Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 103

Visualization of Interface Instabilities in Two-Phase Flow

Alexander Straub

Course of Study:	Informatik
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Grzegorz K. Karch, M.Sc.
Commenced:	May 4, 2016
Completed:	November 3, 2016
CR-Classification:	G.1.2, G.1.3, I.3.5, I.6.6, J.2

Abstract

The simulation of two-phase flow is a very important topic which influences many modern fields of research, such as the development of combustion engines and turbines. Hence, it is a crucial task to analyze these simulation results. To this end, this thesis introduces new ideas for the calculation of the change in size of the surface area of droplets, the so-called interface stretching, and its visualization. As underlying vector field, both the surface tension force, as well as the actual velocity field provided by the simulation are used. Reconstructing the interface of a Volume-of-Fluid (VOF) field using PLIC (Piecewise Linear Interface Calculation), the changes of the interface can be calculated using different methods and are visualized using various techniques. Coloring corresponding to the stretching or contraction of the surface, as well as vector glyphs can be used to find areas of interest and provide enough information that rough estimates of the transformation of the droplets can me made. Especially using the surface tension force, local changes, such as deformation and changes in topology can be predicted.

Contents

List of Figures

List of Algorithms

1 Introduction

Simulation, especially of fluids and droplets in two-phase flow, is one of the most important subjects of modern science as it allows us to use computers and models to gain insight into complex processes and thereby often replacing difficult, time-consuming and expensive experiments. In other fields or areas of application, experiments might not even be an option, relying completely on simulation. Simulation itself is based on mathematics, physics and computer science but has its application in all kinds of research areas such as chemical industry, cooling in nuclear power plants, climate systems such as clouds, and the design, testing and improvement of combustion engines and turbines.

In this thesis, the data is provided by a simulation of two-phase flow. This means that there are two different media involved in the simulation process. Often, these are liquid and gas, but other combinations such as two different liquids are also possible. Here, all data sets are from simulations involving one liquid and one gas in the form of dispersed two-phase flow, whereby the liquid phase is present as droplets. While there exist different models used in simulations, here, the Volume of Fluid (VOF) method is used.

In order to understand the results of such a simulation, visualization is often necessary and used for the analysis of the simulation steps and thus to find and explain interesting behavior. Hence, new methods for analyzing and visualizing the results are needed.

This work therefore focuses on the local changes in area size of the fluid's surface and their visualization and interpretation. To this end, different methods for describing the change in area size will be introduced in this thesis, as well as various visualization techniques for analysis will be applied and explained. Finally, the visualization will be evaluated and interpreted.

Structure

This master's thesis is structured as described in the following, with chapters 3 to 5 explaining the previous work this thesis is based upon. In chapters 6 and 7, the actual work of this thesis is explained, while chapter 8 shows the results.

Chapter 2 – Related Works
Overview of other relevant scientific works in the same field of research.

Chapter 3 – Data structures
Explains the used data structures, their advantages and algorithms that have been customized to work on these data sets.

Chapter 4 – Volume of Fluid (VOF)
This chapter details the interface reconstruction needed in the following chapters and the visualization of volume of fluid (VOF) data obtained as result of the simulation.

Chapter 5 – Interface curvature
In this chapter, the method of Popinet for the calculation of the interface curvature is recapitulated.

Chapter 6 – Interface stretching and contraction
Many different methods for calculating the change in size of the fluid interface are presented in this chapter.

Chapter 7 – Visualization
Here, different visualization techniques are shown in order to visualize the change of the fluid's surface area.

Chapter 8 – Results
The results obtained from the chapters on *Interface stretching and contraction* and *Visualization* are explained here.

Chapter 9 – Summary and future work
Finally, a summary of the work is given in this last chapter.

2 Related Works

In fluid dynamic simulation it is important to not only simulate the flow itself, but also to simulate the interaction between fluids and gases or between fluids of different kinds and of different viscosity. While the Navier-Stokes equations are used to simulate the interior of a fluid or so-called phase, they are extended by the surface tension force in order to additionally simulate the interaction between phases. For the calculation of the surface tension force, there exists a variety of techniques, such as parabolic reconstruction of surface tension (PROST) [RR02] or balanced-force algorithms [FCD+06] [Pop09]. The latter is also used in the simulation framework FS3D [EEG+16], which was used to provide the data sets used herein and which this thesis is based on.

FS3D provides code for the simulation of multiphase flow or, as used in this case, two-phase flow, meaning that the interaction between two different phases was simulated. As technique to save the state of the phases in each time step, VOF (Volume of Fluid) [HN81] is used. Here, every cell of a grid is assigned a value between zero and one, indicating the fraction of the liquid phase in this cell. Different methods exist to calculate the VOF [AMM14], some even for more dynamic grid refinement, using quadtrees [Gre04] or adaptive grids [JY98]. However, other techniques than VOF can be used to track the interface of fluids, such as Level Set (LS) [SSO94] or a combination of both, named CLSVOF [SP00].

For the simulation, interface reconstruction is necessary. For this reconstruction there are many different techniques. In FS3D and also in this thesis, PLIC (Piecewise Linear Interface Calculation) [KSM+13] is used. This is the three-dimensional adaption of SLIC (Simple Line Interface Calculation) [NW76] and is based on the idea, that the interface is represented by a single plane in 3D or a line in 2D. More complex, but not necessarily more accurate methods are for example Marching Cubes [LC87] or MIR-techniques (Material Interface Reconstruction) [AGDJ08] [AGDJ10].

The surface tension force, as already mentioned above, is not only used within the simulation, but is also needed in this thesis to identify interface instabilities. In order to be able to calculate the surface tension force satisfactorily, it is necessary to calculate the interface curvature. For this task, there already exist various techniques, some of them designed especially for the use on VOF fields. One of those methods uses a height field with a static stencil [CFK05] and was later adapted to a more dynamic approach, due to the limitations of the height field approach [BCM+11], using a dynamic stencil and including a fallback onto another method, fitting a parabola through the interface barycenters [Pop09]. A similar approach that would additionally work even without a grid, also uses a grid-aligned height function in well-resolved areas and a height function aligned solely on the interface normal in under-resolved regions [OD15]. For other interface tracking methods, even better techniques may exist, for example with fourth-order accuracy [CG16] for LS.

Most of the work done in this area focuses on the interface reconstruction with the intent to improve the quality of the simulation. Only few research projects are dedicated to the use of reconstructed interfaces as tool in visualization and to allow further analysis of the simulation results. These works aim to provide the necessary requirements for an "accurate calculation of interface statistics" [AGDJ10], such as the possibility to visualize properties, for example the interface curvature on a PLIC-reconstructed surface [KSM+13]. Other papers use visualization to identify different fluids [LBM+06], especially in multiphase flow, or to show the flow using stream or streak lines along the fluid's surface [Wij03].

Visualizations to represent the stretching of the interface creating so-called time-surfaces are not widely researched. However, there already exist some works on tensor visualization using tensor glyphs, textures, or stream or streak lines to provide a visual representation of interface stretching [OJ12]. Additionally to this master's thesis, visualization of interface stability [OCHJ12] has already been done in a different way.

3 Data structures

As data structures, point data and rectilinear grids are used in this thesis. Both are described in this chapter, together with algorithms for the calculation of gradients needed in those cases.

3.1 Rectilinear grid

All of the simulation data is stored in rectilinear grids. Prior to a simulation, these grids can be refined in areas where details are expected, thus supplying higher accuracy where it is needed. A rectilinear grid consists of rectangular cells that may have a different length in each of the axis-aligned directions. To describe the cell sizes, an array is used for each dimension, containing the cell sizes for its particular direction. The two-dimensional case is depicted in figure 3.1, where dx_i are the components of the array describing the cell sizes along the x-axis, dy_j analogously along the y-axis.

Because of different cell sizes, distances between cell centers are not necessarily equidistant. Therefore, some calculations, such as the calculation of gradients, need to be adjusted. Hence, in the following section I will show an algorithm that is adapted to rectilinear grids.

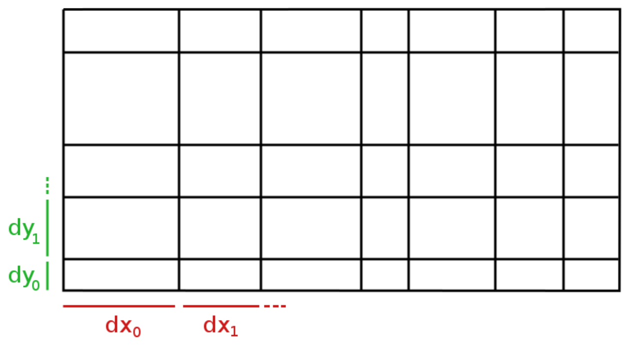

Figure 3.1: Rectilinear grid in the two-dimensional case.

3.1.1 Calculation of gradients

For the following chapters, the calculation of gradients is very important as it is needed for the reconstruction of the surface as described in chapter 4 and the calculation of the interface curvature in chapter 5.

For the calculation of the gradients in a rectilinear grid, some interpolation needs to be done. The following figure 3.2 shows the interpolation steps for the calculation of the gradient in y-direction in the two-dimensional case. The gradients along the other axis are calculated likewise. The remarks in parentheses show the difference to the 3D case that can be implemented analogously. The algorithm is part of the FS3D framework by Eisenschmidt et al. [EEG+16].

a) Calculate difference between two neighboring function values in a 3x3 (3x3x3) stencil,

b) Linearly (bilinearly) interpolate differences at cell edges,

c) Divide by $\mathrm{d}x\ \mathrm{d}y$ $(\mathrm{d}x\ \mathrm{d}y\ \mathrm{d}z)$,

d) Interpolate gradients at the cell center.

3.2 Point data

Although all data is stored in rectilinear grids, most of the derived data is not defined at the cell center or on the cell nodes, but at the barycenter of the cells' interface. Thus, the data points are arbitrarily placed and require more refined algorithms. For example, the interface is defined at the barycenter and therefore needed for its reconstruction, for the calculation of the curvature in chapter 5 and for the calculation of the surface tension in chapter 6.

3.2.1 Calculation of gradients

The method used to calculate the gradients is called a regression based method and was described by Correa et al. [CHM11]. It accepts not only points from a rectilinear grid, but works with arbitrarily placed points. The idea of this approach is to define an over-constrained linear system of equations that is then solved using linear least squares.

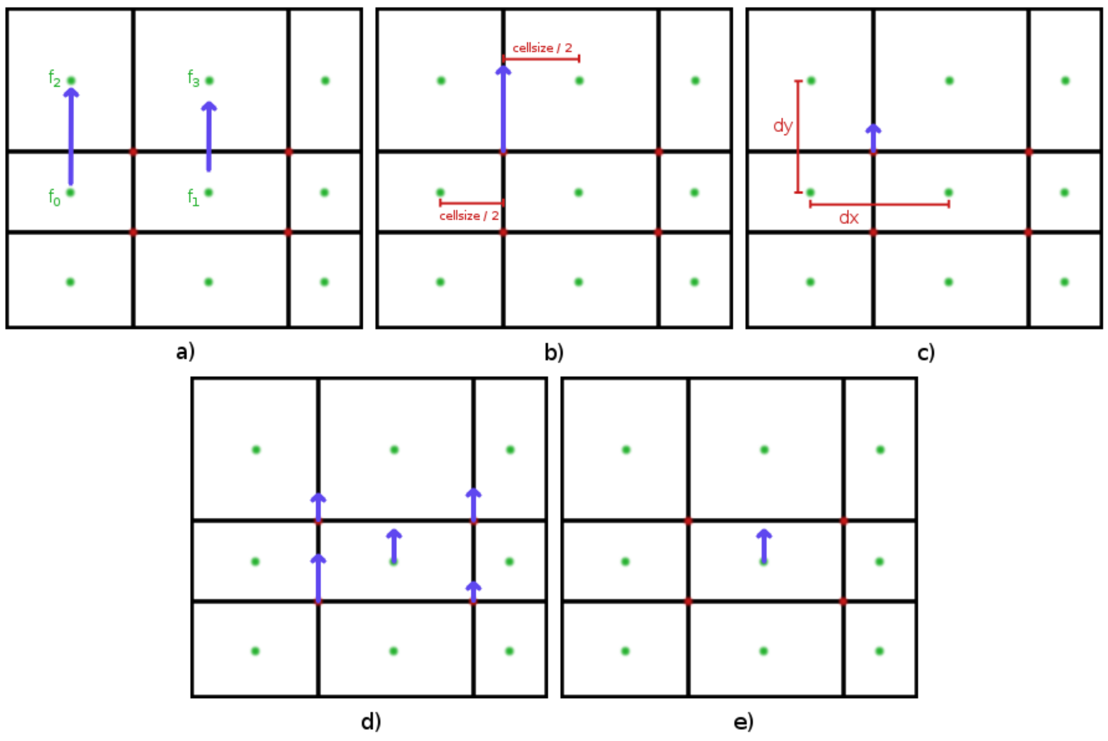

Figure 3.2: Calculation of the gradient in y-direction using interpolation in the 2D case. Images **a)** to **d)** visualize the steps of the algorithm, **e)** shows the result.

