especially in curved regions and the real interface is approximated only coarsely, see section 3.6. Thus, the achieved mesh mean curvature and final volume deviation in cells are still relatively high. Therefore, by using the refinement technique described in section 3.6 further improvements can be achieved:



(a)



(b)

Figure 4.15: Result of Taubin's method along with volume correction:
(**a**) The combined approach can yield a lower volume difference and curvature as it was the case with a weighted Laplacian smoothing (see figure 4.13). The approach is stable even for a higher volume correction factor σ and only slight variations can occur. Yet, if the factor is chosen too high the shifts created by the volume correction lead to an increasing volume difference and mean curvature. Oscillations are created since smoothing reduces the peaks created by the volume correction before new peaks are introduced. Here, the parameter values $\lambda = 0.330$ and $\mu = -0.331$ are used [Tau95].
(**b**) It can be seen that for an increasing factor σ (1 top, 250 bottom left, 4000 bottom right) local volume differences are getting lower (less color deviations). The final result is therefore not only due to the effect of smoothing but is the result of a combined approach with volume correction.

1. Like it is mentioned in section 3.6 refinement yields a mesh with more vertices that can be shifted in order to fit the underlying VOF field. This results in a mesh that is visually more appealing since the directions of adjacent mesh normals deviate less. Further, a curved flow of the VOF field can be approximated better with more triangles. It leads to a lower mean curvature since peaks on the mesh can be reduced.

2. According to formula 3.56 smaller triangles contribute less to the volume inside a cell. Therefore, by shifting these triangles a more precise depiction of the volume inside cells can be achieved.

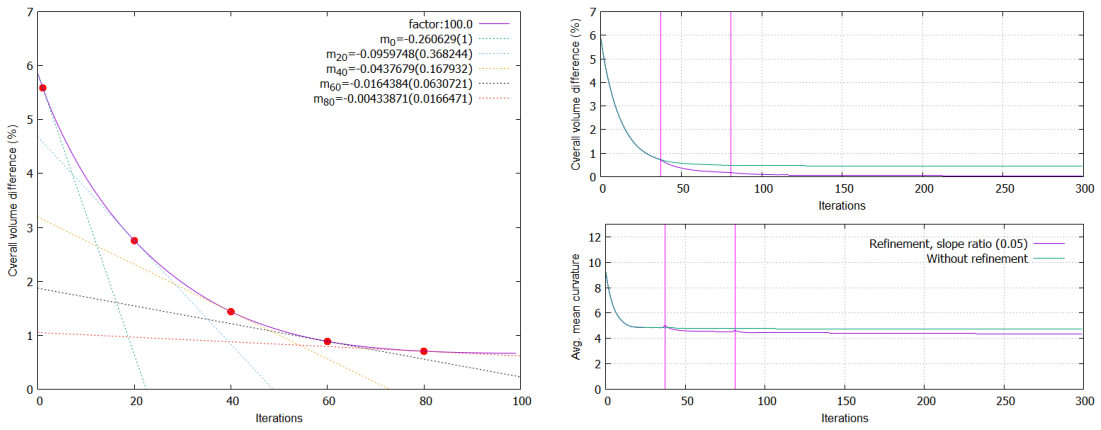
Ideally, the mesh should only refined if a further reduction of the volume difference can not be achieved using the current mesh. This indicates that the coarse mesh structure cannot not be changed using the current vertex resolution and distribution of vertices. Yet, choosing a fixed number of iterations after which refinement is done is difficult since for an arbitrary mesh the achieved result can change with respect to the given parameters. A better approach is thus to refine the mesh if a

Algorithm 4.3 New iteration scheme with added refinement.

```
procedure ITERATION(M, C, σ, λ, μ, γ, R)      // γ = threshold, R = refinement cycles.
    REFINE(M, C, γ, R)                        // Check threshold and refine if possible.
    SMOOTH(M, C, λ)
    SMOOTH(M, C, μ)
    CALCULATEVOLUME(M, C)
    CORRECTVOLUME(M, C, σ)
end procedure
```

certain condition is met. This can be, e.g., a sudden increase of the volume difference if the best possible adaption of the VOF field has already been achieved. However, such an increase may never happen in some cases since the volume difference converges to a stable value. Refinement with respect to a certain change rate of the volume difference demands the definition a of fixed threshold. Choosing such a threshold can be difficult because it is not known when such a change rate is low enough so that only slight changes on the mesh occur. Further, it must be chosen for each mesh separately since these have independent cell related properties such as the cell dimensions. Another method is thus to use a cell independent measurement depending on the flow of the volume difference curve itself since it should behave in a similar matter for all datasets. Refinement can therefore be done if the curve is flattened and the remaining change in volume difference is low. However, an investigation of the change rate of tangent slopes only depicts the flow of the curve at a specific iteration and the value can fall below an inappropriate chosen threshold already at the beginning. This is the case if, e.g., the curve already starts with a low slope. See figure 4.5a for an example of a curve that could make the choice of such a threshold difficult. A better idea is thus to define a particular ratio γ of the tangent slope at the beginning m_0 to the tangent slope m_x at which refinement should be done, hence $\gamma = \frac{m_x}{m_0}$. It can be chosen independently of the mesh and relates to the flatness of the curve at the current iteration in comparison to the beginning. Choosing an appropriate γ thus leads to refinement if the curve has been flattened significantly and further iterations change the mesh only slightly. Yet, an inappropriate choice of γ could lead to early refinement on a relatively steep curve. However, this can be compensated to some extend by further iterations on higher refinement depths. In figure 4.16a the tangent curves at different iteration cycles for the tested sphere dataset are shown. It can be seen that the tangent slope of the curve as well as the ratios change rapidly over multiple iterations. Here, the absolute ratio value ($\gamma = \left|\frac{m_x}{m_0}\right|$) is used since for each curve the volume difference should decrease over time and refinement should be done before effects which lead to further deviations (increasing volume difference) can occur. The chosen value for testing ($\gamma = 0.05$) is based on the curve and ratios seen in figure 4.16a. Refinement is done at the beginning of an iteration as soon as the ratio of the tangents slopes is below the chosen threshold γ. Each triangle is refined using the approach described in section 3.6. See algorithm 4.3 for the resulting iteration scheme.

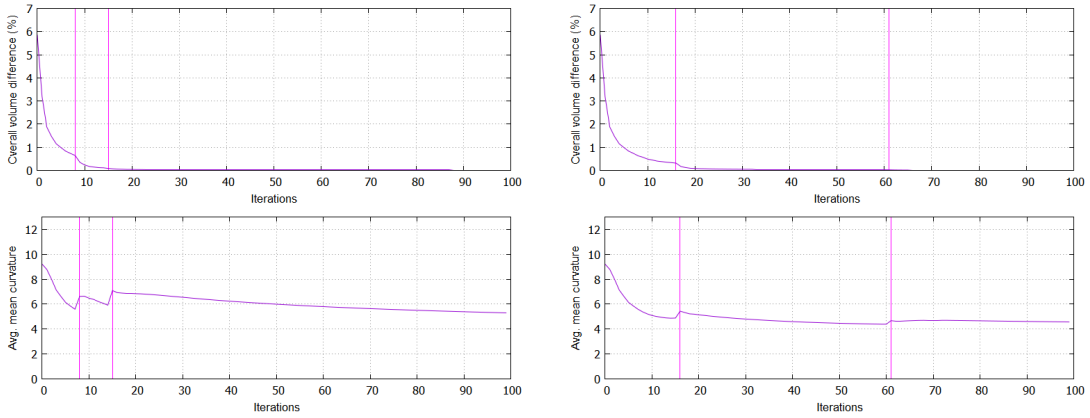
In figure 4.16b the result of different refinement cycles on the sphere dataset ($51x51x51$ nodes) can be seen. The iterations at which refinement was done are marked with vertical lines. It is shown that the volume difference and mean curvature can be decreased further than it was the case without refinement. For the volume difference this was the case for all tested factors σ. Yet, a increase of the mean curvature after refinement can be observed. This effect is based on formula 3.75 where the Voronoi area A_v (see formula 3.50) is used. Since smaller triangles lead to a smaller Voronoi area the mean curvature values at vertices are higher. This effect is especially strong if the overall mean



