

CHAPTER III

Methodology

3.1 Overview

Solving a multiphase flow problem with a one-fluid formulation consists in using a single set of the Navier-Stokes equations using variable physical properties, while the fluid-fluid interface is modelled using a source term for the interface tension force. In the front-tracking method of Tryggvason, this force field is calculated on a separate framework with a Lagrangian reference frame, which explicitly represents the position of the interface (UNVERDI; TRYGGVASON, 1992). From the perspective of the Eulerian framework, this interface represents a discontinuity and is, therefore, modelled with a representation of the Dirac delta function. This function is used for both, transferring information about the interface tension force from the Lagrangian framework to the Eulerian framework and transferring information about the velocity field from the Eulerian framework to the Lagrangian framework. An indicator function is used for updating the distribution of the physical properties without the need for solving an additional advection equation for the density and viscosity fields. This procedure can be summarized as follows:

- 1 set-up boundary conditions and initial conditions
- 2 set-up physical properties
- 3 calculate the interface force

- 4 solve the Navier-Stokes equations
- 5 update the interface position
- 6 loop over steps 2 through 5 until the end of the simulation

Although this algorithm is very simplified and no assumption is made about space or time discretization, it describes the main steps of the Tryggvason front-tracking method. In the remaining of this chapter, these steps will be expanded and detailed.

3.2 The Eulerian Framework

3.2.1 Governing Equations

In the one-fluid formulation, multiphase flows are modelled as a single phase flow with variable density and viscosity. Therefore, the same balance equations for momentum and mass are used (WHITE, 1991):

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} \right] = \nabla \cdot [\mu (\nabla \mathbf{u} + \nabla \mathbf{u}^T)] - \nabla p + \rho \mathbf{g} + \mathbf{f}_\sigma, \quad (3.1a)$$

$$\nabla \cdot \mathbf{u} = 0. \quad (3.1b)$$

Here, $\mathbf{u}$ is the fluid velocity field, $\rho(\mathbf{x}, t)$ is the fluid density, $\mu(\mathbf{x}, t)$ is the fluid viscosity, p is the pressure, including the fluid-static part, $\mathbf{g}$ is the gravity and $\mathbf{f}_\sigma$ is the interface tension force. The distribution of the density and viscosity fields along the domain is now determined by the flow field. However, instead of solving an additional advection equation for each property, an indicator function is used for associating each cell in the Eulerian domain to the respective fluid phase. Details about the implementation of the indicator function will be given in section 3.2.2. procedure for calculating the interface tension will be detailed in section 3.3.3.

There are three moments that require communication between the Lagrangian and Eulerian framework:

- The advection of the interface requires the velocity field, which is calculated in the Eulerian framework by solving the Navier-Stokes equations.
- The solution of the Navier-Stokes equations requires the interface tension force, which is calculated in the Lagrangian domain and distributed over the Eulerian framework.
- The transport of the viscosity and density fields is performed via an indicator function, rather than solving an advection equation.

Using the ideas of Peskin (1977), Unverdi and Tryggvason (1992) addressed the sharp jump in fluid properties across the interface by introducing a numerical thickness. The Dirac delta was approximated by a distribution function which satisfy some special conditions listed in (PESKIN, 2002). The 3D distribution is approximated by the product of three 1D functions, according to expression (3.2):

$$D(\mathbf{x} - \mathbf{X}) = \frac{1}{h_x h_y h_z} W\left(\frac{x - X}{h_x}\right) W\left(\frac{y - Y}{h_y}\right) W\left(\frac{z - Z}{h_z}\right), \quad (3.2)$$

where

$$W(r) = \begin{cases} \frac{1}{4}(1 + \cos(\frac{\pi}{2}r)), & r < 2, \\ 0, & r \geq 2, \end{cases} \quad (3.3)$$

and

$$r = \frac{x - X}{h_x}, \frac{y - Y}{h_y}, \frac{z - Z}{h_z}. \quad (3.4)$$

Therefore, the interface tension force, originally calculated at the Lagrangian interface, is distributed over an interface, according to Eq. (3.5):

$$\mathbf{f}(\mathbf{x}, t) = \int \mathbf{F}(\mathbf{X}, t) D(\mathbf{X} - \mathbf{x}) d\mathbf{X}, \quad (3.5)$$

where t is the time, $\mathbf{X}$ represents the position in the Lagrangian framework, $\mathbf{F}$ represents the interface tension force field in the Lagrangian framework, $\mathbf{x}$ represents the position in the Eulerian framework and $\mathbf{f}$ is the interface tension force field in the Eulerian framework.