The following linear equation calculates the gradients around a point x_0:

(3.1) $X\nabla f = b$
$$\begin{bmatrix}(x_1 - x_0)^\mathsf{T}\\(x_2 - x_0)^\mathsf{T}\\\vdots\\(x_k - x_0)^\mathsf{T}\end{bmatrix}\nabla f = \begin{bmatrix}(f(x_1) - f(x_0))\\(f(x_2) - f(x_0))\\\vdots\\(f(x_k) - f(x_0))\end{bmatrix}.$$

In the equation above, only the positions and function values of the points are taken into account. To further increase the accuracy, weighted least squares can be used. Therefore, each point is assigned a weight, based on its inverse distance. This can be done by multiplying both sides by a diagonal weight-matrix $W = diag(w_i)$, with $w_i = \frac{1}{\|x_i - x_0\|_2^2}$, leading to the following equation:

(3.2) $WX\nabla f = Wb.$

3.2.2 Jacobian matrix

To calculate the Jacobian matrix, it is possible to easily adapt the above method for calculating the gradients by just executing the algorithm three times, once per vector component. Each of the execution results in one row of the Jacobian matrix, as demonstrated in the following equation for the first row using the x-component:

$$(3.3)\quad X\nabla f_x = b_x \qquad \begin{bmatrix}(x_1 - x_0)^\intercal \\ (x_2 - x_0)^\intercal \\ \vdots \\ (x_k - x_0)^\intercal\end{bmatrix} \nabla f_x = \begin{bmatrix}(f_x(x_1) - f_x(x_0)) \\ (f_x(x_2) - f_x(x_0)) \\ \vdots \\ (f_x(x_k) - f_x(x_0))\end{bmatrix}.$$

The extension to weighted least squares can be done analogously:

$$(3.4)\quad WX\nabla f_x = Wb_x.$$

The result is the Jacobian matrix:

$$(3.5)\quad J = \begin{pmatrix}\nabla f_x^\intercal \\ \nabla f_y^\intercal \\ \nabla f_z^\intercal\end{pmatrix}.$$

4 Volume of Fluid (VOF)

The volume of fluid method (VOF) is a widely used method to describe the free boundaries in fluid simulations. This method was originally introduced by Hirt and Nichols in 1981 [HN81]. The idea is to only store one scalar value per cell indicating the fractional volume of the liquid phase, where 1 indicates that the cell is an interior cell that is completely filled, 0 indicates that this is a gas cell or cell belonging to the other phase, and a value between 0 and 1 indicates that the cell is only partially filled. The VOF is therefore the fraction of the cell's volume that is occupied by the liquid phase.

4.1 Interface reconstruction (PLIC)

The code used to reconstruct the interface in my implementation was borrowed from the FS3D framework by Eisenschmidt et al. [EEG+16], which is also the source of the data used in this work. For this reason, the PLIC (Piecewise Linear Interface Calculation) method is used which has been described in great detail by Karch et al. [KSM+13], thus keeping consistency with the simulation. Although Marching Cubes would have yielded a more visually pleasing reconstruction, PLIC has the advantage of not using interpolation as Marching Cubes does in order to create a seamless visualization. An example of a reconstructed interface using PLIC can be seen in figure 4.1. All three images are from the same dataset but of different time steps. The central image additionally shows the edges of the PLIC surfaces.

An only partially filled cell is called an interface cell. The interface itself is only a straight line in the 2D case or a plane in 3D. Although the cell has an interface, it does not necessarily mean that the interface is continued in a neighboring cell. This means that there can be C^0 or even C^{-1} discontinuities. Using the VOF value, it is possible to recreate the interface by calculating the gradient of the VOF field and using its inverse as interface normal. Thus, no additional information than the VOF itself needs to be stored.

To reconstruct the interface, the following algorithm 4.1 is a possible solution, using the PLIC (Piecewise Linear Interface Calculation) method.

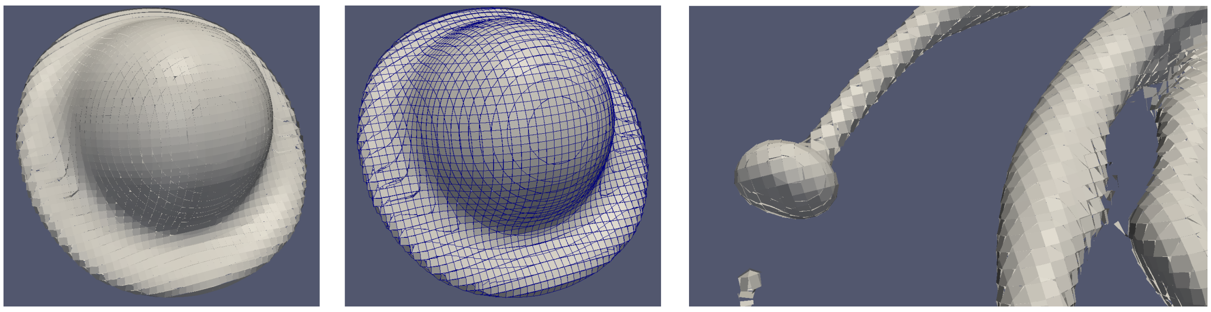

Figure 4.1: Reconstructed interface using PLIC

1. Calculate normal as inverse normalized gradient of the VOF field,
2. Get cell corner that would be the last corner covered with fluid if the fluid drained in the direction of the gradient,
3. Calculate distance between the corner and the straight line (2D) or plane (3D) defined by the VOF and the interface normal,
4. Calculate up-point on the interface,
5. Intersect plane (or straight line) defined by the normal and the interface point with all the cell edges.

The intersections calculated in the last step are the corners of the convex polygon (line in 2D) that represent the interface of that cell.

The calculation of the distance between the corner and the interface, as needed by the algorithm, can be done by subdividing the cell iteratively until a satisfying approximation has been found. A possible method for the function *Approx_Distance_To_Interface* could be as follows:

1. Set error $e = \infty$,
2. While error $e > \epsilon$,
   a) Guess a good subdivision plane (straight line in 2D), with normal defined by inverse gradient of the VOF field, or do binary subdivision,
   b) Calculate fractional volume of fluid VOF' using the subdivision plane as interface,
   c) Update error as difference between calculated and given VOF value: $e = |VOF - VOF'|$.
3. Return distance between corner and subdivision plane.

Algorithmus 4.1 Reconstruction of the interface

```
Reconstruct_Interface ( cell_center : Vector, VOF : Field, cellsize : Array[] )
  normal := Gradient ( cell_center, VOF, cellsize ).normalized.inverse

  corner := cell_center
  if (normal[x] < 0) corner[x] += cellsize[x] / 2, else corner[x] -= cellsize[x] / 2
  if (normal[y] < 0) corner[y] += cellsize[y] / 2, else corner[y] -= cellsize[y] / 2
  if (normal[z] < 0) corner[z] += cellsize[z] / 2, else corner[z] -= cellsize[z] / 2

  distance := Approx_Distance_To_Interface ( corner, VOF[cell_center], cellsize, normal )

  interface_point := corner + distance * normal

  FOR EACH cell_edge
    edges.Add ( Intersect ( plane { interface_point, normal }, cell_edge ) )
  END FOR EACH

  RETURN edges
END Reconstruct_Interface
```

4.2 Interface barycenter

As most of the data is defined at the interface barycenter, for example the curvature and the surface tension, its calculation is important regarding the next chapters. To calculate the barycenter for a cell, the following algorithm can be used:

1. Reconstruct the interface as shown in the previous section,
2. Calculate centroid using the algorithm 4.2:
    a) Transform all polygon edges into 2D space,
    b) Use an algorithm to sort the edges, providing the convex hull,
    c) Calculate the centroid as shown in equation 4.1,
    d) Use inverse transformation on the centroid to get it back into 3D space.

Algorithmus 4.2 Calculation of the centroid of a polygon in 3D

```
Centroid ( edges : Vector[], normal : Vector )
  origin := edges[0]
  edges_2D := Transform_to_2D ( edges, normal, origin )

  edges_2D_convex_hull := Graham_Scan ( edges_2D )

  centroid_2D := Calculate_Centroid ( edges_2D_convex_hull )

  centroid := Transform_to_3D ( centroid_2D, normal, origin )

  RETURN centroid
END Centroid
```

$$
\begin{aligned}
C_x &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(x_i + x_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right) \\
C_y &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(y_i + y_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right), \text{with}
\end{aligned}
\tag{4.1}
$$

$$
A = \frac{1}{2}\sum_{i=0}^{n-1}\left(x_i y_{i+1} - x_{i+1} y_i\right)
$$

5 Interface curvature

In this chapter, a method to calculate the curvature of the interface is explained. This algorithm, introduced by Popinet in 2009 [Pop09], works on a VOF data field and is used because it is a state-of-the-art algorithm and was also used in the FS3D framework by Eisenschmidt et al. [EEG+16]. The curvature is later used to calculate the surface tension in chapter 6.

5.1 Height field-based method by Popinet

In order to find the interface curvature of a cell, I use the method introduced by Popinet [Pop09]. This method is a combination of two different and improved methods that leads to more robust results.

The main algorithm shown below in algorithm 5.1, combines two methods and calls them successively if the previous algorithm could not yield a satisfying result. The first called algorithm tries to calculate the curvature using a height function, the second one tries to approximately fit a parabola using least squares and then calculating its curvature directly on the reconstructed polynomial.

The curvature algorithm 5.1 therefore starts by calculating the interface normal as normalized inverse gradient of the VOF field. As second step it then uses the normal to find the coordinate axis to which the angle is minimal, thus, to which it is best aligned to. In 3D this is one of the following directions, returned by the function *Get_Best_Aligned_Directions*: left, right, up, down, back or to the front.
The height function curvature method is called using the direction as argument. If it is able to calculate the curvature, it directly returns it, terminating the algorithm. If unable to find a consistent result, it proceeds with the next best direction. If it does not find a solution using one of the directions, the main algorithm skips to the next method.

For the second method, the algorithm uses the consistent positions of the failed first approach. If there were not enough consistent, linear independent positions returned by the first algorithm, it uses all the barycentric interface positions in a 3x3 stencil,

Algorithmus 5.1 Method by Popinet for the calculation of the curvature

```
Curvature ( cell : Cell, VOF : Field )
 normal := Gradient ( cell, VOF ).normalized.inverse

 direction[] := Get_Best_Aligned_Directions ( normal )

 FOR EACH direction
  curvature, local_positions[] := Height_Function_Curvature ( cell, VOF, direction )
  consistent_positions[].add ( local_positions[] )

  IF curvature IS CONSISTENT, RETURN curvature
 END FOR EACH

 IF consistent_positions.Linear_Independent_Size() < 6 // (< 3 in the 2D case)
  consistent_positions[] := Get_Interface_Positions ( cell, VOF )
 END IF

 IF consistent_positions.Linear_Independent_Size() < 6, RETURN 0 // (< 3 in the 2D case)

 RETURN Parabola_Fitted_Curvature ( cell, consistent_positions[] )
END Curvature
```

respectively in a 3x3x3 stencil in 3D. In the case of not enough linear independent neighboring interface cells however, the algorithm is unable to yield a result and returns a curvature of zero. Having enough independent positions though, the second algorithm is executed. This algorithm then tries to approximately fit a parabola through these positions using linear least squares. The curvature of the resulting polynomial can be calculated directly.

5.1.1 Height-function curvature

The first algorithm used in the method of Popinet [Pop09] is the height function curvature algorithm 5.2. The idea behind it is to calculate the interface height by summing up all the fractions along a certain axis. At the same time it also determines the position of the interface.

In the first step, the interface height and the base cell are determined for the central cell, followed by the calculation of the interface position. These steps are then repeated for all neighboring cells in the plane perpendicular to the given direction. Additionaly, the calculated interface height needs to be adjusted by setting it to the same base level as of the center cell, in order to provide a common origin. Then, the algorithm checks for the consistency of all individual heights and in case of an inconsistency, it

Algorithmus 5.2 Height-function curvature

```
Height_Function_Curvature ( cell : Cell, VOF : Field, direction : Direction )
  height[0], base[0] := Interface_Height ( cell, VOF, direction )
  interface[0] = base[0] + height[0] * direction

  FOR EACH neighbor : Neighbor ( cell, direction )
    height[i], base[i] := Interface_Height ( neighbor, VOF, direction )
    interface[i] = base[i] + height[i] * direction

    height[i] := height[i] + ( base[i] - base[0] )
  END FOR EACH

  IF NOT Consistent ( height[] ), RETURN { , interface[] }

  RETURN { Curvature ( height[] ), {} }
END Height_Function_Curvature
```

only returns the positions for which the interface height has been calculated successfully. However, if it passed the consistency check, the curvature can be calculated using the equation 5.1, as provided by Cummins et al. in [CFK05] for the two-dimensional case and, derived from the fundamental matrix form, from [Gol05] for three dimensions.

$$(5.1)\quad \kappa = \begin{cases} \dfrac{\frac{\partial^2 h}{\partial x^2}}{(1+\frac{\partial h}{\partial x}^2)^{3/2}} & \text{in 2D} \\ \dfrac{\left(1+\left(\frac{\partial h}{\partial x}\right)^2\right)\frac{\partial^2 h}{\partial y^2}-2\frac{\partial h}{\partial x}\frac{\partial h}{\partial y}\frac{\partial^2 h}{\partial x\partial y}+\left(1+\left(\frac{\partial h}{\partial y}\right)^2\right)\frac{\partial^2 h}{\partial x^2}}{(1+\left(\frac{\partial h}{\partial x}\right)^2+\left(\frac{\partial h}{\partial y}\right)^2)^{3/2}} & \text{in 3D} \end{cases}$$

Height function

The height function curvature method calculates the interface height for a specific column. Here I needed to make a modification to the method provided by Popinet [Pop09] for it to work on rectilinear grids.