(a) **(b)**

Figure 4.16: Result of added refinement using a slope ratio:
(**a**) Tangents of the volume difference curve at different iterations. The absolute ratio γ of the current slope m_x to the starting slope m_0 is shown in brackets. It can be seen that the slope approaches zero over time and so does the ratio.
(**b**) After the slope ratio ($\gamma = \frac{m_x}{m_0}$) of the volume difference curve is below 0.05 refinement is done (at vertical lines). The new tangent slope after refinement is then selected as m_0. Here, both the volume difference and mean curvature can be decreased further than without refinement ($\sigma = 250$, $\lambda = 0.330$, $\mu = -0.331$).

curvature is still relatively high (slope of the curve is not low enough) and the mesh thus has many peaks (areas with high local mean curvature) when refinement is done, see figure 4.17a. The former can be the case if, e.g., a high factor σ or threshold γ has been chosen or the volume difference is reduced faster than the mean curvature. Like mentioned in section 3.5 connections on a finer mesh between vertices are shorter and smoothing is done more locally since distant positional information is distributed over a longer period of time. The reduction speed of the mean curvature and volume difference is thus even slower on a refined mesh. It takes a longer period of time until such peaks of the mean curvature curve are reduced to an eventually lower value than without refinement.
To avoid a sudden increase of the mean curvature curve it's calculation could be changed by using a formula that does not rely on triangle areas for measurement. However, in the context of this work such an approach could not been found. In order to counterfeit this issue one idea is therefore to lower the threshold γ so that refinement is done after a higher number of iterations (volume difference curve is flatter). This method does not include information about the slope of the mean curvature curve. Yet, it can be seen that it's flow is similar to the volume difference curve with a higher slope at the beginning and flattening over time (see, e.g., figure 4.16b). For a completely flat curve lower curvature values are not achieved in succeeding iterations. Therefore, the increase of the overall mean curvature after refinement should be minimal. An additional slope ratio check for the mean curvature is thus chosen to reduce the jump of mean curvature values (γ is checked for both curves). Refinement is triggered only if the slope ratios of both curves are low. In figure 4.17b it is shown that the jump of the mean curvature curve can be decreased as expected. Yet, it can take more iteration until refinement is triggered since the volume difference and mean curvature don't always change at the same speed for different meshs and parameters.
Depending on the triangle position and orientation refinement can lead to vertices located outside



(a) **(b)**

Figure 4.17: Result of different thresholds and check for mean curvature:
(**a**) If the threshold and σ are chosen too high refinement is done (at vertical lines) while the curve of the mean curvature still has a high slope. Areas with high curvature are still contained on the mesh. Due to formula 3.56 the value of the mean curvature at these areas increases further since the Voronoi area (formula 3.50) for every triangle is lower. The mean curvature decreases again over time. Yet, on a refined mesh this takes a higher number of iterations since smoothing acts more locally and distant information takes longer to distribute. $\sigma = 4000$, $\lambda = 0.330$, $\mu = -0.331$, $\gamma = 0.05$.
(**b**) Choosing an additional slope ratio check can yield a smaller jump of the mean curvature when the mesh is refined. Less iterations are necessary to compensate for the jump. See (**a**) for the chosen parameter set.

interface cells, see figure 4.18c. This can happen if vertices are close to cell borders. Triangles can then intersect empty cells yielding edge centers located outside of interface cells. To solve this issue one idea is to shift vertices shift back inside the nearest interface cell. Yet, this alters the volume in adjacent cells and creates a bumpier surface (see section 4.1). Another method is to prevent refinement of the respective triangle if it intersects a non-interface cell. However, this affects the distribution of vertices by having areas with bigger and smaller triangles on the mesh. Adding the affected non-interface cells to volume correction so that vertices are shifted back into interface cells could further lead to a higher overall runtime. Thus, vertices outside interface cells are not considered for volume correction so far. This leads to a mesh with lower mean curvature and vertices can eventually be shifted into interface cells by smoothing.
In figure 4.18 the mesh for the sphere dataset ($51x51x51$ nodes) created with two refinement cycles can be seen. The overall volume difference is lower than before and the final mesh is close to the approximation created using PLIC. However, in table 4.2 it can be seen that the number of triangles increases by a factor of four with each refinement cycle. This leads to an overall higher computation time and is especially noticeable if the mesh created by the MC algorithm already consists of a high number of triangles. Therefore, the number of iterations should be kept low by choosing appropriate parameters to reduce the volume difference and mean curvature on a coarse mesh as much as possible. Like mentioned in section 3.5 tests have shown that more than two refinement

Refinement depth	#Triangles of mesh	Duration (iteration in s)
0	5864	0.473
1	23456	0.638
2	93824	1.119
3	375296	2.564

Table 4.2: More refinement cycles result in a higher iteration runtime due to more triangles (see algorithm 4.3 and section 4.1). Here the average duration of an iteration using the test implementation [17] on the sphere dataset ($51x51x51$ nodes) is shown.

cycles do not create a better visual result[16] but increase the overall computation time significantly. In the context of this work the results are therefore created using a maximum of two refinement cycles.

Like mentioned in section 4.1 the algorithm can be stopped after a defined number of iterations. However, the values of the volume difference and mean curvature after a fixed number of iterations are not known in advance. Another possibility is thus to stop if the maximal defined refinement depth is exceeded. Yet, for the results shown in the next sections a fixed number of iterations was used to have a better comparability of the results.

To sum up, by adding refinement a further reduction of the volume difference and mean curvature can be achieved. The mesh is refined if the slope ratios of both the volume difference and average mean curvature curves are below a threshold ratio γ. Yet, γ must be chosen so that both curves are relatively flat before refinement is done. This prevents a strong peak in the mesh's curvature plot and high local curvatures values at vertices that are due to the creation of smaller triangle areas. Up to this end an approach to automatically find the best fitting slope ratio γ for a mesh could not be found. The reduction speed of the volume difference and the mean curvature decreases with a higher number of refinement cycles. Additionally, the runtime of an iteration is increasing significantly which is especially the case for meshs that already have a high number of triangles before refinement.

In the next section the final combined approach (volume correction, smoothing and refinement) is tested on different meshs and it's overall applicability is analyzed.

[16] At some point the reduction of the volume difference or mean curvature does not affect the final visual result anymore.
[17] It can be found at `https://github.com/MaBa90/VOFVis`.