On the other hand, the velocity field at the interface is interpolated from the Eulerian framework to the Lagrangian interface using Eq. (3.6):

$$\mathbf{U}(\mathbf{X}, t) = \int \mathbf{u}(\mathbf{x}, t) D(\mathbf{x} - \mathbf{X}) d\mathbf{x} \quad (3.6)$$

where $\mathbf{U}$ represents the velocity field in the Lagrangian framework and $\mathbf{u}$ is the velocity field in the Eulerian framework.

Numerically, the integrals (3.5) and (3.6) are replaced by a discrete summation over a four cells-thick zone around the interface.

3.2.2 Indicator Function - Closest Point Transform

The Closest Point Transform (CPT) was developed by Mauch (2003) with the objective of transforming an explicit representation of a manifold embedded in a grid into an implicit one, the implicit representation being the distance field stored on the grid. Conversely, finding the zero-distance iso-value in this distance field yields the surface of the manifold.

In the context of a front-tracking algorithm, the manifold represents the interface between two fluid phases. Only closed interfaces will be addressed here, but the method could be expanded to open interfaces as well. The basic concepts will be first explained for a two-dimensional domain for the sake of clarity and then expanded for a three-dimensional domain

Two-dimension meshes

A two-dimensional interface, after being discretized, is composed by vertices and elements, the latter being usually line segments. Therefore, if x is the closest point of a grid with relation to a mesh C , this point will be either close to an element or to a vertex.

Figure (3.1) shows the situation in which a generic point $x \in \mathbb{R}^2$ is bounded by the normals at the two vertices of the element (v_1 and v_2). Since the point lies outside the domain delimited by C , the outer normals are used, as shown in the picture on the left. If the point were inside the

domain, the inner normals would have been used, as shown in the image on the right. In either case, the shortest distance between x and C will be calculated via a point-to-line distance formula.

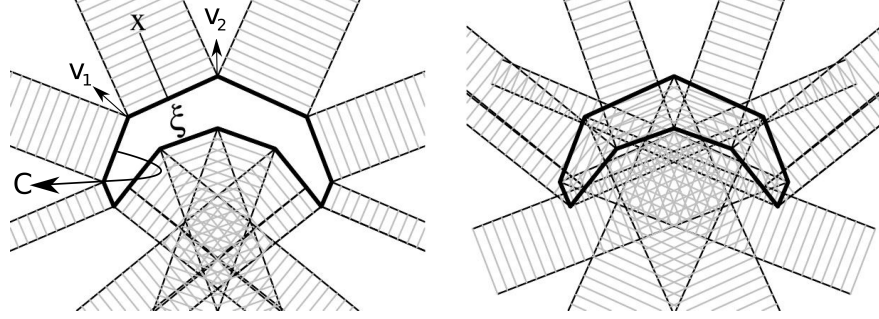


Figure 3.1: A closed curve C and the regions in which the closest points not in C are orthogonal to an element of the mesh. Left: regions delimited by the outer normals. Right: regions delimited by the inner normals (MAUCH, 2003).

Conversely, Figure (3.2) shows a case in which the point x is bounded by the normals of two neighbour elements sharing the vertex ξ . In this case, the distance from x to C will be calculated via a point-to-point distance formula.

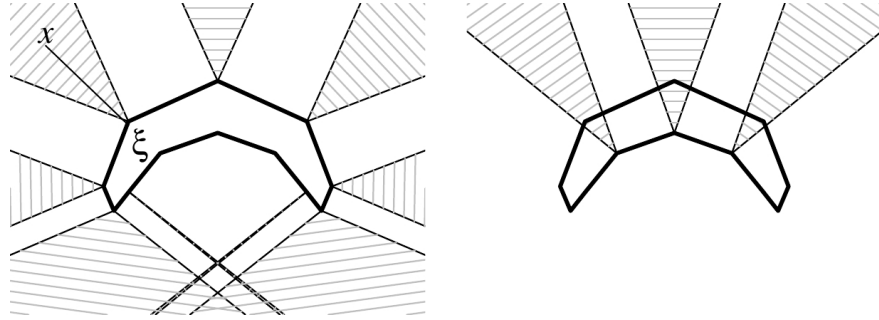


Figure 3.2: The bounding regions are determined by the normals of neighbour elements sharing a vertex. Left: normals point outside C . Right: normals point inside C (MAUCH, 2003).