The change in the algorithm is the calculation and usage of a weight. This weight is needed because of the different cell sizes in a rectilinear grid. As the height is only calculated in one direction though, only the cell size in the same direction needs to be considered.
The algorithm 5.3 starts by assigning the initial height value as the starting cell's volume fraction, multiplied by the cell size in the given axis direction as weight. Then, starting at this cell, the algorithm iterates over the cells in the given direction, adding their weighted fraction to the total height. This is done until the interface was found and the

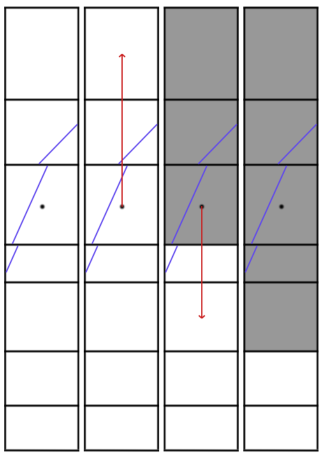

Figure 5.1: Run of the algorithm to calculate the interface height. The gray cells mark the cells whose volume fractions have already been added to the sum, the blue lines represent the interface.

iteration has reached an outer cell. The consistency check then makes sure, that the calculated height in this direction was valid.
Afterwards, the algorithm starts at the same cell as before, repeating the procedure in the opposite direction until an interior cell is reached. After a renewed sanity check for consistency, the calculated height and the base cell are returned.

In figure 5.1 you can see an example run of the algorithm in two-dimensional space. From the starting cell, the algorithm iterates to the top until it reaches a cell without any fluid. The volume fractions from all of these cells, multiplied by the corresponding cell heights, are summed up as indicated by the gray area. The same procedure is then repeated in the opposite direction, again starting at the marked cell, down to a cell that is on the inside of the fluid. These cells' weighted volume fractions are also added to the interface height. The lowest cell is called the base cell and is then returned together with the calculated interface height.

5.1.2 Parabola-fitted curvature

The second algorithm, used by Popinet [Pop09], uses the interface positions of the neighboring cells to fit a parabola. In a first step, the normal is again calculated using the inverse gradient of the VOF field. In the second step, the barycenter of the interface in the central cell is calculated by using the algorithm introduced in section 4.2 from the previous chapter. Using an orthonormal coordinate system with the barycenter as origin

Algorithmus 5.3 Interface height

```
Interface_Height ( cell : Cell, VOF : Field, direction : Direction )
 weight := Cell_Size ( cell, direction )
 height := VOF ( cell ) * weight

 current_cell := cell
 current_fraction := VOF ( cell )

 interface := current_fraction < 1

 WHILE NOT interface OR 0 < current_cell < 1
  current_cell := Neighbor ( current_cell, direction )
  current_fraction := VOF ( current_cell )

  weight := Cell_Size ( current_cell, direction )
  height := height + current_fraction * weight

  IF 0 < current_fraction < 1, interface := true
 END WHILE

 IF current_fraction != 0, RETURN NOT_CONSISTENT

 current_cell := cell
 current_fraction := VOF ( cell )

 interface := current_fraction > 0

 WHILE NOT interface OR 0 < current_cell < 1
  current_cell := Neighbor ( current_cell, direction.inverse() )
  current_fraction := VOF ( current_cell )

  weight := Cell_Size ( current_cell, direction )
  height := height + current_fraction * weight

  IF 0 < current_fraction < 1, interface := true
 END WHILE

 IF current_fraction != 1, RETURN NOT_CONSISTENT

 RETURN height, current_cell
END Height_Function_Curvature
```

Algorithmus 5.4 Parabola-fitted curvature

```
Parabola_Fitted_Curvature ( cell : Cell, VOF : Field, positions : Vector[] )
  normal := Gradient ( cell, VOF ).normalized.inverse
  origin := Barycenter ( cell, VOF )

  positions := Transform ( positions, normal, origin )

  function := Least_Squares ( positions )

  RETURN Curvature ( function )
END Parabola_Fitted_Curvature
```

and the normal as new z-axis, the interface positions are transformed. Thus, the x- and the y-component are the new positions in two-dimensional space, with the z-component as function value. Using least squares, the following equation is then minimized:

$$F(a_i) \equiv \sum_{1\leq j\leq n} \left[z'_j - f(a_i, \vec{x}_j{}')\right]^2, \text{with}$$
$$f(a_i,\vec{x}) \equiv \begin{cases} a_0x^2 + a_1x + a_2 & \text{in 2D,} \\ a_0x^2 + a_1y^2 + a_2xy + a_3x + a_4y + a_5 & \text{in 3D} \end{cases} . \tag{5.2}$$

The result of least squares is a second order polynomial that can be used directly to calculate the curvature using the equation

$$\kappa \equiv \begin{cases} \frac{2a_0}{(1+a_1^2)^{3/2}} & \text{in 2D,} \\ 2\frac{a_0(1+a_4^2)+a_1(1+a_3^2)-a_2a_3a_4}{(1+a_3^2+a_4^2)^{3/2}} & \text{in 3D} \end{cases} . \tag{5.3}$$

5.2 Results

Because of the combination of methods, the algorithm reaches high robustness. Most of the time, it is sufficient to apply the first method using the height field and is thus mostly independent of interface reconstruction methods. Only in the last case, needing to substitute the derived interface positions by the barycenters of the interface, interface reconstruction becomes necessary.

In total, the algorithm by Popinet [Pop09] achieves a second-order accuracy and due to the dynamic stencil used to calculate the height field, it is also an improvement to the Hessian matrix that uses a fixed 3x3 stencil, respectively 3x3x3 stencil in 3D.

6 Interface stretching and contraction

Using the velocity field computed in the simulation or the surface tension as described in the following paragraphs, it is possible to visualize a variety of interesting properties.

The surface tension is a force with direction perpendicular to the interface that depends on the curvature and the fluids or gases involved. The following is the definition of the surface tension as given by Eisenschmidt et al. [EEG+16], where σ is a constant depending on the media and κ is the interface curvature:

(6.1) $s = \sigma\kappa\nabla f.$

To visualize the change of the interface area, e.g. the stretching or contraction of the surface, a velocity field or vector field derived from the surface tension force is used. In the following, the expression velocity field stands as generalized for either of the both vector fields.

For the calculation of the ratio between new and old interface size, I introduce different methods in the following sections. Most of them use the symmetric part of the Jacobian matrix J^+ of the velocity field, as there must not be any complex Eigenvalues and the method introduced in the following works only on divergence-free vector fields. The method to calculate the Jacobian matrix from a vector field with no explicit topology was already introduced in the previous section 3.2.2 of the chapter *Data structures*.

Also, in some of the approaches I use parts of the method of stretch-minimizing from Bartoň et al. [BKC15]. The idea is, that the Jacobian matrix gives an indication in its vicinity about how a vector is stretched or contracted. An easy example can be found in image 6.1. Here, the vector field **F**, represented by the two vectors $\vec{x_1}$ and $\vec{x_2}$, is used to calculate the Jacobian matrix. Applying the following formula

(6.2) $u^\intercal J u,$

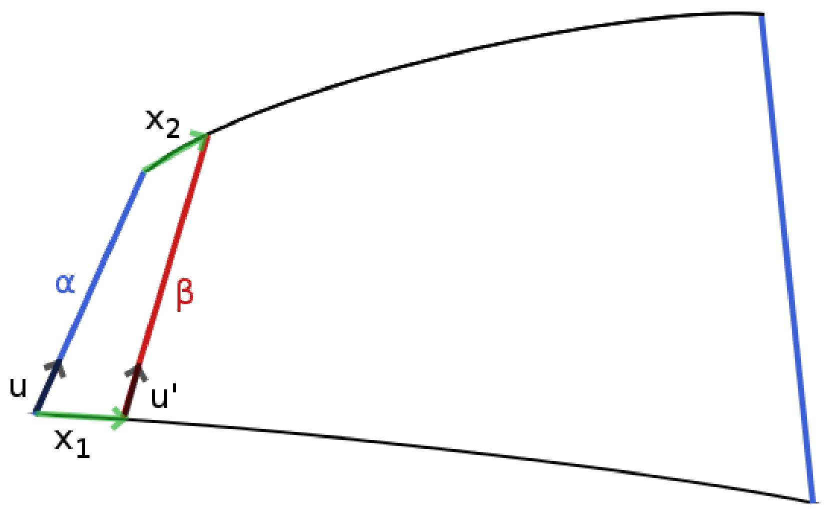

Figure 6.1: Stretch-minimizing example

on the unity vector $\vec{u}$, the stretching or contraction is calculated. Thus, the length β is approximated by

$$\beta = \alpha + \alpha * (u^{\intercal} J u)\,, \tag{6.3}$$

which is a first-order Taylor approximation of stretching. The above equation indicates how the length of a unity vector u is scaled along its direction. For an arbitrary vector v this can be expressed as

$$|v'| = |v| + \left(\left[\frac{v}{|v|}\right]^{\intercal} J \left[\frac{v}{|v|}\right]\right) * |v| = \left[1 + \left(\left[\frac{v}{|v|}\right]^{\intercal} J \left[\frac{v}{|v|}\right]\right)\right] * |v|. \tag{6.4}$$

This leads to the following equation 6.5 that shows the multiplicative change of the vector:

$$1 + \left(\left[\frac{v}{|v|}\right]^{\intercal} J \left[\frac{v}{|v|}\right]\right) = 1 + \frac{\mathrm{d}v'}{\mathrm{d}t}. \tag{6.5}$$

Furthermore, it is very important to keep the time step size δ_t in mind to get the actual stretching instead of the stretching rate and only by applying the time step size — assuming that the time step was well chosen for the simulation — it can be assured that the result of the equation is a positive number. Also, as explained in the introduction to this section, it is necessary to only use the symmetric part of the Jacobian matrix. The following equation 6.6 contains all the mentioned changes:

$$1 + \delta_t * \left(\left[\frac{v}{|v|} \right]^{\intercal} J^{+} \left[\frac{v}{|v|} \right] \right). \tag{6.6}$$

6.1 Geometrical approach

The following algorithm calculates the ratio between the area after applying the Jacobian matrix to calculate the amount of stretching or contraction and the original area:

1. Reconstruct the interface using the algorithm 4.1 introduced in chapter 4,
2. Calculate the barycenter applying the algorithm 4.2,
3. Calculate the area A_{orig} of the polygon using the equation 4.1, solving for A,
4. Use method above, applying equation 6.6 on each of the vectors from the barycenter to the edges calculated in step 1,
5. Calculate new distances between the barycenter and the edges by scaling with the result of the previous step and moving the edges accordingly,
6. Calculate the area A_{trans} of the polygon defined by the translated edges using the equation 4.1, solving for A,
7. Return the ratio A_{orig}/A_{trans} as result.

An example is given in figure 6.2. In *a)* you can see a PLIC tile from above, with its edges and barycenter marked. The vectors along which the expansion or contraction is calculated are additionally shown in *b)*. Finally in *c)*, the translated vertices are marked in green, together with the new area. The green area in relation to the gray area gives us the ratio of change of the surface area in the given cell. In figure 6.3 a) you can see this method applied on the dataset of two colliding droplets.

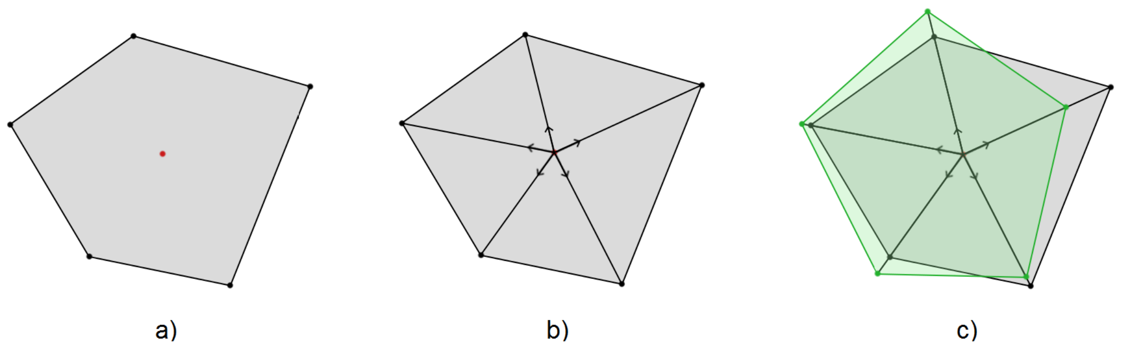

Figure 6.2: Geometrical approach. Image **a)** shows the PLIC surface for one cell with its barycenter marked red. In image **b)**, the vectors from the barycenters towards the edges are shown additionally. Image **c)** demonstrates the new surface with its edges translated along the vectors.

6.1.1 Alternative approach

Another possible approach could be the following:

1. Get all neighboring interface cells in an axis-aligned plane that is closest to perpendicular to the interface normal of the central cell,
2. Calculate their barycenters,
3. Create a triangle fan around the barycenter of the central cell using the barycenters of the neighbors,
4. Calculate the sum of the areas of the triangles,
5. Proceed like in the algorithm above and repeat the creation of a triangle fan for the translated barycenters,
6. Return the ratio A_{orig}/A_{trans} as result.