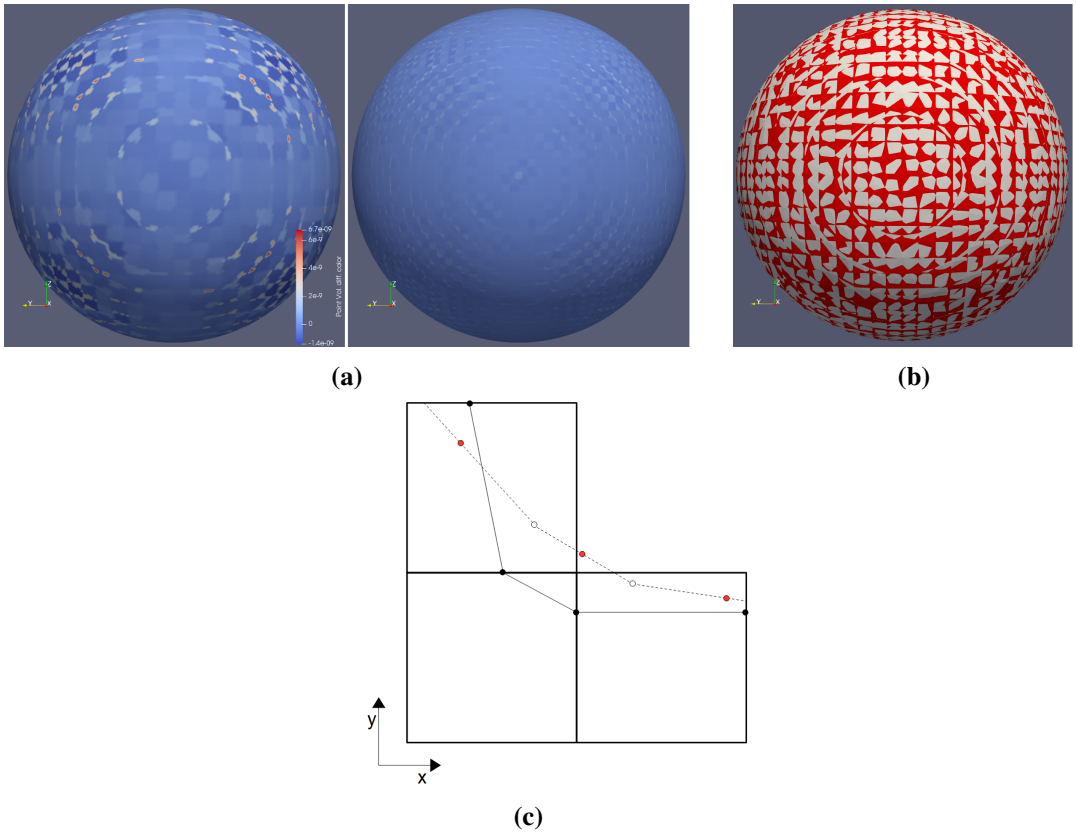


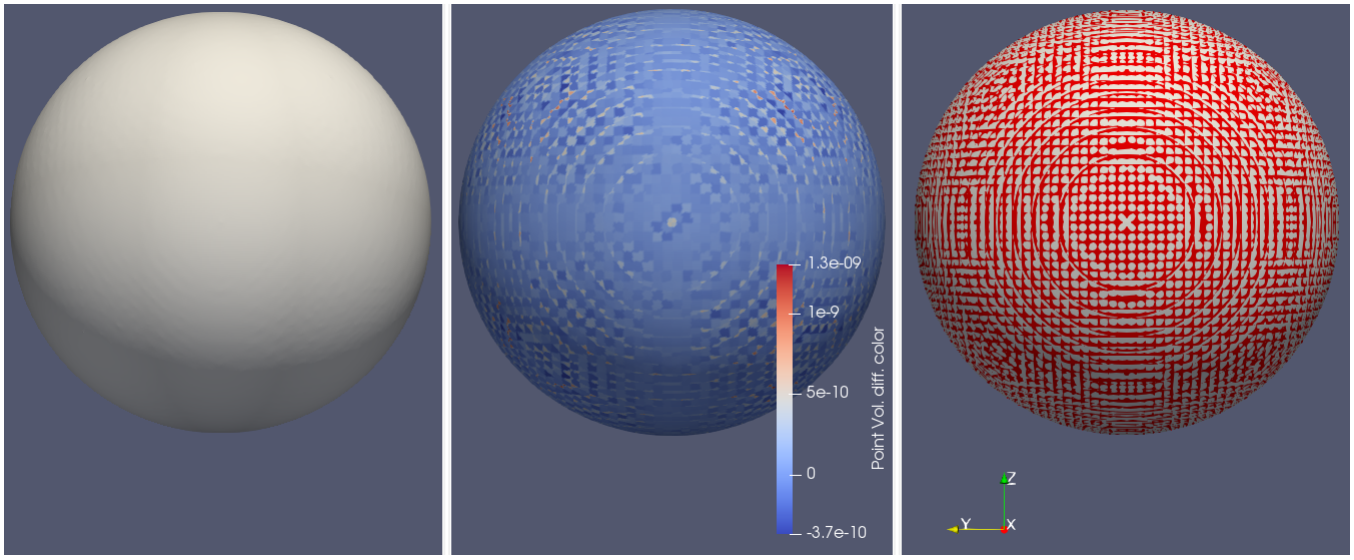
Figure 4.18: Result of added refinement and comparison with PLIC:
(**a**) Volume differences without (left side) and with (right side) refinement on a sphere ($51x51x51$ nodes). In both cases a slope ratio γ of 0.05, a factor σ of 250 and 300 iterations were used. It can be seen that using refinement the overall volume difference is decreased even further (less local color deviations). $\lambda = 0.33$ and $\mu = -0.331$ as defined in [Tau95].
(**b**) The overall result is close to the approximation created using PLIC (red). The deviation of both meshes is due to the linear nature of the planes elements used by PLIC. These cannot depict curved regions inside cells accurately.
(**c**) Example of a vertex created outside of interface cells in 2D. After a few iterations the mesh created by the MC (black points and continuous line) is shifted. The new mesh is then depicted by the white points and the dashed line. Refinement adds vertices (red points) in the middle of edges. One of these vertices is located outside of interface cells. Further, the mesh has been moved out of the bottom left cell due to, e.g., smoothing.

5 Final results

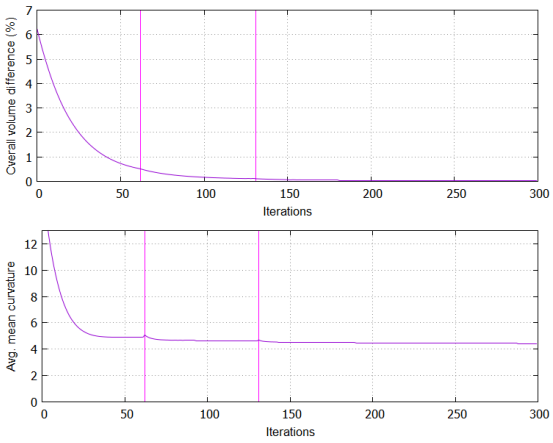
In this section the general applicability of the final scheme (see algorithm 4.3) is investigated. For that datasets with different structures in terms of cell connectivity, node resolution and contained cell clusters (e.g., small droplets) are used to test the approach under various conditions. Like mentioned in the preceding section a fixed number of iteration is used for testing. This enables a better comparison of the results since depending on the used stopping criterion the algorithm could quit after an unknown number of iterations. Up to this point finding the best parameter combination (σ, λ, μ and γ) with respect to reduction speed and final mesh deviation must be done manually using multiple tests. Yet, for testing only σ is changed with respect to the underlying cell dimensions to increase the reduction speed of the volume difference. Other parameters are set equal to those used in figure 4.18c. The idea is that an overall qualitative good result (low volume difference and mean curvature) on different meshs should be achieved without the additional effort of finding necessary parameter sets.
In figure 5.1a the result of another sphere dataset with $\sigma = 500$ can be seen. It was chosen because it's geometry is similar to the test dataset used in the last sections while having a higher node resolution ($101x101x101$) with smaller and differently arranged cells. Thus, if the approach works as expected the result should be close to the one achieved on the test mesh, see figure 4.18c. It can be seen that the volume difference is reduced to a value close to zero. The mean curvature decreases and approaches the analytical value of $\kappa_H = 4.0$. As mentioned in section 4.3 smaller triangles can yield a sudden increase of the mean curvature after refinement is done. Yet, for this mesh the effect is neglectable and the peaks on the curve are removed after a few iterations. The final mesh is close to the approximation created by PLIC [You82]. Again, the remaining deviation of the mesh is due to the linear plane segments that cannot depict curved regions accurately. In figure 5.1c vertex clusters (dense areas) on the mesh can be seen. Taubin's method [Tau95] is based on Laplacian smoothing. As described in section 3.5 it tends to create such clusters by shifting vertices together. This effect leads to small bumps on the final mesh and cannot completely be removed even with a higher number of iterations or stronger smoothing factors λ and μ. For that a different smoothing approach (see preceding section) has to be used to move vertices with respect to the current distribution on the surface.
The next mesh is based on one timestep of a simulated collision of droplets (see section 3.1) in a grid with a resolution of $257x257x257$ nodes. This particular dataset was chosen because it contains flat as well as curved regions. Therefore, the approach can be tested with different flows of the VOF field. For testing σ was set to 1500 and the other parameters were set to the values used in figure 5.1. As it can be seen in figure 5.2b the volume difference inside cells is lowered as expected approaching a value closer to zero. Yet, in section 3.7 it was already discussed that due to a higher node resolution the reduction of the volume difference and mean curvature needs more iterations. Here, this is still the case even though a higher factor σ was chosen. The mesh is again close to the approximation created using PLIC, see figure 5.2a. Plane elements are intersected by the resulting mesh which shows that it was adapted correctly to the underlying VOF field. However, the graph of the average mean curvature is increasing even though the overall mesh structure is