In the context of a front-tracking method, this means that each geometric primitive in the mesh C , be it a vertex or an element, must have the information about the coordinates of the Eulerian grid which surrounds that primitive. Then, the distance between the primitives and the grid points located inside their bounding polygons as defined above is computed. Since there may be a polygon overlap in some situations, as depicted in Fig. (3.2), only the smallest value calculated for a given grid point is stored.

Three-dimension meshes

The procedure described above holds for both, two-dimensional and three-dimensional domains. However, in three-dimensions, the interface is composed by three kinds of geometric primitives: vertices, edges and elements. Therefore, the distance between any point in 3D space and the surface may be calculated by one of three expressions:

- point-to-point distance;
- point-to-curve distance;
- point-to-surface patches distance.

Also, the bounding regions in 3D will be polyhedra instead of polygons. Figure (3.3) shows the bounding polyhedra associated to a triangular element. In a procedure similar to the 2D case, the bounding region is limited by the normal of the element at its vertices. The height of the polyhedra will depend on the numerical thickness adopted for the interface.

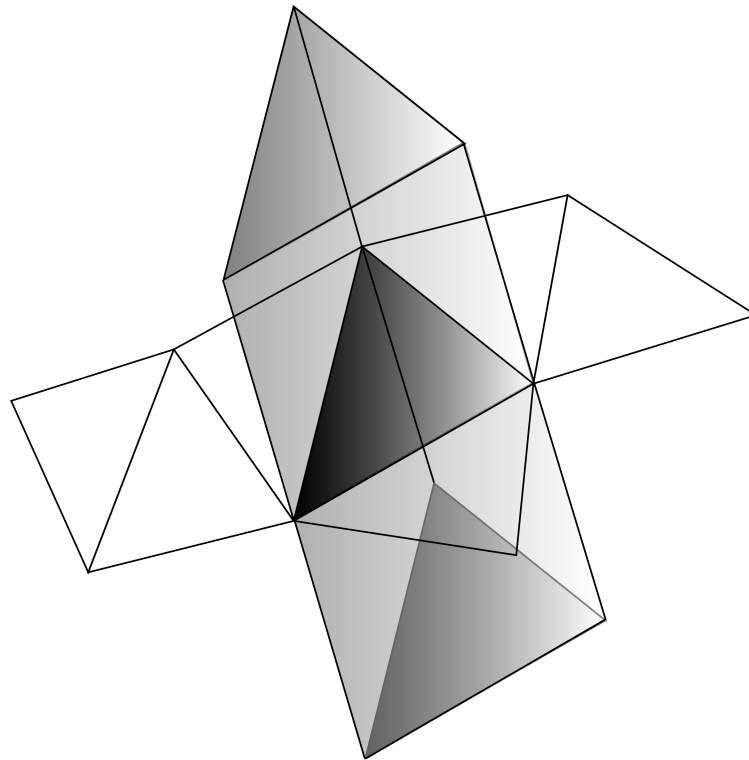


Figure 3.3: Delimiting prism for the shortest distance from a point in space and the face of an element of a mesh.

The shape of the polyhedra associated to edges and vertices is shown in Fig. (3.4). Since in a closed surface any edge is shared by 2 faces, the bounding region will be the wedge determined by the edge itself and the normals of the two neighbouring faces. Finally, in the polyhedron associated to a vertex of the mesh, the bounding region will have a pyramid shape, delimited by the bounding regions of the surround edges and faces. These two scenarios are shown in Fig. (3.4).