The problem with this approach is its higher complexity. To create the triangle fan to calculate the area, it is necessary to find the convex hull of the projection of the vertices onto a plane. Additionally, as the Jacobian is only defined locally, it is better to use it on points that are closer to its point of definition.

6.2 Using Eigenvalues

Instead of looking for vectors and calculating the change in size in their direction, the Eigenvalues and Eigenvectos can be computed and used as such. As only the symmetric part of the Jacobian matrix is taken into account, all Eigenvalues are real and as long as there are no multiple Eigenvalues or Eigenvalues of value zero, their corresponding Eigenvectors build an orthonormal base of $\mathbb{R}^3$. Using the equation 6.6 as origin, with $J^+v = \lambda v$ and $|v| = 1$, the following equation 6.7 holds:

$$
\begin{aligned}
&1 + \delta_t * \left(v^\intercal J^+ v\right) \\
=&1 + \delta_t * (v^\intercal \lambda v) \\
=&1 + \delta_t \lambda * (v^\intercal v) \\
=&1 + \delta_t \lambda * |v|^2 \\
=&1 + \delta_t \lambda.
\end{aligned}
\tag{6.7}
$$

Thus, the computed Eigenvalues and Vectors can directly be used to describe the change of size in their respective directions.

As there are three Eigenvalues, there is a possibility of using only those that have special properties. Therefore, I implemented different methods of choosing the appropriate Eigenvalue or Eigenvector. The different methods are described as follows. Every method has its own advantages and disadvantages, mostly depending on the intent of the user as they can highlight different properties.

6.2.1 Minimum Eigenvalue

Opposite to the maximum Eigenvalue, the minimum highlights areas of contraction, as a small Eigenvalue $(\lambda < 0)$ indicates a contraction in the direction of its corresponding Eigenvector. The figure 6.3 g) also shows an example for this case.

Result: Change in direction of the Eigenvector.

6.2.2 Maximum Eigenvalue

The maximum Eigenvalue is bound to highlight areas with the highest rate of expansion, as a high Eigenvalue $(\lambda > 0)$ means a stretching in the direction of the Eigenvector. An example image can be seen in figure 6.3 h).

Result: Change in direction of the Eigenvector.

6.2.3 Maximum absolute Eigenvalue

Using the maximum absolute Eigenvector, the Eigenvector with the strongest influence is selected. Thus, per interface cell it indicates whether there is a contraction or an expansion. See figure 6.3 i) for an example.

Result: Change in direction of the Eigenvector.

6.2.4 Mean Eigenvalue

As combination of the minimum and the maximum Eigenvalue, the mean Eigenvalue gives an indication for the actual change. Again an example can be found in figure 6.3 j).

Result: Change of the area spanned by two Eigenvectors.

6.2.5 Closest-to-parallel Eigenvectors

The selection of the Eigenvalue as shown above can give a good indication. However, as the change in size relates to the surface area, it is better to find an Eigenvector that is parallel or almost parallel to it. Using two almost parallel Eigenvectors has the advantage that these span a two-dimensional sub space. Thus, the resulting change in size relates even more to the area, instead of to a single direction. See figure 6.3 f) for an example.

Result: Change of the area spanned by two Eigenvectors.

6.2.6 Eigenvectors from 2D space

For this special case, the Jacobian matrix is transformed into a 2x2 Jacobian matrix, defined on the surface area. This so-called metric tensor thus indicates the stretching and contraction of vectors in a two-dimensional subspace, e.g. along the interface. For this purpose it was used by Obermaier et al. in [OJ12]. First, the Jacobian matrix is multiplied by the time step, then added to the identity matrix:

(6.8) $J' = \delta_t * J + I.$

Now, given two vectors $\vec{r}$ and $\vec{s}$ on the interface, with $|r| = |s| = 1$, and forming a matrix $(r|s)$, J' is multiplied, resulting in a 3x2 matrix:

(6.9) $J_{3x2} = J' * (r|s).$

Finally, by multiplying the matrix with its transposed, the two-dimensional Jacobian matrix can be calculated:

(6.10) $J_{2D} = J_{3x2}^{\intercal} J_{3x2}.$

This matrix is now used to provide the local Eigenvectors v_1, v_2 and Eigenvalues λ_1, λ_2. To transform these Eigenvectors back into 3D space, it is necessary to multiply them by the rs-matrix:

(6.11)
$$\begin{aligned} v_{3D} &= (r|s) * v, \\ \lambda_{3D} &= \sqrt{\lambda}. \end{aligned}$$

This case is also depicted in figure 6.3 e). These local Eigenvectors are of further interest as they are used for texturing as can be seen in the next chapter in section 7.3.

Result: Change of the area spanned by two Eigenvectors.

6.3 Combined method

This method is an improvement to the above described method of taking the two closest-to-parallel Eigenvectors. Here, it just takes the closest-to-parallel Eigenvector and its corresponding Eigenvalue. Another vector is then calculated as orthonormal to this Eigenvector and the interface normal. Thus, the two vectors are nearly parallel to the interface and indicate the change of the area. Because the second vector does not have a corresponding Eigenvalue, the change in its direction is again calculated as in equation 6.6. In figure 6.3 b), this method is also visualized as an example.

6.4 Using advection

Using advection instead of the stretch computation as above, similar results can be obtained.

6.4.1 Advection on interfaces

Another approach is to simulate the advection of the interfaces, thus giving an approximation of their location in the next time step or in between. To this end, the surface tension field or another velocity field are used to move the interface barycenters accordingly. Both, before and after translation, the surface points in a 3x3 neighborhood perpendicular to the interface normal are used to create a triangle mesh to calculate an approximation of the surface area. The ratio of these surface areas is then the indicator for expansion or contraction of this particular interface cell. In 6.3 c) a visualization of this method can be found.

6.4.2 Advection on particles or interface edges

Similarly to the advection of the interface barycenters as described above, it is possible to advect the edges of the reconstructed interface or a set of particles on the surface. Since the velocity field is only defined at the barycenters however, it needs to be interpolated or approximated at those positions. This approximation can be done by applying the Jacobian matrix, defined at a position $\vec{x}$, as shown in the following equation:

$$f(\vec{x_i}) = f(\vec{x}) + J_f(\vec{x}) * (\vec{x_i} - \vec{x}). \tag{6.12}$$

This matches the first-order Taylor approximation for a position $\vec{x_i}$ close to $\vec{x}$. An example can be found in 6.3 d).

6.5 Results

In figure 6.3 you can see the differences between the above described methods. Not all of them are useful to describe the stretching and contraction of the surface area, but may provide other useful information.

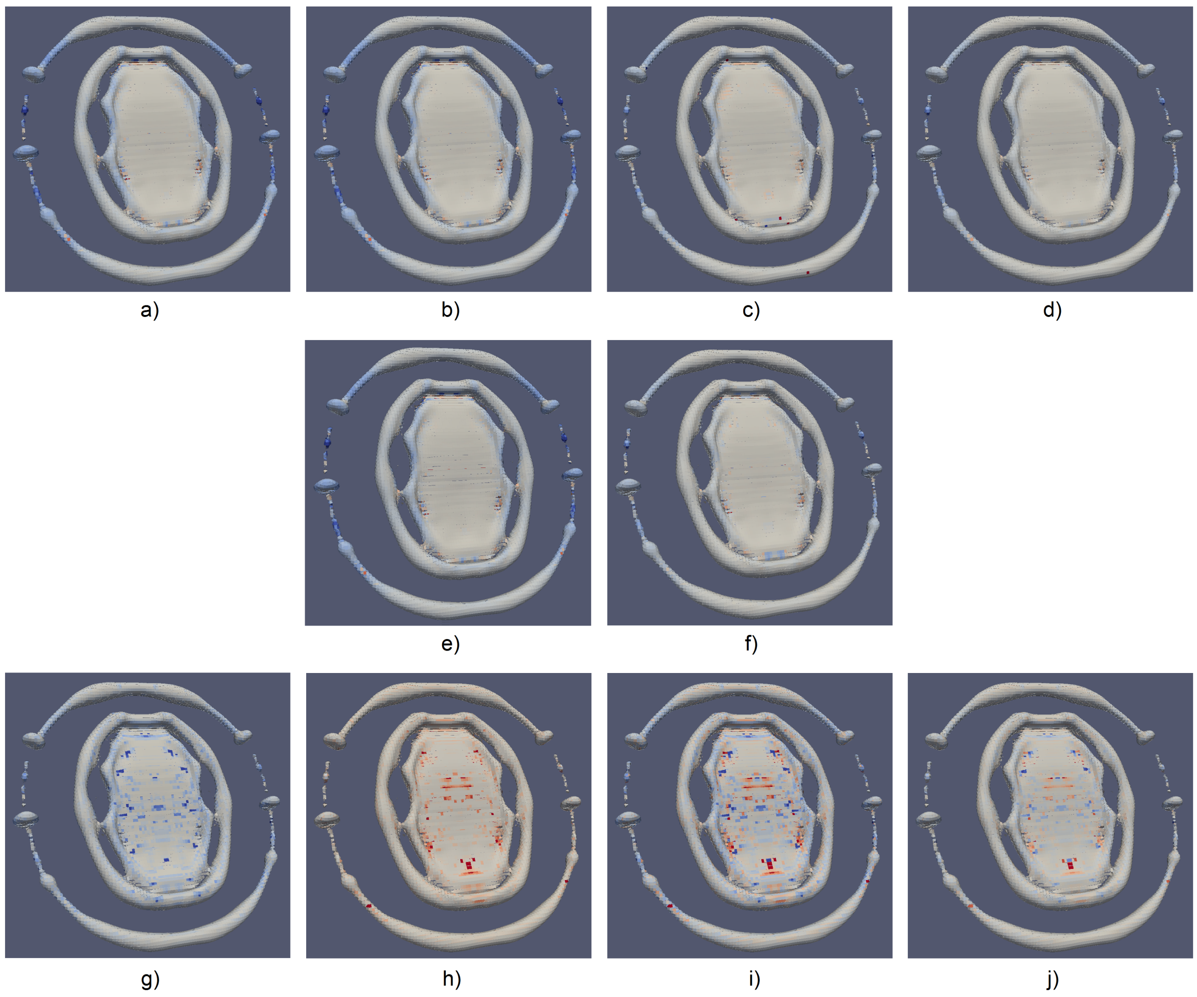

Figure 6.3: Comparison between different approaches as described in the following sections: **a)** Geometrical approach, **b)** Combined method, **c)** Advection on interfaces, **d)** Advection on particles or interface edges, **e)** Eigenvectors from 2D space, **f)** Closest-to-parallel Eigenvectors, **g)** Minimum Eigenvalue, **h)** Maximum Eigenvalue, **i)** Maximum absolute Eigenvalue, **j)** Mean Eigenvalue.

The images a) to f) are all very similar to each other, highlighting the same regions and only differing slightly. These methods are all equally useful in visualizing stretching and contraction of the surface area.

The methods used in the images in the bottom row, g) to j), are less useful in giving an indication about the stretching or contraction of the surface area, but better highlight the deformation in a specific direction. Thus, they can be used to identify areas of stress and possibly changes in topology.

7 Visualization

Based on the numerical analysis of the previous chapter, here I introduce and explain different methods of visualization. This is done on a dataset of two colliding droplets of the same material, produced by the simulation framework FS3D described in Eisenschmidt et al. [EEG+16]. The images are taken from two different time steps of the simulation, visualizing the vector field of the surface tension force and the velocity field respectively.

In figure 7.1, the four top images a) to d) are from one time step to the next, while the four lower images e) to h) represent another two subsequent time steps. For every image there is also one close-up on an interesting region in the image right below. These zoomed-in regions are used in the following sections, applying different methods of visualization.

7.1 Coloring

The first method uses only color to visualize the change in size of the surface area. This was done using one of the methods described in the previous chapter 6. As explained in its results section 6.5, different approaches can be used as they differ only slightly. In the following, the calculated Eigenvectors from the two-dimensional subspace of the interface plane from section 6.2.6 are used.

In figure 7.2 a), the image from the first time step using the surface tension force highlights the area below the ring-like structure. Comparing this region to the next time step, as can be seen in figure 7.1 d), it seems that the highlighted area has changed significantly.

Another obvious change can be found in figure 7.2 c), compared to the next time step in figure 7.1 h). Here, the blue area is subject to thinning.

Using the velocity field on the right side in image 7.2, it is harder to make such an assumption. A direct link between the coloring and the change can not be made.