getting flatter ($\kappa_H \rightarrow 0$). This can be explained by mesh regions that have a higher local mean curvature. Due to the chosen value of σ the volume correction shifts vertices for a long distance in some cells. This results in a change of the volume in adjacent cells which has then to be corrected with an even higher shift. The effect is thus similar to the one described in figure 4.4b. Further, vertices are stopped at the border to non-interface cells (see section 4.1) and some are stuck since any following operation would move them outside the cell. While the rest of the mesh is getting flatter the peak at these cell boundaries is hence getting sharper leading to a higher mean curvature, see figure 5.3b. Moreover, a precise calculation of the cell gradient cannot always be done for the given node resolution. This can be seen at the border of the inner region where high deviations of the plane element directions occur, see figure 5.3a. These inaccuracies lead to a bumpier mesh surface because shift directions are based on interpolated gradient at nodes between cells. Even though these effects affect the overall visual result only slightly they lead to the observed increase of the mean curvature. Enabling vertices to leave cells under certain conditions, another curvature related smoothing technique as well as a higher node resolution could solve these issues.
Like mentioned in section 4.1 tests have shown that some cells have a very low or high VOF value and therefore only a small MC surface patch is created. One step of the volume correction or smoothing operation can thus lead to a shift of these patches out of the cell. Further, the MC algorithm cannot create triangles for certain cell arrangements. This is the case if the flow of the VOF field can be shaped arbitrarily, see section 3.2. In figure 5.3c the surface mesh is depicted with empty interface cells. Like discussed in section 4.1 empty interface cells are not added to the overall volume difference calculation. Moreover, cells with very low or high VOF value are most certainly created by numerical inaccuracies and do not contribute to an accurate depiction of the real interface. In most cases the mesh resides close to these cells over multiple iterations and additional empty interface cells are not noticeable in the final result. Again, a higher node resolution could solve this issue yielding a more precise representation of the VOF field.
Another mesh was created based on the third timestep of a simulated collision of droplets (see section 3.1). Again, the grid has a resolution of $257x257x257$ nodes. This particular dataset was chosen because it contains not only flat and curved regions but also unconnected cell clusters. It is used to test the behavior with respect to different connection properties of cells. σ was set to 2000 and the other parameters to the same values that were used for the other tested datasets. With respect to the creation of peaks shown in figure 5.3b and in contrast to the idea presented in section 4.1 vertices are now able to leave interface cells. The idea is that this way the number of artificial peaks on the mesh surface is reduced since smoothing can now act freely on the vertices.
In figure 5.4a the resulting mesh along with the volume difference and a comparison to PLIC can be seen. The overall volume difference on the mesh is lowered and the coarse mesh structure is again close to the approximation achieved using PLIC. Yet, areas with a higher volume difference are visible. Like it was observed for the last dataset numerical inaccuracies or low node resolutions can yield high change rates of VOF values and thus gradients between cells. Therefore, only a coarse approximation of the VOF field is achieved and the mesh volume deviates inside the affected cells. See figure 5.6b for an example of such a region.
The graph in figure 5.4b shows the expected reduction of the volume difference. Yet, the value of the average mean curvature is increasing and oscillations can be seen. It was shown before that the formation of vertex clusters by smoothing and numerical inaccuracies can further lead to a bumpier surface (see section 3.5) with higher mean curvature. Yet, the observed effect can only be described by regions with very high curvature values. In this third dataset small cell clusters do occur. These are represented by only few closely connected triangles. As it can be seen in figure 5.5a the non-shrinking behavior of Taubin's method cannot be achieved for some of these clusters.

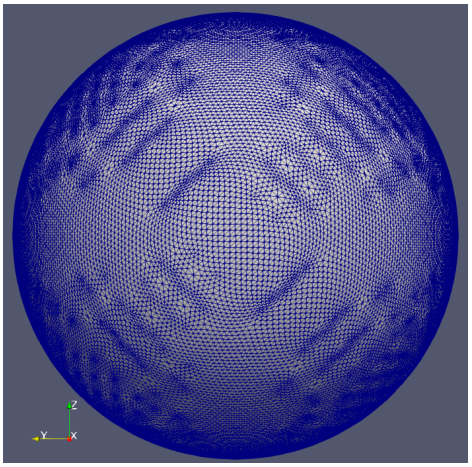
Due to peaks and short distances between opposite vertices the shifts created by the shrinkage step lead to a strong impact on the mesh's evolution. Over multiple iterations this results in a flattened disc like mesh region with smaller triangles and thus significantly higher curvature values (see formula 3.56) at the borders. Due to the permanent alternation of volume correction and smoothing high curvature values can then oscillate in the graph. Volume correction can further move the whole mesh cluster. This is te case, e.g., if the nodes are shifted in the same direction as done by the shrinkage step of smoothing. See figure 5.5b for another example of such behavior in which a tube like structure is flattened and moved. The observed oscillations can further trigger refinement even if the mean curvature is increasing. Yet, over multiple iterations the curvature is decreasing again. Areas with high curvature are then sampled with more vertices and are thus flattened over time.
Like mentioned above for some cells the MC algorithm cannot created triangles and the mesh can easily leave interface cells which have a very high or low VOF value. As it can be seen on the left side of figure 5.3c the created mesh stays close to these cells which was also the case for the second tested dataset (see above).
The presented approach works as expected for simple cases. On datasets with higher complexity (triangle clusters, inaccuracies, flat and curved regions) problems arise that can affect the visual and plotted results. However, in figure 5.4a it can be seen that the effect on the overall visual result is low since the mentioned problems are mostly bound to local properties of the dataset. Further, the overall mesh structure was always close to the approximation created with PLIC. Yet, the plot can deviate significantly as it can seen in figure 5.4b.
To solve some of the issues mentioned above and prevent numerical inaccuracies a higher mesh resolution can be used. It leads to a better depiction of the VOF field and thus less deviation of the mesh. Also, having more cells and therefore a finer MC mesh with more vertices could prevent the shrinkage of small vertex clusters. However, a higher resolution results in an even slower decrease of the volume difference and mean curvature. With respect to the duration of a single iteration this is not practicable. Moreover, for some problem cases a more sophisticated solution is needed. This is discussed in the next section.



(a)

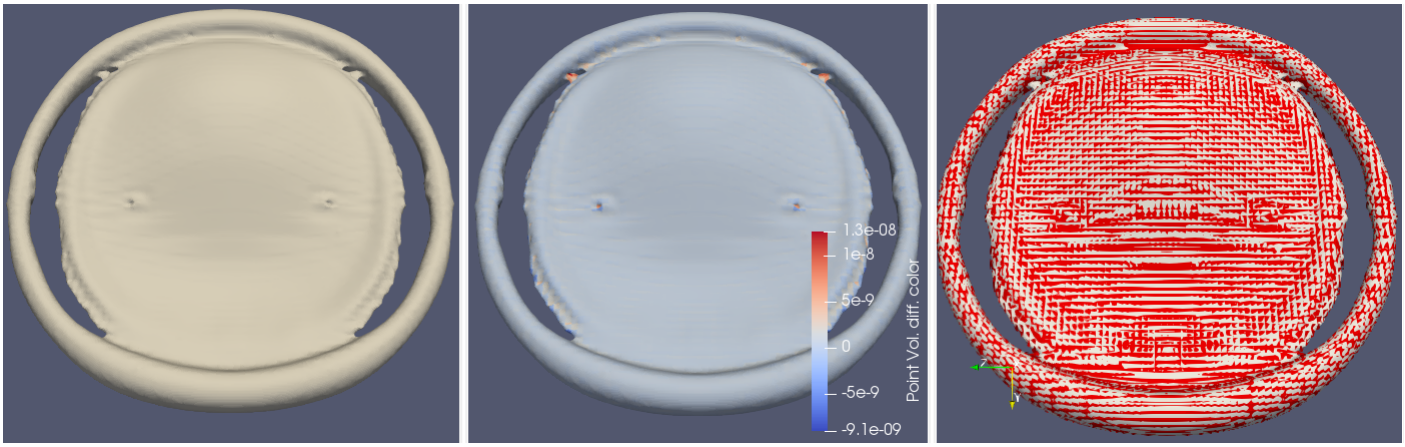


(b)