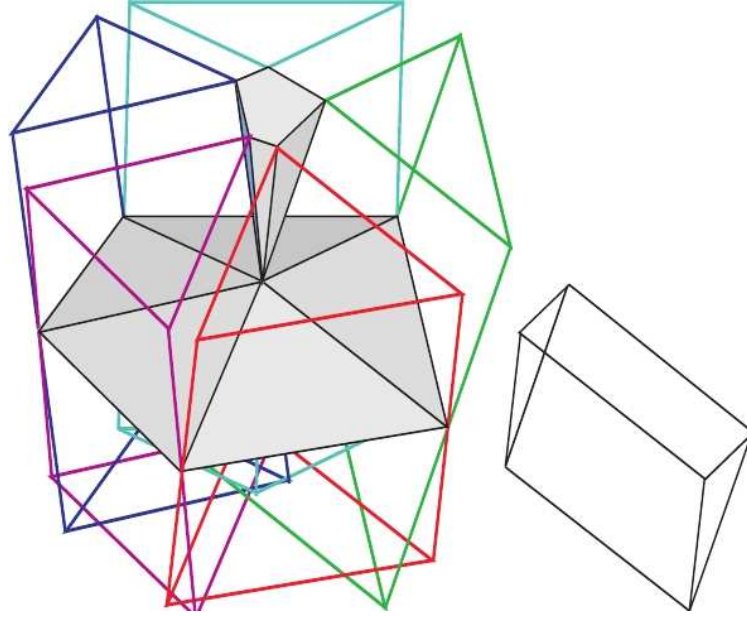


Figure 3.4: CPT Bounding regions for edges and vertices (AZEREDO, 2007).

The height of the polyhedra depends on the region of interest around the interface. In the case of the front-tracking method, it will be the same as the thickness adopted for the distribution function used for replacing the Dirac delta function. With regards to their shape, performing search tasks inside such irregular shaped prisms can be an expensive operation. Therefore, these polyhedra will be replaced by the smallest cartesian bounding box associated to them. Also, regardless of the shape of the bounding polyhedron there may be an overlapping between two polyhedra, as shown in Figs. (3.2) and (3.1). This feature requires initializing the Eulerian cells properly, comparing the current value to the newly computed value and storing the smallest tone.

Computing the physical properties.

The distance field calculated by the indicator function is limited to an interval $[-\gamma, \gamma]$, where γ is the absolute value of the largest distance from the interface to a given point in the Eulerian framework. In order to use the indicator function to calculate the distribution of the physical

properties over the Eulerian domain, a Heaviside function is applied to the indicator function field in order to map it onto the interval $[0, 1]$. Following the idea of a finite thickness interface, a smooth version of the Heaviside function was used ((SETHIAN; SMEREKA, 2003),(YOKOI, 2008)):

$$H(\varphi) = \begin{cases} 1, & \varphi > \gamma, \\ 0.5(1 + \frac{\varphi}{\gamma} + \frac{1}{\pi} \sin(\frac{\pi\varphi}{\gamma})), & \|\varphi\| \leq \gamma, \\ 0, & \varphi < -\gamma. \end{cases} \quad (3.7)$$

After being mapped, the indicator function can be used for calculating the distribution of the density (ρ) and viscosity (μ) fields follows:

$$\rho(\varphi) = H(\varphi)\rho_1 + (1 - H(\varphi))\rho_2, \quad (3.8)$$

$$\mu(\varphi) = H(\varphi)\mu_1 + (1 - H(\varphi))\mu_2.$$

3.2.3 Temporal Discretization

The time discretization scheme presented here is similar to the scheme presented in (VILLAR, 2007) with a few changes, such that various semi-implicit schemes can be chosen by setting up values for some constants. The second-order semi-implicit scheme is given by:

$$\frac{\rho^{n+1}(\phi)}{\Delta t}(\alpha_2 \mathbf{u}^{n+1} + \alpha_1 \mathbf{u}^n + \alpha_0 \mathbf{u}^{n-1}) = \beta_1 f(\mathbf{u}^n) + \beta_0 f(\mathbf{u}^{n-1}) + \quad (3.9a)$$

$$\lambda [\theta_2 \nabla^2 \mathbf{u}^{n+1} + \theta_1 \nabla^2 \mathbf{u}^n + \theta_0 \nabla^2 \mathbf{u}^{n-1}] - \nabla p^n + \rho^{n+1} \mathbf{g}, \quad (3.9b)$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0,$$

where $\lambda = \|\mu\|_\infty$ and $f(\mathbf{u})$ depends on the diffusive, advective and forcing terms:

$$f(\mathbf{u}) = -\lambda \nabla^2 \mathbf{u} + \nabla \cdot [\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)] - \mathbf{u} \cdot \nabla \mathbf{u} + \mathbf{f}_\sigma \quad (3.10)$$