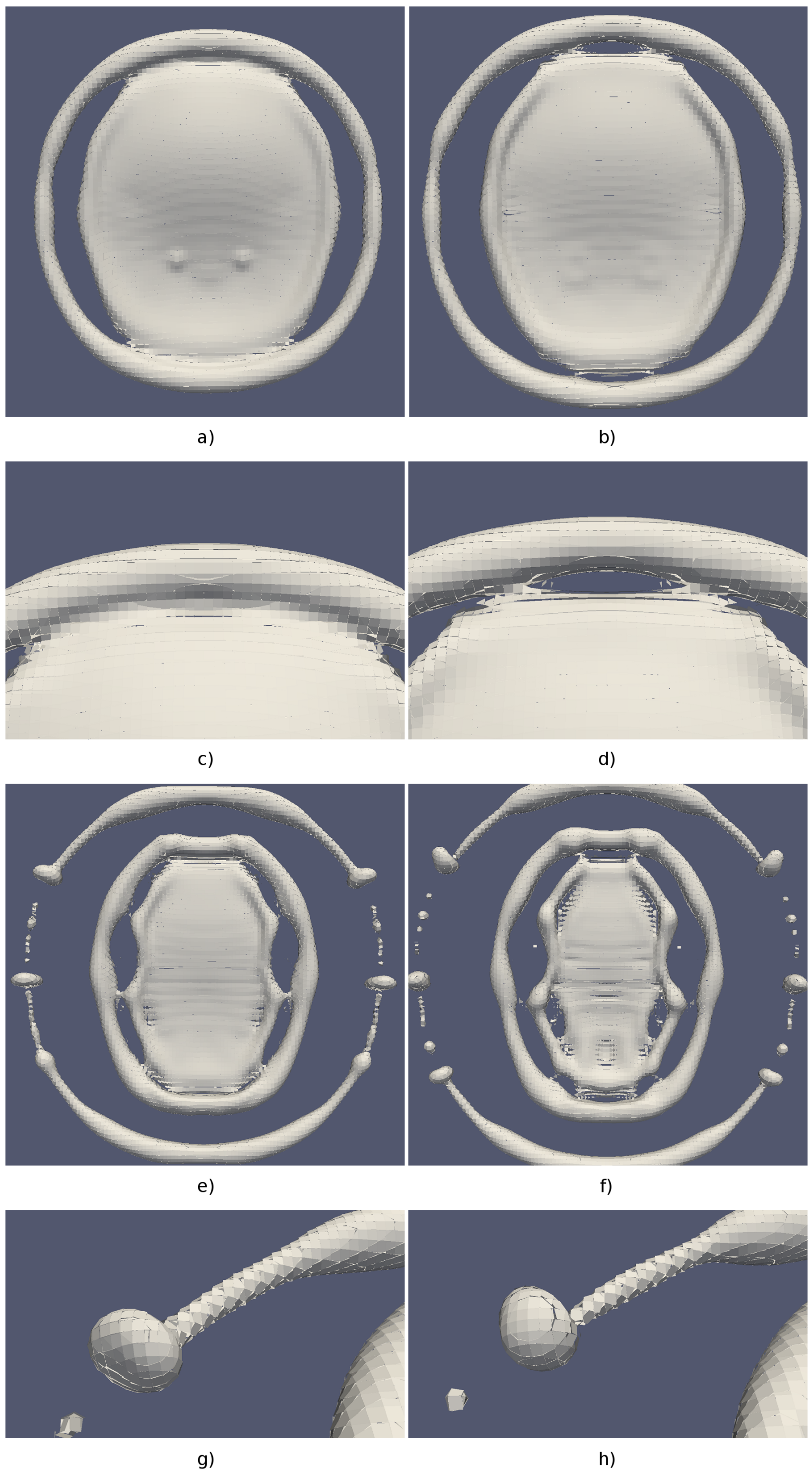

Figure 7.1: Overview of the used dataset. Image **a)** is from the first time step, image **b)** is from the subsequent time step. The same is true for images **c)** and **d)**, with them being a zoom-in on an interesting region of **a)** and **b)**. Images **e)** and **f)** are again two consecutive time steps, with **g)** and **h)** zooming-in on an interesting area.

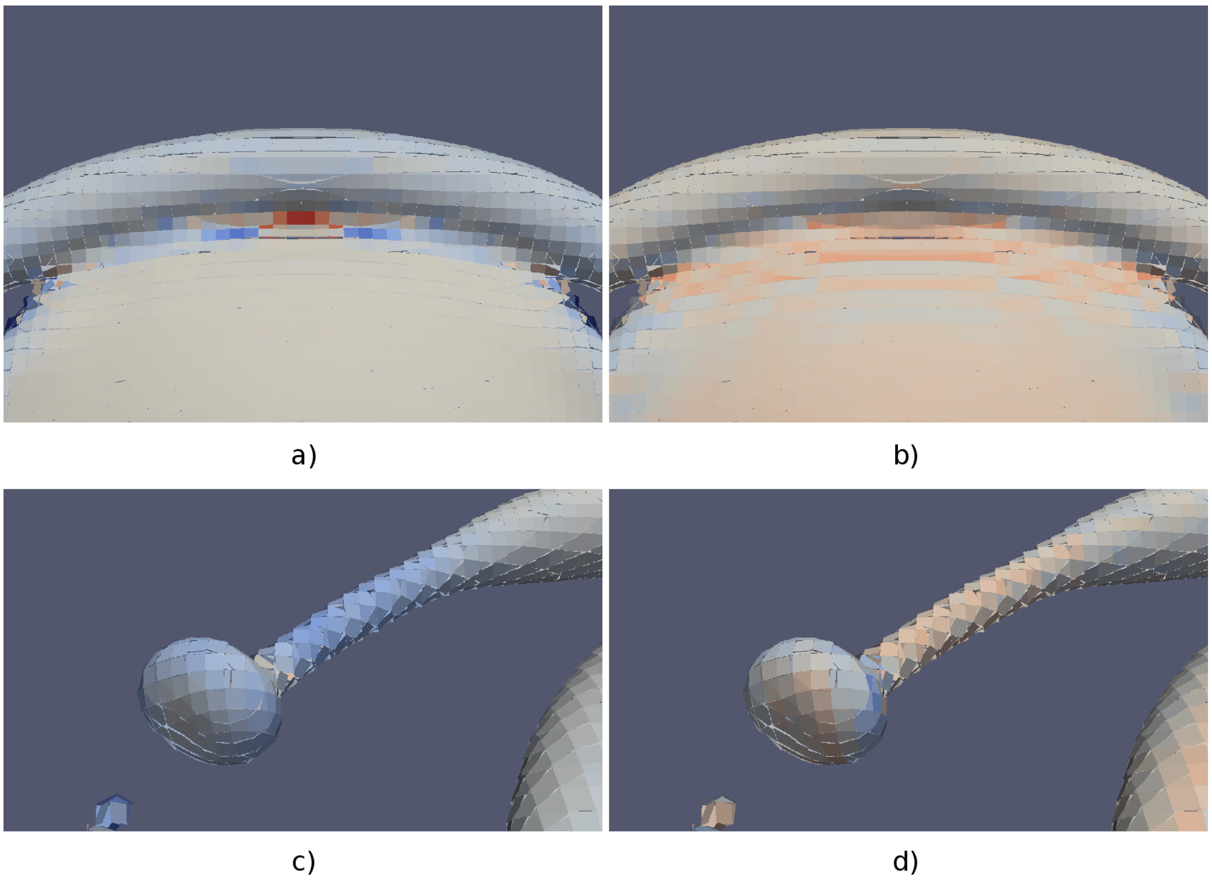

Figure 7.2: Colored surface on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps.

7.2 Glyphs

With only color, the direction of the change has been neglected. To additionally visualize the direction, two types of glyphs are used in the following: vector glyphs indicating one direction and tensor glyphs which need three vectors as input.

7.2.1 Vector glyphs

As stretching and contraction along a line has no particular orientation, it is better to use tube glyphs instead of arrows. Using the two local Eigenvectors on the surface as described in section 6.2.6, the glyphs form a cross with differently colored tubes, depending on the stretching or contraction along the respective tube. This can be easily observed in figure 7.3. However, it seems quite difficult to identify the same regions as with color alone.

This becomes easier though when looking at only one glyph per interface cell at a time as is the case in figure 7.4. Here, the Eigenvectors have been split into two groups, thus, in images a) and b) the Eigenvector corresponding to the larger one of the two Eigenvalues per cell is shown, while in images c) and d) it is the one that corresponds to the smaller Eigenvalue. These I call primary and secondary Eigenvectors in the following.

Further of interest seems to be that the primary Eigenvector runs along the pipe-like structure while at the same time, the secondary Eigenvector runs perpendicular and alongside the curved surface.

7.2.2 Tensor glyphs

Usually, tensor glyphs represent a matrix. Here however, they represent three vectors: the two Eigenvectors calculated using the method described in section 6.2.6, and a perpendicular normalized vector. The result can be seen in figure 7.5. In this case, the glyphs are like discs. The representation is very similar to the two-dimensional glyphs from the previous section representing the primary Eigenvectors. It seems to be easier however to recognize patterns and structures, but there is no indication of the amount of stretching and contraction.

7.3 Texturing

Another method to visualize the stretching and contraction can be achieved using texturing. In figure 7.6 it is shown how this can be done using two-dimensional Eigenvectors and their corresponding Eigenvalues for scaling of the texture. The Eigenvectors used have been previously calculated using the method in section 6.2.6 of the previous chapter.

For the actual texturing in figure 7.7, a 100x100 pixel wide texture with white Gaussian noise is used. This makes spotting interesting regions quite easy, as can be seen especially in image 7.7 a). However, it is harder to tell the extent of stretching or contraction as opposed to using colors.

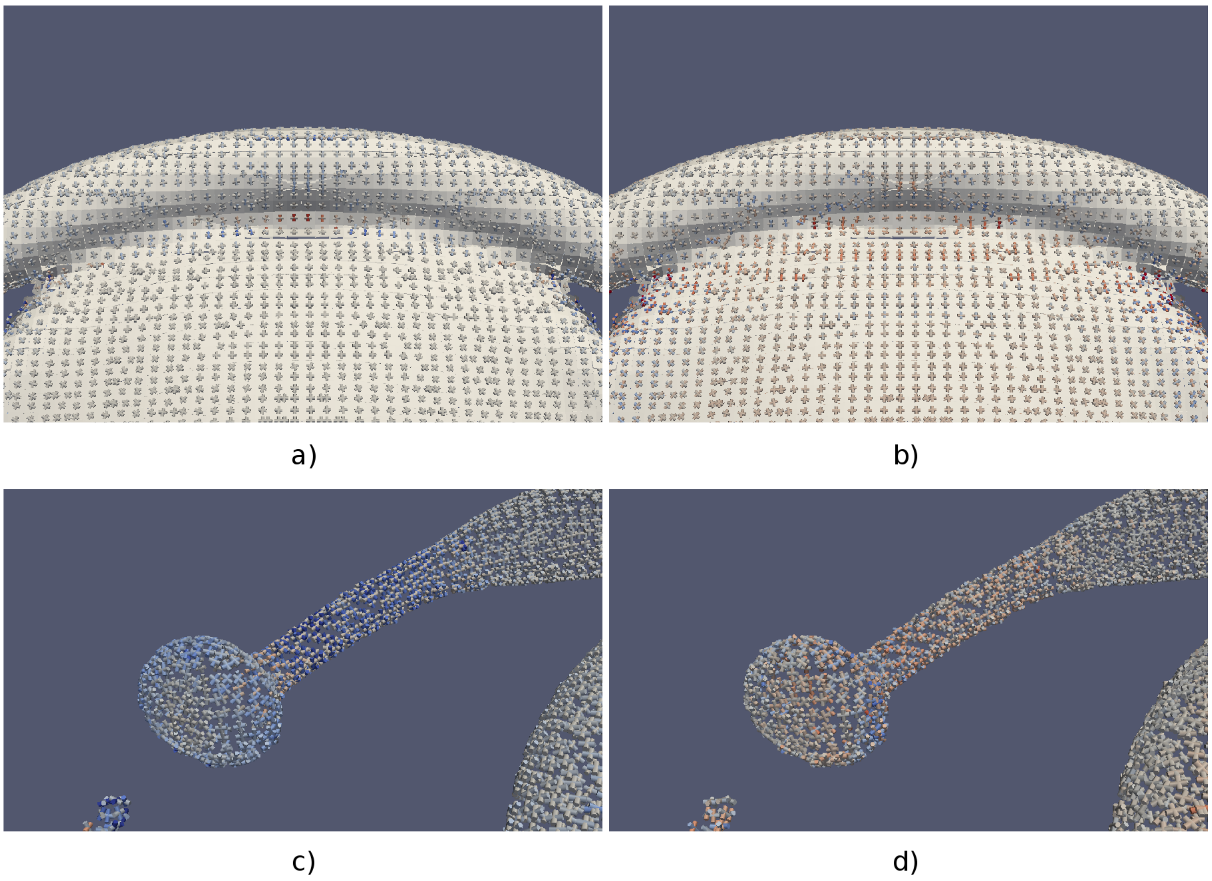

Figure 7.3: Vector glyphs on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps. In the top images **a)** and **b)**, the reconstructed interface is shown additionally to the glyphs to avoid visual clutter.

7.4 Combination of methods

In this section, I will discuss some combinations of the above described methods. However, not all of these methods can be combined in a meaningful fashion and more than two single methods combined might lead to more visual clutter than it would help.