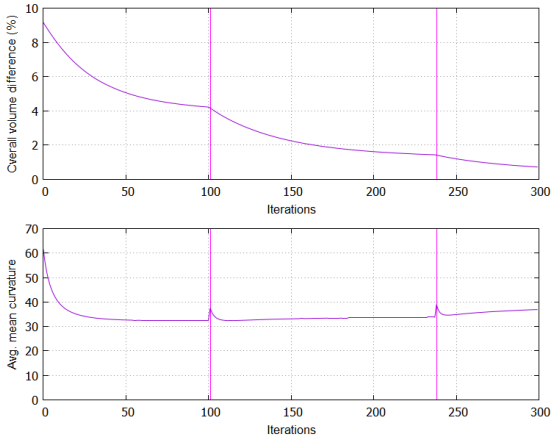


(c)

Figure 5.1: Results of the final scheme applied on a sphere with high node resolution:
(**a**) On the left the resulting mesh for a sphere ($101x101x101$ nodes) can be seen. Parameters were chosen equally to figure 4.18a but with $\sigma = 500$ in order to reduce the volume difference faster. The volume deviations within the cells are close to zero (center image) and the resulting mesh is close to the approximation created using PLIC [You82] (red mesh, right).
(**b**) For this mesh the graph of the volume difference and mean curvature show the expected behavior. The volume difference is reduced further with each refinement cycle (vertical lines) and the overall mean curvature approaches a value close to the analytical one ($\kappa_H = 4.0$).
(**c**) Here, vertex clusters (dense areas) as described in section 3.5 can be seen. This results in the mesh being not completely smooth. A possible solution for this issue is proposed in section 6.

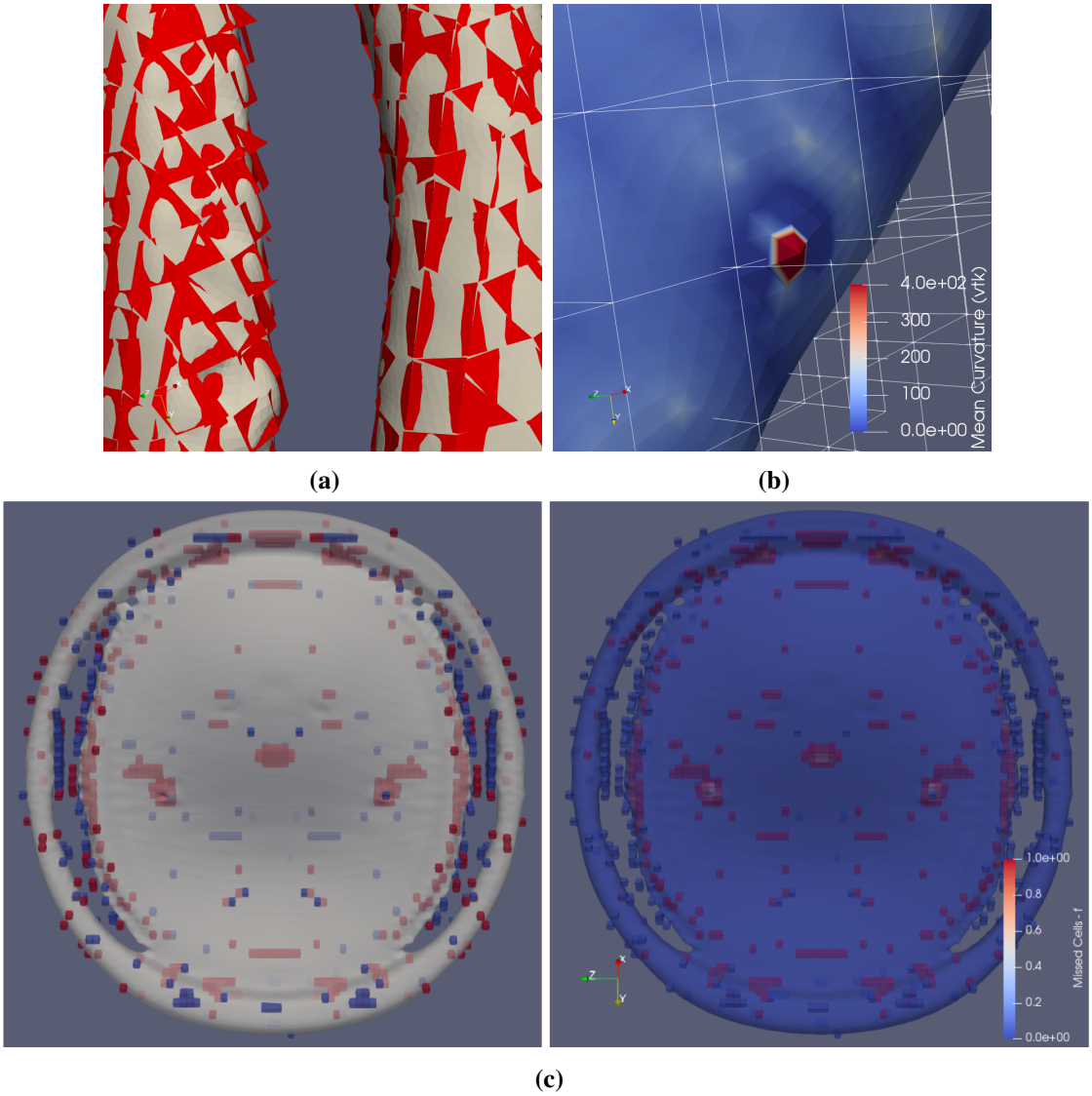


(a)



(b)

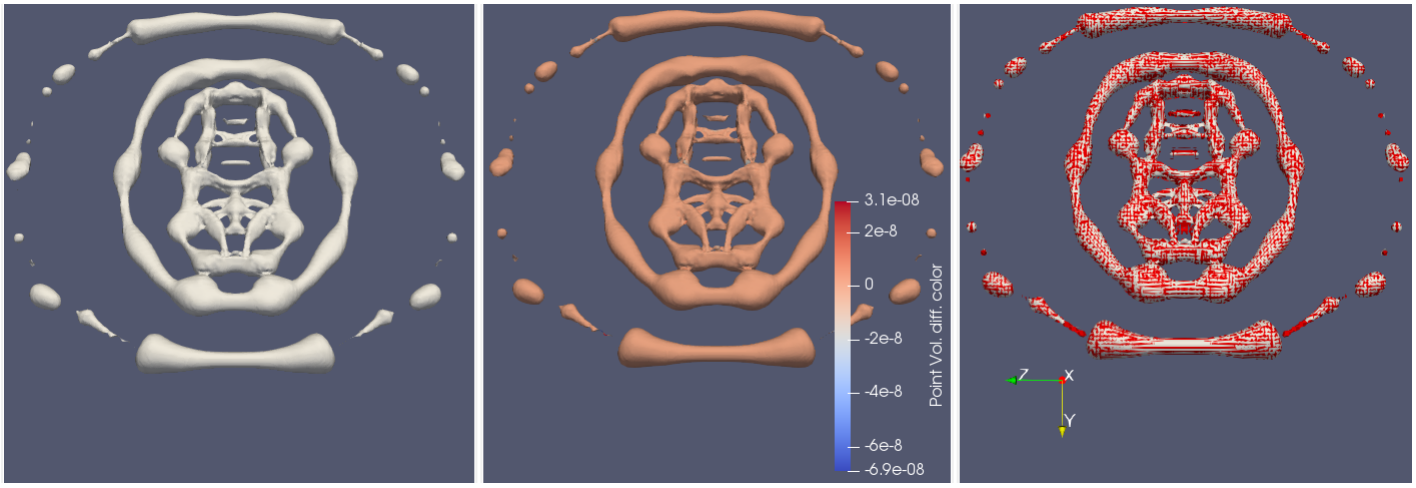
Figure 5.2: Results of the final scheme applied on the second droplet collision dataset:
(**a**) On the left the resulting mesh for the second timestep of a droplet collision ($257x257x257$ nodes) can be seen. Parameters were chosen equally to figure 5.1 but with $\sigma = 1500$ in order to reduce the volume difference faster. The overall volume deviation within the cells is close to zero (center) and the resulting mesh is close to the approximation created using PLIC (red mesh, right).
(**b**) The graph of the volume difference shows the expected reduction. Yet, the mean curvature increases again due to local peaks in the mesh (see figure 5.3b). They are created by the high σ value and therefore alternating volume corrections in adjacent cells (see figure 4.4b). Additionally, a bumpier surface is created by numerical inaccuracies in the dataset, see figure 5.3a.