The values of α_i , β_i and θ_i are:

$$\begin{aligned}
 \alpha_0 &= \frac{(2\gamma - 1)\omega^2}{1 + \omega}, \\
 \alpha_1 &= (1 - 2\gamma)\omega - 1, \\
 \alpha_2 &= \frac{1 + 2\gamma\omega}{1 + \omega}, \\
 \beta_1 &= -\gamma\omega, \\
 \beta_0 &= 1 + \gamma, \\
 \theta_0 &= \frac{c}{2}, \\
 \theta_1 &= 1 - \gamma - \left(1 + \frac{1}{\omega}\right)\frac{c}{2}, \\
 \theta_2 &= \gamma + \frac{c}{2\omega},
 \end{aligned} \tag{3.11}$$

where $\omega = \Delta t_{n+1}/\Delta t_n$ is the ratio between two consecutive time steps. A family of numerical schemes involving the two parameters γ and c can be derived as showed by Ascher, Ruuth and Wetton (1997):

- Crank-Nicholson Adams-Bashforth (CNAB): $(\gamma, c) = (0.5, 0.0)$;
- Modified Crank-Nicholson Adams-Bashforth (MCNAB): $(\gamma, c) = (0.5, 0.125)$;
- Crank-Nicholson Leap Frog (CNLF): $(\gamma, c) = (0.0, 1.0)$;
- Semi-Backward Difference (SBDF): $(\gamma, c) = (1.0, 0.0)$.

In the present work the above SBDF temporal time discretization is chosen in all cases simulated here with a variable size of time step.

Applying a fractional step approach to Eqs. (3.9a) and (3.9b) yields:

$$\frac{\rho^{n+1}}{\Delta t} (\alpha_2 \mathbf{u}^* + \alpha_1 \mathbf{u}^n + \alpha_0 \mathbf{u}^{n-1}) = \beta_1 f(\mathbf{u}^n) + \beta_0 f(\mathbf{u}^{n-1}) + \tag{3.12}$$

$$\lambda \left[\theta_2 \nabla^2 \mathbf{u}^* + \theta_1 \nabla^2 \mathbf{u}^n + \theta_0 \nabla^2 \mathbf{u}^{n-1} \right] - \nabla p^n + \rho^{n+1} \mathbf{g},$$

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\alpha_2} \frac{\nabla q}{\rho^{n+1}}, \tag{3.13}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0. \tag{3.14}$$

Once the *provisional velocity* $\mathbf{u}^*$ is computed from (3.12) imposing $\mathbf{u}^* \doteq \mathbf{u}^{n+1}$ on the boundaries, it is *projected* onto the space of divergence-free vector fields. This is accomplished by solving the

Poisson equation for q defined by (3.13) and (3.14) along with homogeneous Neumann boundary conditions $\partial q / \partial \mathbf{n} = 0$, where $\mathbf{n}$ is the outer normal of the domain.

Since a fractional time step is used, multilevel-multigrid methods must be solved both for the provisional vector field, $\mathbf{u}^*$ in (3.12), and for the pressure increment q , which is given by:

$$\nabla \cdot \left[\frac{1}{\rho^{n+1}} \nabla q \right] = \frac{\alpha_2}{\Delta t} \nabla \cdot \mathbf{u}^* \quad (3.15)$$