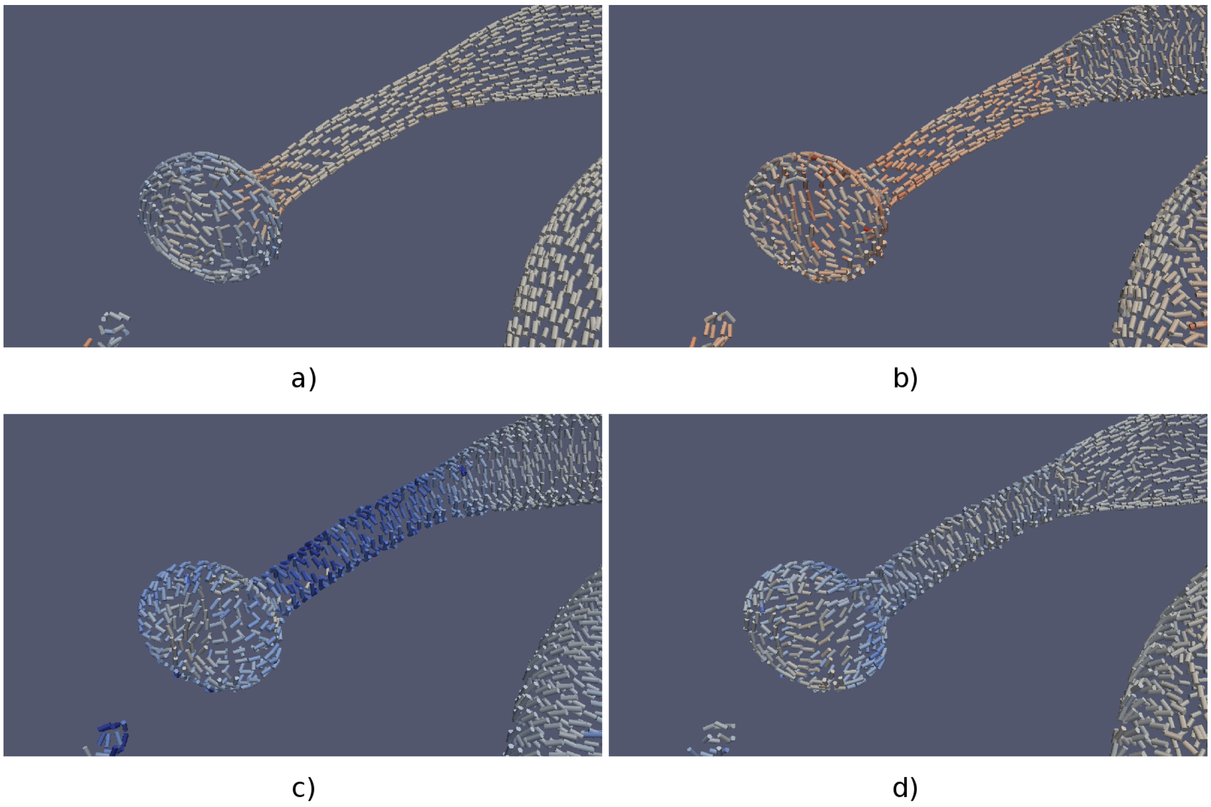

Figure 7.4: Separated into primary glyphs in images **a)** and **b)** and secondary vector glyphs in **c)** and **d)**. The two images on the left are again visualizing the surface tension force, the two images on the right the velocity field.

7.4.1 Color and vector glyphs

As the downside of color-mapping was the missing information about the directions along which the change occurs, the combination of coloring and using vector glyphs solves just that. The result can be observed in figure 7.8. Now, interesting regions can be identified quickly and at the same time further interpretation is made easier.

7.4.2 Color and tensor glyphs

Nearly the same as with vector glyphs in the previous section can be said about the combination of color and tensor glyphs. Image 7.9 now shows the disc glyphs together with a colored PLIC surface. Again, interesting regions can be found quickly through colors while the tensor glyphs indicate the primary direction of change.

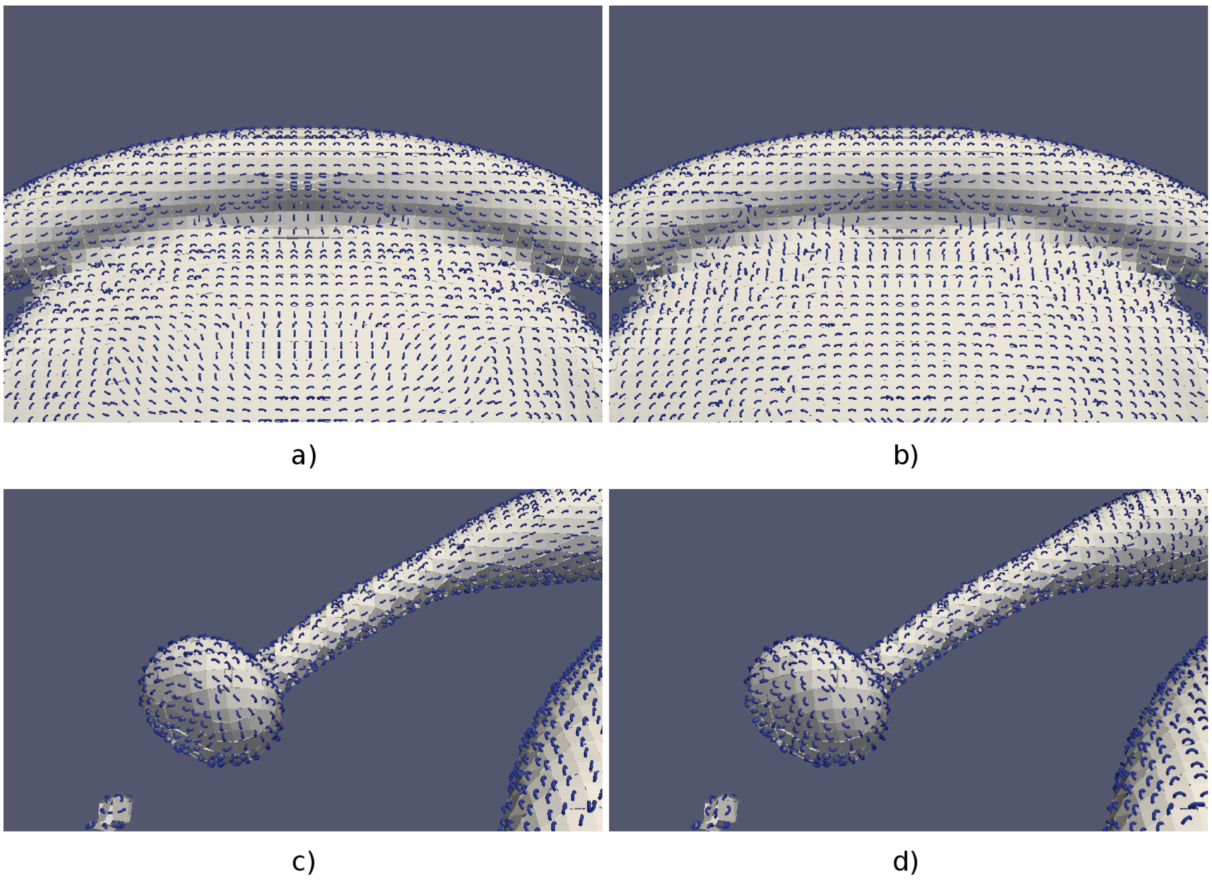

Figure 7.5: Tensor glyphs on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps. In all images, the reconstructed interface is shown additionally to the glyphs to avoid visual clutter.

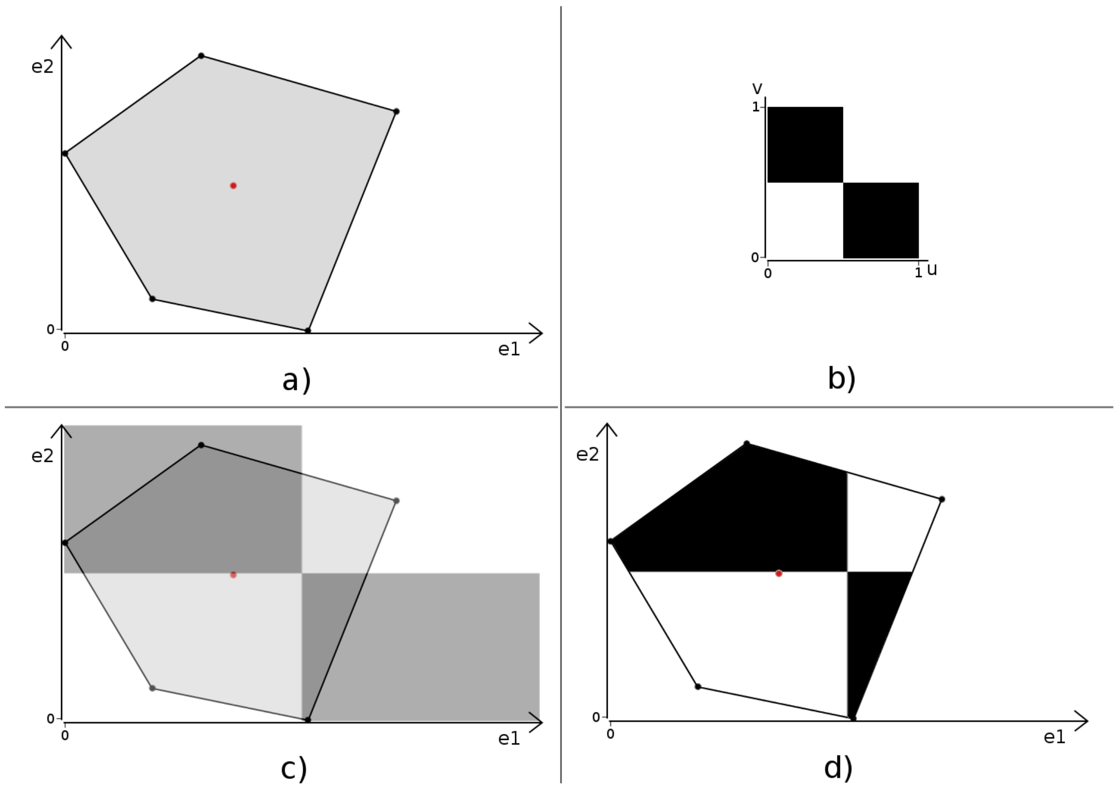

Figure 7.6: Texturing of the PLIC surface. Image **a)** shows the PLIC interface with its origin in 2D space, as well as its two-dimensional Eigenvectors e_1 and e_2, scaled by their corresponding Eigenvalues. A simple texture with its texture coordinates u and v is shown in image **b)**. In image **c)** you can see the texture scaled using the Eigenvalues, while image **d)** shows the result: the textured PLIC polygon.

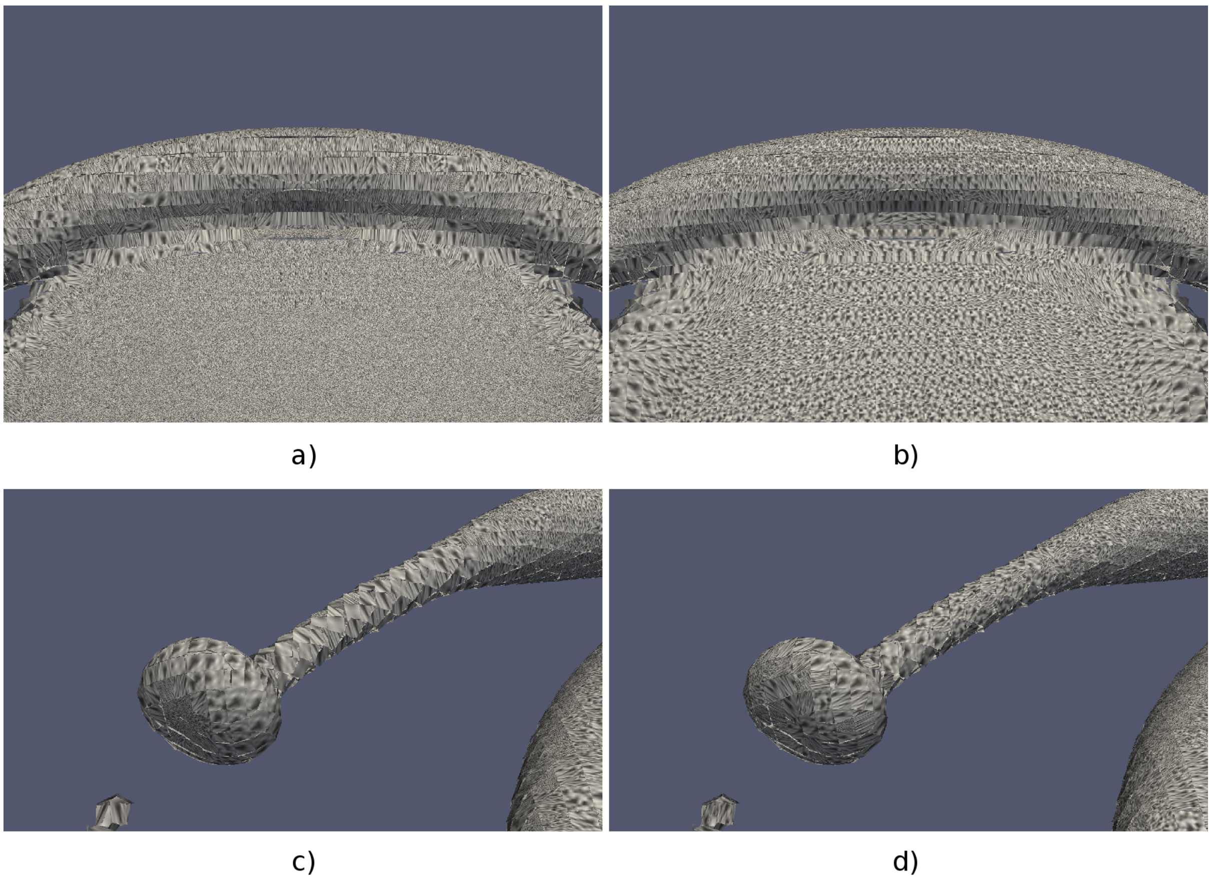

Figure 7.7: Texturing on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps.