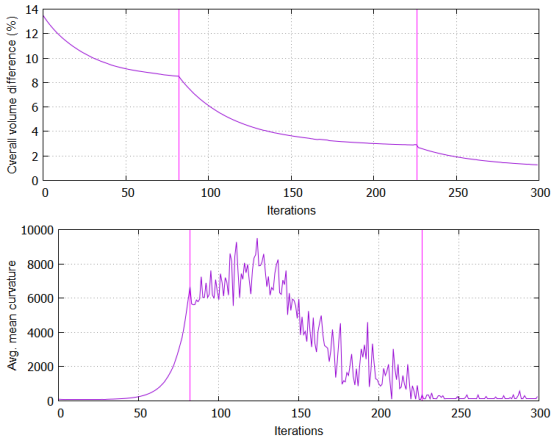
(a) **(b)**

(c)

Figure 5.3: Further investigation of the result achieved on the tested second dataset:
(**a**) Due to a low mesh resolution the flow of the VOF field cannot be depicted precisely. This can be seen by inconsistent directions of PLIC planes (red surface) over cell boundaries. It leads to a bumpier surface when the final scheme is used (yellow mesh).
(**b**) Peaks and thus areas with high mean curvature can be created due to a high factor σ. This is because vertices are stuck (red vertice) at cell boundaries if further corrections lead to a shift out of interface cells (white cubes). Here, only positive curvature values are shown.
(**c**) On the left cells missed by the MC algorithm are shown as red and cells which are emptied over multiple iterations as blue cubes. On the right the same cells are shown using their VOF value. It can be seen that the mesh is only moved out of cells with a very low or high VOF value. Since the mesh stays close to the nodes of these cells the overall visual result is not affected. Further, such cells are introduced most likely due to numerical inaccuracies in the dataset.

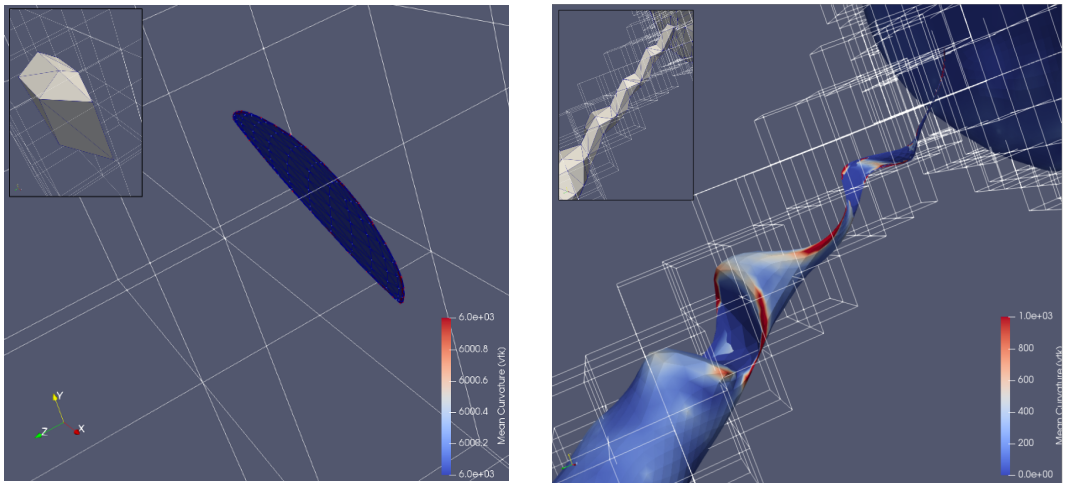


(a)



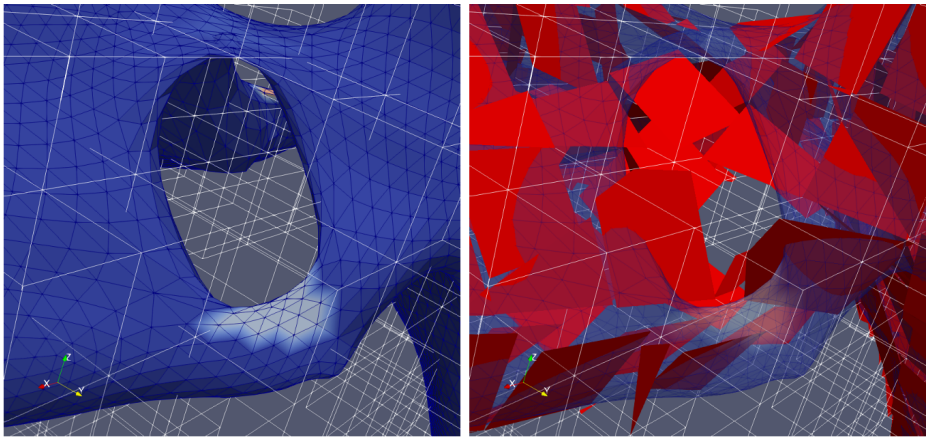
(b)

Figure 5.4: Results of the final scheme applied on the third droplet collision dataset:
(**a**) General results are close to the one achieved with the datasets tested in 5.1 and 5.2. The resulting mesh can be seen on the left. The overall volume differences (center) are close to zero. Yet, there are some areas with a high volume and mesh deviation, see figure 5.6a. On the right side it is shown that the result is close to the approximation created by PLIC. σ was chosen to 2000.
(**b**) The graph of the volume difference shows the expected behavior. Yet, the curvature shows a high increase and oscillations. This can only be explained by vertex clusters which are made of very small triangles. According to formula 3.56 these can lead to an increased curvature value, see figure 5.4a. Further, the threshold γ is hit due to the high slope change for the mean curvature and refinement is triggered.

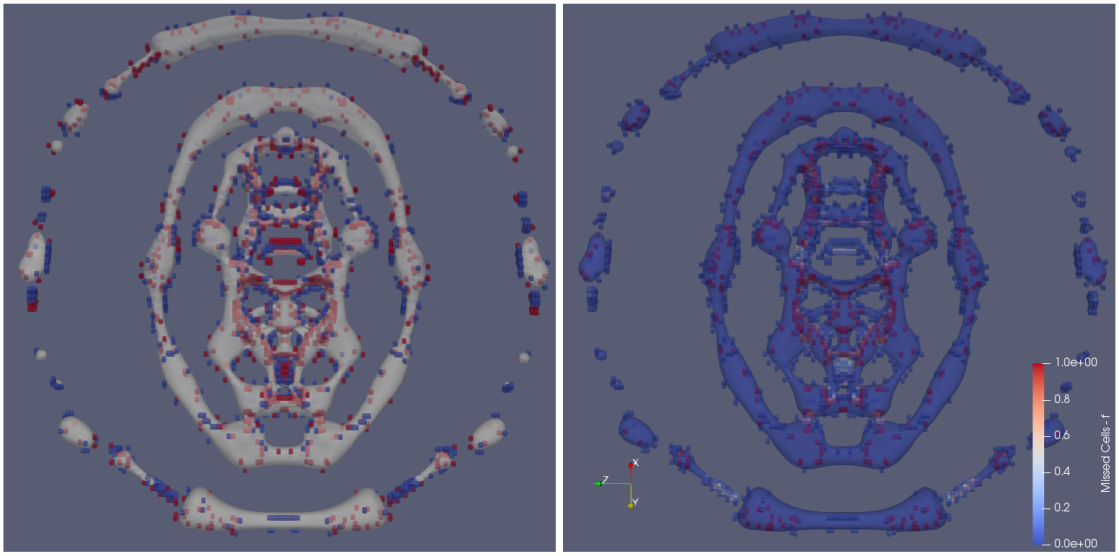


(a) **(b)**

Figure 5.5: Example of areas of high curvature on the third dataset:
(**a**) Vertices with a high mean curvature are shown in red. This effect is created by a shrinkage of small cell clusters. They consist of close opposing vertices (small image) for which the first step of Taubin's method has a higher impact. These clusters shrink over multiple steps which yields flat mesh artifacts. After refinement the smoothing only affects vertices over a small area and the shrinking is slower. Yet, the final position of these artifacts depends on the overall shift due to volume correction. In this case the resulting mesh is moved inside a single cell and no further volume correction is possible. Cells are depicted as white boxes. Color scaling is selected so that only high curvature values are marked.
(**b**) For narrow tube like structures a similar behavior can be observed. Opposing triangles are close together and curved areas are represented with only few triangles (small image) leading to peaks. Again, this arrangement results in a higher impact of the shrinkage step. The volume correction can increase this effect further and influence the final position of the mesh.