The time step is calculated following stability criteria for explicit schemes, taking into account the advective, diffusive and capillary terms, as defined by Eq. (3.16):

$$\Delta t_{adv} = \frac{1}{\frac{\|u\|_\infty}{\Delta x} + \frac{\|v\|_\infty}{\Delta y} + \frac{\|w\|_\infty}{\Delta z}}, \quad (3.16a)$$

$$\Delta t_{diff} = a_1 h, \quad (3.16b)$$

$$\Delta t_{cap} = \sqrt{\left(\frac{\rho_C + \rho_D}{2} \right) \frac{h^3}{\pi \sigma}}, \quad (3.16c)$$

$$\Delta t = \min(b_1 \Delta t_{adv} + b_2 \Delta t_{diff} + b_3 \Delta t_{cap}). \quad (3.16d)$$

where Δt_{adv} , Δt_{diff} and Δt_{cap} are the maximum time step allowed by the advective term, the diffusive term and the interface force, respectively. $\|u\|_\infty$, $\|v\|_\infty$ are $\|w\|_\infty$ values of the infinite-norm for the u , v and w components of the velocity; $h = \min(\Delta x, \Delta y, \Delta z)$ is the characteristic length of the grid and Δx , Δy and Δz are the grid spacing in the x-, y- and z- directions. $0 < a_1 \leq 1$, $0 < b_1 \leq 1$, $0 < b_2 \leq 1$ and $0 < b_3 \leq 1$ are safety coefficients. Notice that, in order to expression (3.16b) can be used, a_1 must have dimensions of time per length, that is: $[a_1] = TL^{-1}$.

According to Tryggvason, Scardovelli and Zaleski (2011), the constraint for second order schemes for the diffusive term is given by

$$\Delta t = \frac{\rho h^2}{6\mu}. \quad (3.17)$$

Making $\Delta t_1 = a_1 h$ and $\Delta t_2 = \frac{\rho h^2}{6\mu}$ yields the following relation:

$$\frac{\Delta t_2}{\Delta t_1} = h \frac{\rho}{6a_1\mu} \quad (3.18)$$

Since $a_1 \leq 1$ and, in the context of gas-liquid flows $\rho \gg 1$ and $\mu < 1$, relation (3.18) shows that the constraint adopted in this work is stricter than Eq. (3.17). The same can be said about the capillary constraint since, in the present work, since the mean density is adopted.

3.2.4 Spatial Discretization

Space is discretized using a SAMR framework, which is based on the version of the Immersed Boundary (IB) method introduced by Roma, S. and J. (1999), and in the hierarchical grid structure proposed by Berger and Colella (BERGER; COLELLA, 1989). In this scheme, regions of the flow bearing special interest are covered by block-structured grids, defined as a hierarchical sequence of nested, progressively finer levels (*composite grids*). Each level is formed by a set of disjoint rectangular grids and the refinement ratio between two successive refinement levels are constant and equal to two. Ghost cells are employed around each grid, for all the levels, and underneath fine grid patches to formally prevent the finite difference operators from being redefined at grid borders and at interior regions which are covered by finer levels. Values defined in these cells are obtained from interpolation schemes, usually with second or third order accuracy, and not from solving the equations of the problem (ROMA, 1996).

A staggered composite grid is used, i.e., pressure and other scalar variables are computed at the centers of the computational cells and vector variables are located at the cell faces. The discretisations of the Laplacian, gradient and divergence differential operators are performed by standard, cell-centered second order stencils.

Although a variable can be defined or initialized in any level, in current work the Lagrangian interface must to be completely covered by the finest level, ensuring that the most important physical phenomena are being captured.

3.2.5 Structured Adaptive Mesh Refinement

The Structured Adaptive Mesh Refinement (SAMR) method (BERGER; OLIGER, 1984), (BERGER; COLELLA, 1989) consists on creating a hierarchy of cartesian grids with various levels of refinement covering the whole domain, concentrating the finer meshes on the regions which require special attention. Within the grid hierarchy, the levels are nested so that the coarsest level covers the entire computational domain and each finer level covers a part of the interior of the previous coarse level. In each level, the grid is composed by the union of logically rectangular regions called patches or blocks, which do not intersect each other. All grid blocks in a given refinement level share the same grid spacing.

The refinement ratio (r) is defined as the quotient between the grid spacing of two consecutively refined levels and is constant for all levels. In this work, $r = 2$ is used. Also, the grid hierarchy must obey two conditions in order to be properly nested:

1. the corners of a block in a given level of refinement must match the corners of cells located at the next coarser level.
2. A grid block in a given level must not touch the boundary of a block belonging to a coarser level of refinement, except when these blocks lie at the boundaries of the computational domain.