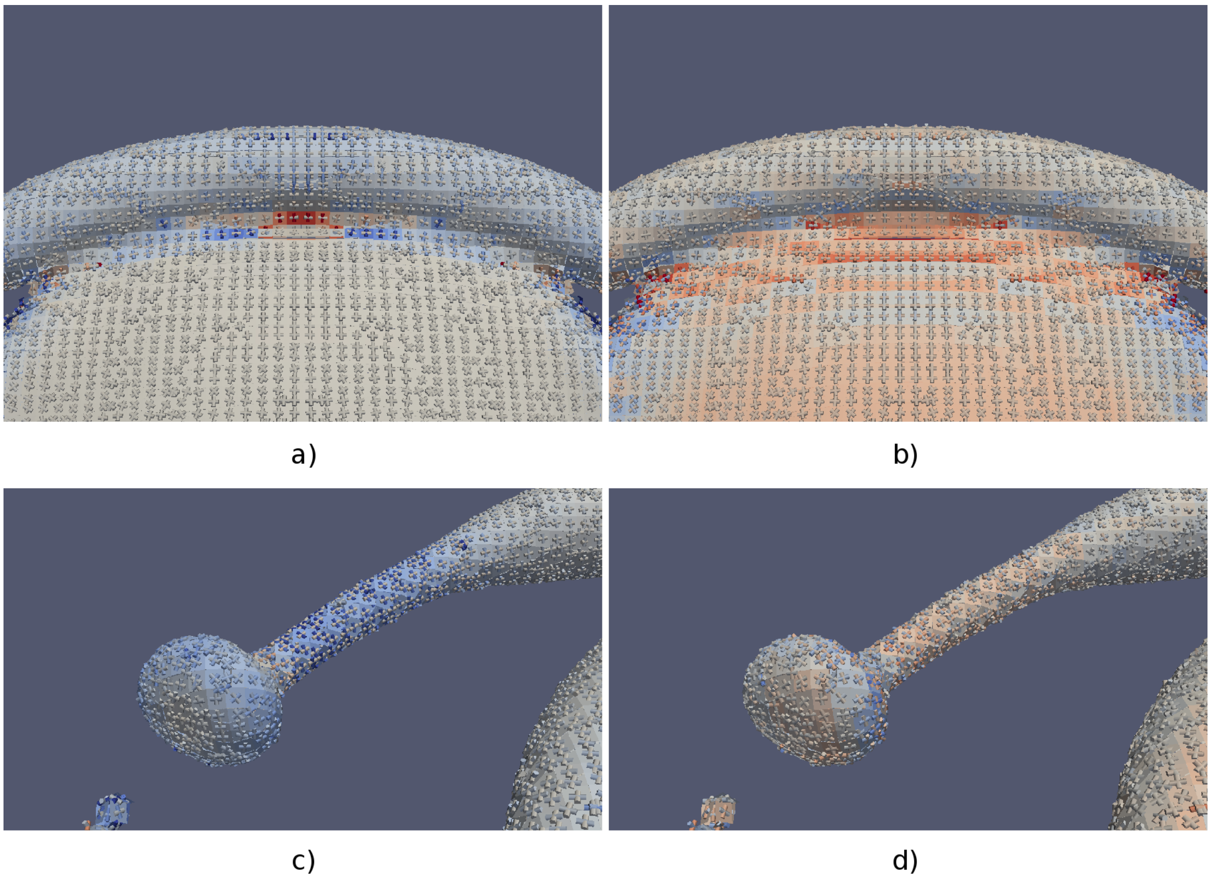

Figure 7.8: Color and vector glyphs on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps.

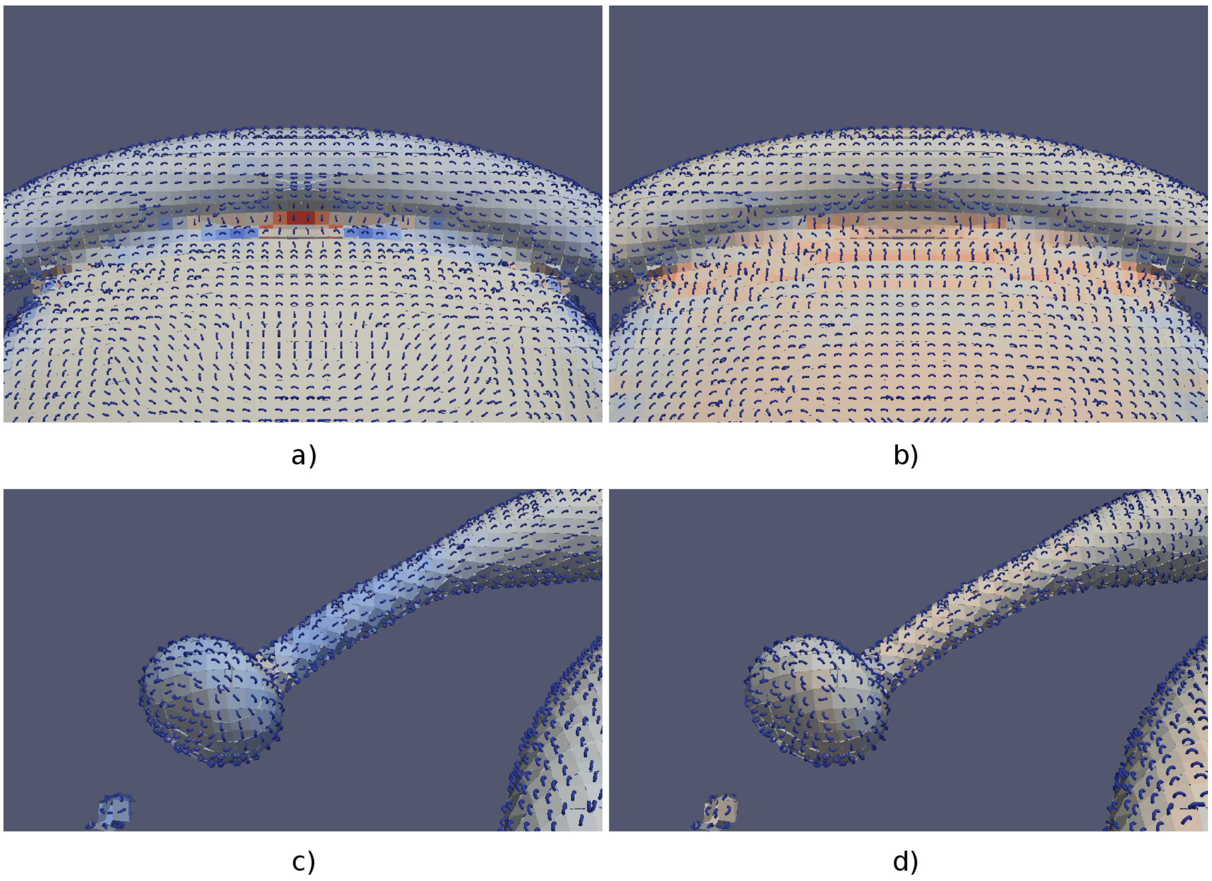

Figure 7.9: Color and tensor glyphs on different time steps. Images **a)** and **c)** visualize the surface tension force, whereas images **b)** and **d)** visualize the velocity field of two different time steps.

8 Results

The goal of the visualization methods of the previous chapter is to **a)** find areas of interest, e.g. areas in which change will occur, and **b)** to be able to make predictions about what will change in topology, as well as object shape and size. To this extent, one can use either the surface tension force or the complete velocity field provided by the simulation.

8.1 Areas of interest

For the first task, using a colored PLIC representation is very helpful to identify these areas as the coloring actually indicates a change in size of the surface area and additionally can be quickly perceived. Another useful method is to use texturing. Here, the contorted texture leaves behind structures that can be further interpreted, but generally give a good indication of where to look for changes. Furthermore, all of the other methods using color can be used as well, especially those which use color in combination with another method or colored vector glyphs.

To more than identify areas of interest, but also to be able to predict the changes, colored vector glyphs can be used. These have the advantage of combining the information about the magnitudes of change and their directions. In order to best be able to interpret what is going to happen and to avoid visual clutter, the vector glyphs can be separated into larger and smaller Eigenvalues per cell. Thus, the visualization of the glyphs corresponding to the smaller Eigenvalue indicates a thinning or contraction along their direction and additionally in some cases a breakdown into separate droplets, as can be seen in figure 8.1. The herein used second data set simulates two droplets moving at different speeds in the same direction and merging upon collision. On the other hand, glyphs corresponding to the larger Eigenvalue indicate coherence or expansion and are especially large where a fusion of droplets is happening, see figure 8.2 for an example. Other methods than vector glyphs do not seem to provide the same amount of information in such an easy-to-understand fashion. However, methods using vector glyphs in combination with any other visualization technique may provide similar results.

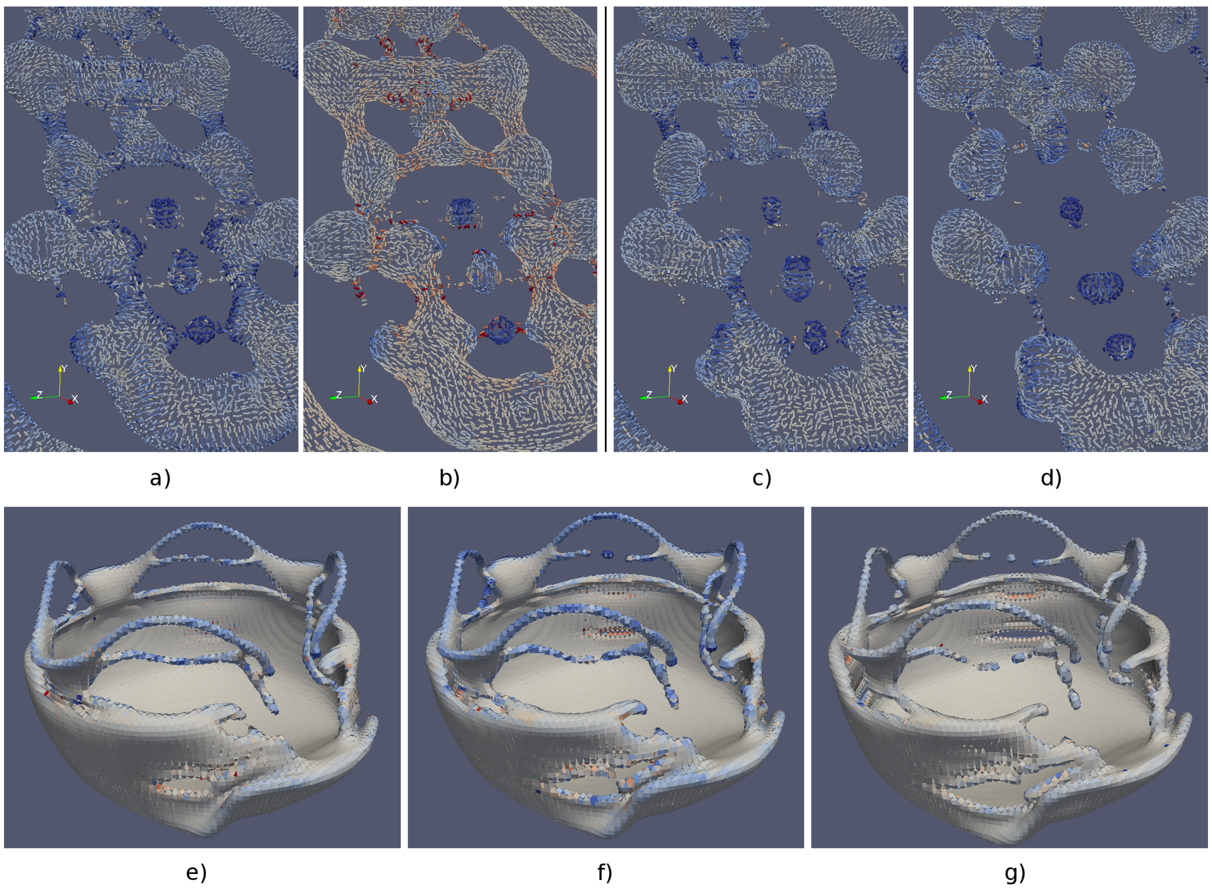

Figure 8.1: Breakdown into separate droplets in two different data sets, showing only the surface tension force. In the images above, you can see many connections between sphere-like structures thinning and collapsing. Images **a)** and **b)** are from the same time step, with **a)** showing the glyphs corresponding to the smaller Eigenvalues and **b)** showing the glyphs corresponding to the larger ones. Images **c)** and **d)** show subsequent time steps showing only the glyphs of the smaller Eigenvalues. The images below represent three succeeding time steps of the second data set using only color as indicator. In the foreground, the collapsing of the arch-like structure into multiple droplets can be seen.