(a)



(b)

Figure 5.6: Example of an area with high volume difference and empty cells:
(**a**) Region with a higher volume difference. On the right side it can be seen that the area cannot be depicted accurately since the PLIC plane and thus the gradient directions deviate significantly. For the presented approach this leads to areas with a higher volume difference (bright region, left side). A higher mesh resolution could solve this issue.
(**b**) Again, the creation of empty interface cells can be observed. On the left side the cells skipped by the MC are shown in red. Cells that are emptied during execution are shown in blue. On the right side cells are colored according to their VOF value. Again, the mesh stays close to the nodes of these cells and the overall visual result is affected only slightly.

6 Future work

With respect to the results of the last section further work needs to be done in order to improve the applicability of the presented approach. In this section current issues as well as possible solutions are discussed.
Up to this end many iterations are needed in order to yield a significant and stable reduction of the mean curvature and volume difference. The current runtime of a single iteration and the impact of additional refinement (see table 4.2) make the approach difficult to use in most scientific scenarios. To reduce the overall runtime one possibility is to alter the volume calculation since it contributes significantly to the duration of an iteration (see table 4.1). Instead of using a direct calculation with intersection points and triangulation (see section 3.7) a less precise volume measurement could be used. For that a simplified representation of the mesh and cell interactions can be created by retrieving intersection points only at cell edges. This results in concave shaped intersection patterns on each cell side that can be triangulated using a simple scheme. With that the direction for volume correction could be estimated to reduce the volume difference in each cell. Yet, this does not allow a precise volume calculation and the resulting mesh will most likely deviate from the actual interface. Further, using a different iterative method than Newton (see formula 4.1) to shift the vertices could reduce the number of iterations needed. This should work for each cell independently to avoid the alternation of adjacent cell volumes (see figure 4.4b). As mentioned in section 4.1 another iterative method that provides such a property was not found in the context of this work. A more sophisticated solution based on schemes like WENO (see section 4.1) is thus needed. The speed of the volume correction also depends on the used implementation [1]. Using a different library could therefore decrease the overall computation time to some extend.
Like mentioned in section 4.1 the mesh can leave interface cells during execution. Shifting the mesh back into cells, e.g., by moving the closest vertices in direction of the cell center leads to deviations of the cell volume. With respect to algorithm 4.3 a shift after smoothing leads to bumps on the smoothed mesh and an increased mean curvature. If the shift is done after volume calculation the calculated volume in all neighboring cells is altered. Shifting after volume correction alters the mesh so that the new mesh structure deviates from the corrected one. A higher node resolution could provide a better depicting of the VOF field and reduce the number of empty interface cells. Yet, this increases the runtime of the algorithm and another method must therefore be found.
The refinement operation can create vertices outside interface cells. A shift of these vertices back into cells or along triangle edges creates peaks, increases the mean curvature and alters the volume of adjacent cells. One idea is thus to add non-interface cells to the volume correction which could lead to an automatic shift of vertices back into interface cells. Yet, more memory and volume corrections are necessary. Without further improvement of the implementation this can lead to a higher overall runtime.
As mentioned in section 4.2 the distribution of vertices must be improved in order to avoid the

[1] The test implementation can be found at `https://github.com/MaBa90/VOFVis`.

formation of dense areas (vertex clusters, see figure 5.1c) with similar properties (gradients, curvature, etc.) on the mesh. The effect of lighting operations (see section 3.5), smoothing and the accuracy of the volume correction is thus different for each of these areas. By implementing further extensions to smoothing (see section 4.2) a more uniform distribution of vertices can be achieved. This results in less surface bumps and hence a lower mesh curvature since positional information can be distributed more equally and faster over the whole mesh.
In figure 5.5a examples of triangle clusters that produces high curvature alternations are depicted. Over multiple iterations these clusters are reduced to flat areas. An adaption of the smoothing parameters to local properties of the mesh could prevent that behavior. This can be done, e.g., by setting λ with respect to the cell size of vertex clusters. Yet, the final scaling of the parameter must be determined for each cell cluster individually.
The condition to refine the mesh has to be made more reliable. Until now the absolute value of slope ratios is used. Refinement can therefore be triggered for different slope directions of the volume difference and mean curvature. However, using the relative value instead of the absolute one does not yield a reliable solution since in figure 5.4b strong oscillations of high curvature values can still trigger refinement. A different measurement must therefore be found that does account for such deviations on complex meshs. Additionally, the method for calculating the mean curvature should be changed in order to avoid the accumulation of high values in the first place. With the current approach different triangle sizes lead to an increasing average mean curvature even though the mesh is getting globally flatter, see figure 5.4b. A new approach could prevent such behavior and allow a more precise depiction of the mean curvature. One idea is to ignore vertices at which the mean curvature oscillates or exceeds a certain threshold. It could be determined using a history of local curvature values or mesh properties. Until then the results must be verified visually by changing the color scale on the mesh to ignore local high curvature values.
To this end the best parameter set (σ, λ, μ, γ) in terms of stability, speed and correctness needs to be chosen manually for each mesh. This implies multiple tests with different parameter sets. By analyzing the dataset using, e.g., machine learning an automatic approach could choose a combination or range of possible parameter values. However, so far it is not entirely clear which dataset-dependent properties need to be analyzed in order to create such an automatic approach.
In the next section a conclusion with respect to the content of this work and the final results of the last section is given.

7 Conclusion

In this work a new approach to visualize VOF datasets was presented. The goal was to use the MC algorithm to create the basic surface mesh (see section 3.2). This mesh should then be adapted to the underlying VOF field iteratively reducing the mean curvature and the overall volume difference inside cells.
In the first sections of this work necessary operations were defined. These are the adapted MC algorithm (section 3.2) for VOF datasets, VOF gradient calculation (section 3.3) and cell volume calculation (section 3.7) for volume correction, Laplacian smoothing (3.5) for mesh curvature reduction, curvature calculation (3.4) for verification and mesh refinement (3.6) for more accurate results. In following sections these operations were then combined (4) to yield the final iterative scheme.
It has been shown that the MC algorithm can be used in VOF datasets to create an isosurface representation of the interface. Also, by using the developed iterative scheme the local mean curvature and the cell volume difference can be reduced in most cases. Yet, numerical inaccuracies and the formation of vertex clusters can lead to empty interface cells and bumps on the surface. This can increase local mean curvature values of the mesh and for some cell arrangements a correct depiction of the fluid volume is not possible since the mesh cannot be fitted to the underlying VOF field precisely. Further, a high number of iterations is needed in order to reduce the volume difference significantly. With respect to the runtime of a single iteration creating a visualization is an expensive task.
Even though the basic goals (mesh creation, volume correction and mean curvature reduction) could be achieved the approach presented here only fits scenarios in which a coarse representation of the interface with a connected mesh is needed. Without further work or a fundamental change of the iterative scheme the execution is too expensive and the resulting mesh too inaccurate to use the approach on a regular basis or as standard tool to visualize VOF datasets. In most cases PLIC results are similar with respect to their expressiveness and can be achieved faster.

Bibliography

[Ale78] S. Alexander. "Review: Michael Spivak, A comprehensive introduction to differential geometry". In: *Bull. Amer. Math. Soc.* 84.1 (Jan. 1978), pp. 27–32. URL: https://projecteuclid.org:443/euclid.bams/1183540369 (cit. on p. 40).

[AP91] N. Ashgriz, J. Poo. "FLAIR: Flux line-segment model for advection and interface reconstruction". In: *Journal of Computational Physics* 93 (Apr. 1991), pp. 449–468. DOI: 10.1016/0021-9991(91)90194-P (cit. on pp. 17, 19, 24).

[CM04] T. Colding, W. Minicozzi. "The space of embedded minimal surfaces of fixed genus in a 3-manifold IV; Locally simply connected". In: *Annals of Mathematics* 160 (Sept. 2004), pp. 573–615 (cit. on pp. 40, 43).

[DHKL01] N. Dyn, K. Hormann, S. Kim, D. Levin. "Optimizing 3D triangulations using discrete curvature analysis". In: (Jan. 2001) (cit. on pp. 57, 58).