Figure (3.5) shows the case of a properly nested grid hierarchy with three levels of refinement and Fig. (3.6) shows two grids violating these conditions: grid (a) violates condition 1 and grid (b) violates condition 2.

Grid adapting involves selecting cells to be refined at a given level, using some error estimator or any refinement criterion, and then grouping the cells into blocks in that region. These blocks are used for creating the next refinement level in the hierarchy. When the remeshing criteria are based on the magnitude of some variable, ranges of this variable are defined so that each refinement level covers a different range of values of that variable. In this work, the following approach was followed:

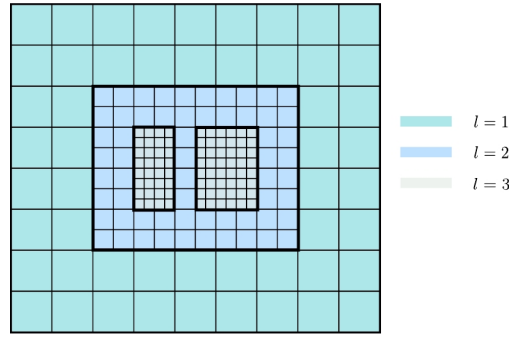


Figure 3.5: An example of block structured adaptive mesh refinement properly nested (VILLAR, 2007).

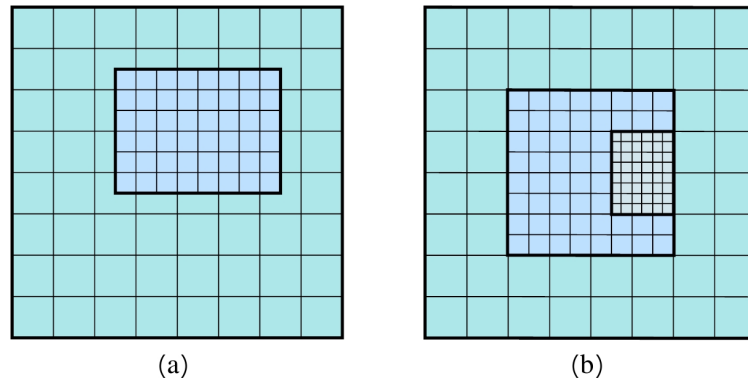


Figure 3.6: An example of block structured adaptive mesh refinement violating the nesting rules: the fine block in grid (a) does not lie on vertices of the coarse level block and the block in the finest level in grid (b) touches the boundary of a block in the next coarse level which is not part of the domain boundary (VILLAR, 2007).

Let the remeshing criterion be based on the magnitude of some variable $\Phi \in [\Phi_{min}.. \Phi_{max}]$. This variable is mapped onto the interval $[0..1]$, which is further divided into as many intervals as the number of refinement levels, so that the finer refinement levels are applied on the top of the interval. Determining the range for each refinement level may depend on the kind of problem being solved. In this work, the following rule was adopted: the finest level covers the interval $[0.1..1.0]$ and the complement of the domain is equally divided between the remaining refinement levels.

Figure (3.7) shows an example of SAMR grid taken from a case simulating a bubble rising in the wobbling regime. The bubble creates a von Karman wake and the mesh refinement is placed over the bubble and over the bubble wake.