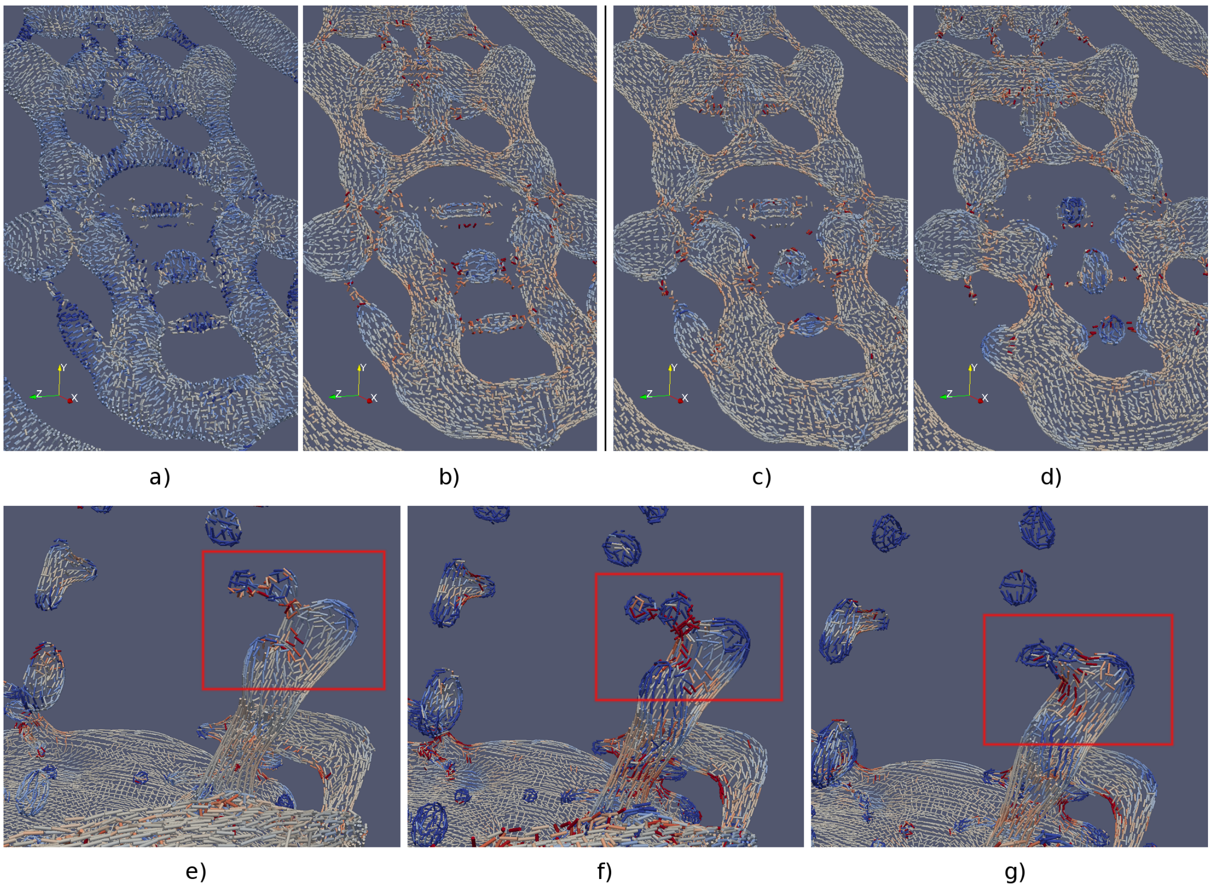

Figure 8.2: Fusion of separate droplets in two different data sets, showing only the surface tension force. In the images above, you can see the fusion of two sphere-like structures on the left side of the images. Images **a)** and **b)** are from the same time step, with **a)** showing the glyphs corresponding to the smaller Eigenvalues and **b)** showing the glyphs corresponding to the larger ones. Images **c)** and **d)** show subsequent time steps showing only the glyphs of the larger Eigenvalues. The images below represent three succeeding time steps of the second data set using only vector glyphs corresponding to the larger Eigenvalue. On the right side of each image, the fusion of droplets can be seen.

8.2 Surface tension force and velocity field

The surface tension force only considers the force field acting on the surface with direction perpendicular to the interface. Thus, this vector field provides results as if the objects were neither moving nor subject to other forces such as gravity. Contrary to this, the velocity field considers the surface tension force as well as velocities and pressures already present. It is therefore the actual velocity field calculated in the simulation. While the velocity field thus gives a global view on changes, the surface tension might give a more locally defined view.

Which vector field to use seems to depend on the goals. If the interest is on how the objects themselves are deforming, the surface tension force might bring insight. To gain more knowledge about the movement of droplets, the velocity field may be used. Looking at the images from the previous chapter 7 however, it gives the impression that the visualization of the surface tension force is easier in terms of interpretation.

9 Summary and future work

In this master's thesis, I used the idea introduced by Bartoň et al. in [BKC15], as well as own ideas, to introduce many different methods for calculating the change in size of a PLIC surface area and the extraction of directions along which those changes occur. To the different methods belong approaches that calculate the contraction or expansion of vectors, use the Eigenvalues and Eigenvectors of the Jacobian matrix, and advection schemes.

On these methods I then used various visualization techniques. I thereby found out that colored PLIC interfaces, indicating the change in size of their area, and texturing help quickly identifying regions that are subject to deformation. Using colored vector glyphs in those regions, guesses and predictions can be made concerning the change in size and form of objects, as well as changes in their topology. Additionally, the usefulness of the surface tension force as basis for the calculation of the change in area size has been shown as it provides an alternative and more local view on the deformation of droplets than the actual velocity field provided by the simulation and thus leads to a better understanding of droplet interaction.

Future work

In this work, only a view visualization techniques have been tested, but many other methods could yield interesting results. Also, the possibility of combining different techniques to further improve the ability of prediction leads to many possible follow-ups. Besides, in this thesis, only local changes have been observed, not having considered the topology. Furthermore, only time-local information has been processed, leaving room for more exploration, for example by tracking particles over subsequent time steps. Additionally, the analysis and visualization can be easily adapted to also support multiphase flow.

Bibliography

[AGDJ08] J. C. Anderson, C. Garth, M. A. Duchaineau, K. I. Joy. “Discrete Multi-Material Interface Reconstruction for Volume Fraction Data.” In: *Computer Graphics Forum* 27.3 (2008), pp. 1015–1022. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2008.01237.x (cit. on p. 13).

[AGDJ10] J. C. Anderson, C. Garth, M. A. Duchaineau, K. I. Joy. “Smooth, Volume-Accurate Material Interface Reconstruction.” In: *IEEE Transactions on Visualization and Computer Graphics* 16.5 (Sept. 2010), pp. 802–814. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.17 (cit. on pp. 13, 14).

[AMM14] W. Aniszewski, T. Ménard, M. Marek. “Volume of Fluid (VOF) type advection methods in two-phase flow: A comparative study.” In: *Computers & Fluids* 97 (2014), pp. 52–73. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2014.03.027. URL: http://www.sciencedirect.com/science/article/pii/S0045793014001261 (cit. on p. 13).

[BCM+11] G. Bornia, A. Cervone, S. Manservisi, R. Scardovelli, S. Zaleski. “On the properties and limitations of the height function method in two-dimensional Cartesian geometry.” In: *Journal of Computational Physics* 230.4 (2011), pp. 851–862. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2010.11.029. URL: http://www.sciencedirect.com/science/article/pii/S0021999110006443 (cit. on p. 14).

[BKC15] M. Bartoň, J. Kosinka, V. M. Calo. “Stretch-minimising stream surfaces.” In: *Graphical Models* 79 (2015), pp. 12–22. ISSN: 1524-0703. DOI: 10.1016/j.gmod.2015.01.002. URL: http://www.sciencedirect.com/science/article/pii/S1524070315000041 (cit. on pp. 31, 57).

[CFK05] S. J. Cummins, M. M. Francois, D. B. Kothe. “Estimating curvature from volume fractions.” In: *Computers & Structures* 83.6–7 (2005). Frontier of Multi-Phase Flow Analysis and Fluid-StructureFrontier of Multi-Phase Flow Analysis and Fluid-Structure, pp. 425–434. ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2004.08.017. URL: http://www.sciencedirect.com/science/article/pii/S0045794904004110 (cit. on pp. 14, 25).

[CG16] M. Coquerelle, S. Glockner. “A fourth-order accurate curvature computation in a level set framework for two-phase flows subjected to surface tension forces.” In: *Journal of Computational Physics* 305 (2016), pp. 838–876. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2015.11.014. URL: http://www.sciencedirect.com/science/article/pii/S0021999115007548 (cit. on p. 14).

[CHM11] C. D. Correa, R. Hero, K. L. Ma. “A Comparison of Gradient Estimation Methods for Volume Rendering on Unstructured Meshes.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.3 (Mar. 2011), pp. 305–319. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.105 (cit. on p. 16).

[EEG+16] K. Eisenschmidt, 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, Part 2 (2016). Recent Advances in Numerical Methods for Hyperbolic Partial Differential Equations, pp. 508–517. ISSN: 0096-3003. DOI: 10.1016/j.amc.2015.05.095. URL: http://www.sciencedirect.com/science/article/pii/S0096300315007195 (cit. on pp. 13, 16, 19, 23, 31, 41).

[FCD+06] M. M. Francois, S. J. Cummins, E. D. Dendy, D. B. Kothe, J. M. Sicilian, M. W. Williams. “A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework.” In: *Journal of Computational Physics* 213.1 (2006), pp. 141–173. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2005.08.004. URL: http://www.sciencedirect.com/science/article/pii/S0021999105003748 (cit. on p. 13).

[Gol05] R. Goldman. “Geometric Modelling and Differential Geometry Curvature formulas for implicit curves and surfaces.” In: *Computer Aided Geometric Design* 22.7 (2005), pp. 632–658. ISSN: 0167-8396. DOI: 10.1016/j.cagd.2005.06.005. URL: http://www.sciencedirect.com/science/article/pii/S0167839605000737 (cit. on p. 25).

[Gre04] D. Greaves. “A quadtree adaptive method for simulating fluid flows with moving interfaces.” In: *Journal of Computational Physics* 194.1 (2004), pp. 35–56. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2003.08.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999103004595 (cit. on p. 13).

[HN81] C. Hirt, B. Nichols. “Volume of fluid (VOF) method for the dynamics of free boundaries.” In: *Journal of Computational Physics* 39.1 (1981), pp. 201–225. ISSN: 0021-9991. DOI: 10.1016/0021-9991(81)90145-5. URL: http://www.sciencedirect.com/science/article/pii/0021999181901455 (cit. on pp. 13, 19).

[JY98] J. H. Jeong, D. Y. Yang. “Finite element analysis of transient fluid flow with free surface using VOF (volume-of-fluid) method and adaptive grid.” In: *International Journal for Numerical Methods in Fluids* 26.10 (1998), pp. 1127–1154. ISSN: 1097-0363. DOI: 10.1002/(SICI)1097-0363(19980615)26:10<1127::AID-FLD644>3.0.CO;2-Q (cit. on p. 13).

[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. 13, 14, 19).

[LBM+06] D. Laney, P. t. Bremer, A. Mascarenhas, P. Miller, V. Pascucci. “Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities.” In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 1053–1060. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.186 (cit. on p. 14).

[LC87] W. E. Lorensen, H. E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm.” In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422 (cit. on p. 13).

[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. DOI: 10.1007/3-540-08004-X_336 (cit. on p. 13).

[OCHJ12] H. Obermaier, F. Chen, H. Hagen, K. I. Joy. “Visualization of material interface stability.” In: *2014 IEEE Pacific Visualization Symposium* (2012), pp. 225–232. DOI: 10.1109/PacificVis.2012.6183595 (cit. on p. 14).

[OD15] M. Owkes, O. Desjardins. “A mesh-decoupled height function method for computing interface curvature.” In: *Journal of Computational Physics* 281 (2015), pp. 285–300. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2014.10.036. URL: http://www.sciencedirect.com/science/article/pii/S0021999114007189 (cit. on p. 14).

[OJ12] H. Obermaier, K. I. Joy. “Derived Metric Tensors for Flow Surface Visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), pp. 2149–2158. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.211 (cit. on pp. 14, 37).

[Pop09] S. Popinet. “An accurate adaptive solver for surface-tension-driven interfacial flows.” In: *Journal of Computational Physics* 228.16 (2009), pp. 5838–5866. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.04.042. URL: http://www.sciencedirect.com/science/article/pii/S002199910900240X (cit. on pp. 13, 14, 23–26, 28).

[RR02] Y. Renardy, M. Renardy. “PROST: A Parabolic Reconstruction of Surface Tension for the Volume-of-Fluid Method.” In: *Journal of Computational Physics* 183.2 (2002), pp. 400–421. ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7190. URL: http://www.sciencedirect.com/science/article/pii/S0021999102971901 (cit. on p. 13).

[SP00] M. Sussman, E. G. Puckett. “A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase Flows.” In: *Journal of Computational Physics* 162.2 (2000), pp. 301–337. ISSN: 0021-9991. DOI: 10.1006/jcph.2000.6537. URL: http://www.sciencedirect.com/science/article/pii/S0021999100965379 (cit. on p. 13).

[SSO94] M. Sussman, P. Smereka, S. Osher. “A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow.” In: *Journal of Computational Physics* 114.1 (1994), pp. 146–159. ISSN: 0021-9991. DOI: 10.1006/jcph.1994.1155. URL: http://www.sciencedirect.com/science/article/pii/S0021999184711557 (cit. on p. 13).

[Wij03] J. J. van Wijk. “Image based flow visualization for curved surfaces.” In: *Visualization, 2003. VIS 2003. IEEE*. Oct. 2003, pp. 123–130. DOI: 10.1109/VISUAL.2003.1250363 (cit. on p. 14).

All links were last followed on October 27, 2016.

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.

place, date, signature