[DY18] D. Dai, A. Y. Tong. "An analytical interface reconstruction algorithm in PLIC-VOF method for 2D polygonal unstructured meshes: analytical interface reconstruction in PLIC-VOF for polygonal meshes". In: *International Journal for Numerical Methods in Fluids* 88 (June 2018). DOI: 10.1002/fld.4664 (cit. on pp. 17, 20).

[HF00] D. Harvie, D. Fletcher. "A New Volume of Fluid Advection Algorithm: The Stream Scheme". In: *Journal of Computational Physics* 162 (July 2000), pp. 1–32. DOI: 10.1006/jcph.2000.6510 (cit. on pp. 17, 19).

[HN81] C. Hirt, B. Nichols. "Volume of Fluid (VOF) Method for the Dynamics of Free Boundary". In: *Journal of Computational Physics* 39 (Jan. 1981). DOI: 10.1016/0021-9991(81)90145-5 (cit. on pp. 17, 19).

[KSM+13] G. K. Karch, F. Sadlo, C. Meister, P. Rauschenberger, K. Eisenschmidt, B. Weigand, T. Ertl. "Visualization of piecewise linear interface calculation". In: *2013 IEEE Pacific Visualization Symposium (PacificVis)*. Feb. 2013, pp. 121–128. DOI: 10.1109/PacificVis.2013.6596136 (cit. on pp. 33, 34, 37).

[KSW+12] G. Karch, F. Sadlo, D. Weiskopf, C.-D. Munz, T. Ertl. "Visualization of Advection-Diffusion in Unsteady Fluid Flow". In: *Computer Graphics Forum* 31 (June 2012), pp. 1105–1114. DOI: 10.1111/j.1467-8659.2012.03103.x (cit. on p. 60).

[Kut01] W. Kutta. "Beitrag zur Näherungsweisen Integration totaler Differentialgleichungen". In: *Z. Math. Phys.* 46 (Jan. 1901) (cit. on p. 69).

[LE87] W. Lorensen, H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ACM SIGGRAPH Computer Graphics* 21 (Aug. 1987), pp. 163–. DOI: 10.1145/37401.37422 (cit. on pp. 17, 23, 26, 27).

[LRL+06] P. Liovic, M. Rudman, J.-L. Liow, D. Lakehal, D. Kothe. “A 3D Unsplit-Advection Volume Tracking Algorithm with Planarity-Preserving Interface Reconstruction”. In: *Computers Fluids* 35 (Dec. 2006), pp. 1011–1032. DOI: 10.1016/j.compfluid.2005.09.003 (cit. on pp. 19, 20).

[LZG+08] J. Lopez, C. Zanzi, P. Gómez, F. Faura, J. Hernández. “A new volume of fluid method in three dimensions—Part II: Piecewise-planar interface reconstruction with cubic-Bézier fit”. In: *International Journal for Numerical Methods in Fluids* 58 (Nov. 2008), pp. 923–944. DOI: 10.1002/fld.1775 (cit. on pp. 17, 19, 20).

[MC02] G. Miller, P. Colella. “A Conservative Three-Dimensional Eulerian Method for Coupled Solid-Fluid Shock Capturing* 1”. In: *Journal of Computational Physics* 183 (Nov. 2002), pp. 26–82. DOI: 10.1006/jcph.2002.7158 (cit. on p. 20).

[MDSB01] M. Meyer, M. Desbrun, P. Schr, A. Barr. “Discrete Differential-Geometry Operators for Triangulated 2-Manifolds”. In: *Proceedings of Visualization and Mathematics* 3 (Nov. 2001). DOI: 10.1007/978-3-662-05105-4_2 (cit. on pp. 40–43, 57).

[Mei75] G. H. Meisters. “Polygons Have Ears”. In: *The American Mathematical Monthly* 82.6 (1975), pp. 648–651. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2319703 (cit. on p. 51).

[MR74] M. Modell, R. C. Reid. *Thermodynamics and Its Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1974. ISBN: 978-0-13-914861-3 (cit. on pp. 17, 43).

[NW76] W. F. Noh, P. Woodward. “SLIC (Simple Line Interface Calculation)”. In: *Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics June 28 – July 2, 1976 Twente University, Enschede*. Ed. by A. I. van de Vooren, P. J. Zandbergen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, pp. 330–340. ISBN: 978-3-540-37548-7 (cit. on p. 19).

[OBB01a] Y. Ohtake, A. Belyaev, I. Bogaevski. “Mesh regularization and adaptive smoothing”. In: *Computer-Aided Design* 33 (Sept. 2001), pp. 789–800. DOI: 10.1016/S0010-4485(01)00095-1 (cit. on pp. 43, 44).

[OBB01b] Y. Ohtake, A. Belyaev, I. Bogaevski. “Mesh regularization and adaptive smoothing”. In: *Computer-Aided Design* 33 (Sept. 2001), pp. 789–800. DOI: 10.1016/S0010-4485(01)00095-1 (cit. on pp. 74–77).

[Pho75] B. T. Phong. “Illumination for Computer Generated Pictures”. In: *Commun. ACM* 18.6 (June 1975), pp. 311–317. ISSN: 0001-0782. DOI: 10.1145/360825.360839. URL: http://doi.acm.org/10.1145/360825.360839 (cit. on p. 44).

[Rud97] M. Rudman. “Volume-Tracking Methods for Interfacial Flow Calculations”. In: *International Journal for Numerical Methods in Fluids - INT J NUMER METHOD FLUID* 24 (Apr. 1997), pp. 671–691. DOI: 10.1002/(SICI)1097-0363(19970415)24:73.0.CO;2-9 (cit. on p. 17).

[SEG+15] K. Schulte, M. Ertl, H. Gomaa, C. Kieffer-Roth, C. Meister, P. Rauschenberger, M. Reitzle, K. Schlottke, B. Weigand. “Direct numerical simulations for multiphase flows: An overview of the multiphase code FS3D”. In: *Applied Mathematics and Computation* 272 (June 2015). DOI: 10.1016/j.amc.2015.05.095 (cit. on pp. 17, 23).

[SG19] P. Sebah, X. Gourdon. “Newton’s method and high order iterations”. In: (May 2019) (cit. on p. 61).

[SGD18] M. Skarysz, A. Garmory, M. Dianat. “An iterative interface reconstruction method for PLIC in general convex grids as part of a Coupled Level Set Volume of Fluid solver”. In: *J. Comput. Physics* 368 (2018), pp. 254–276 (cit. on pp. 20, 62).

[SP00] M. Sussman, E. Puckett. “A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase Flows”. In: *Journal of Computational Physics* 162 (Aug. 2000), pp. 301–337. DOI: 10.1006/jcph.2000.6537 (cit. on pp. 17, 20).

[SR18] H. Scheufler, J. Roenby. “Accurate and efficient surface reconstruction from volume fraction data on general meshes”. In: (Jan. 2018) (cit. on pp. 20, 27).

[Tau95] G. Taubin. “Curve and Surface Smoothing Without Shrinkage”. In: July 1995, pp. 852–857. ISBN: 0-8186-7042-8. DOI: 10.1109/ICCV.1995.466848 (cit. on pp. 75–78, 83, 85).

[Tau99] G. Taubin. “A Signal Processing Approach To Fair Surface Design”. In: *Computer Graphics (Proceedings of Siggraph ’95)* 29 (July 1999). DOI: 10.1145/218380.218473 (cit. on pp. 44, 73, 76).

[VMM99] J. Vollmer, R. Mencl, H. Müller. “Improved Laplacian Smoothing of Noisy Surface Meshes”. In: *Computer Graphics Forum* 18.3 (1999), pp. 131–138. DOI: 10.1111/1467-8659.00334 (cit. on p. 44).

[You82] D. Youngs. “Time-Dependent Multi-material Flow with Large Fluid Distortion”. In: vol. 24. Jan. 1982, pp. 273–285 (cit. on pp. 17, 19, 21, 31, 60, 85, 88).

[ZC01] C. Zhang, T. Chen. “Efficient Feature Extraction For 2D/3D Objects In Mesh Representation”. In: vol. 3. Feb. 2001, 935–938 vol.3. DOI: 10.1109/ICIP.2001.958278 (cit. on pp. 47, 48, 50).

All links were last followed on June 30, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 01.07.2019,

place, date, signature