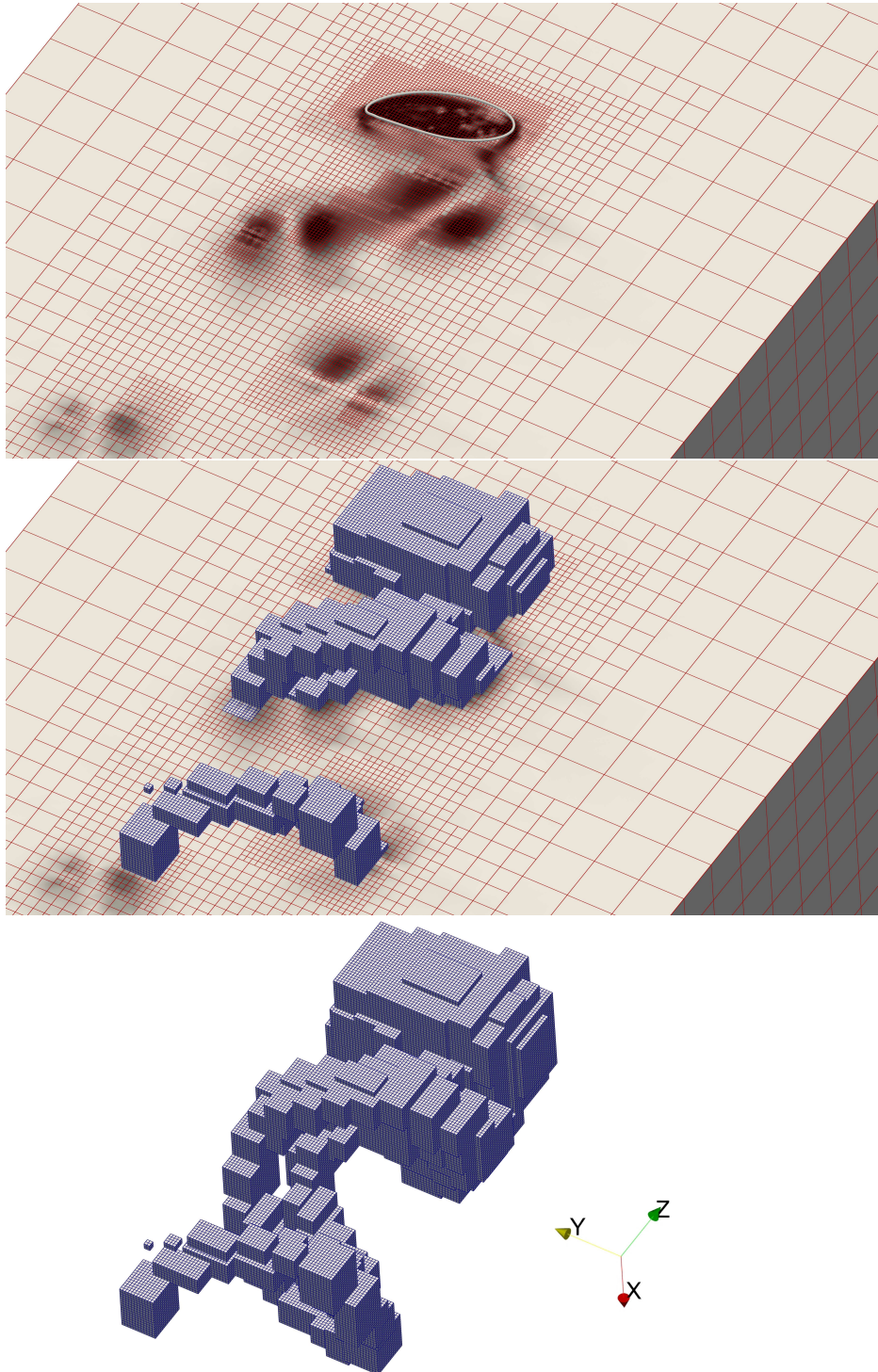


Figure 3.7: Example of adaptive mesh refinement using based on the vorticity magnitude.

3.3 The Lagrangian framework

This section describes the operation performed in the Lagrangian framework, in the sequence as they are executed in the algorithm.

3.3.1 Interface advection

The interface advection is performed in a Lagrangian manner. After the velocity field is found by solving the Navier-Stokes equations, this velocity field is interpolated on the Lagrangian interface. The interface position is then updated by solving the ODE given by

$$\frac{d\mathbf{X}(t)}{dt} = \mathbf{U}(\mathbf{X}(t), t). \quad (3.19)$$

where $\mathbf{X}(t)$ is the position of the Lagrangian interface and $\mathbf{U}(\mathbf{X}(t), t)$ is the velocity of the interface. In the first time step, Eq. (3.19) is solved using the first order Euler method, given by:

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \cdot \mathbf{U}^n(\mathbf{X}^n). \quad (3.20)$$

After solving the first time step, the Navier-Stokes solver enters the temporal loop. In this situation, a second order time integration scheme may be used, since the temporal history for both, velocity and position of the interface, can be stored. Therefore, the following integration method is used:

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \frac{\Delta t}{2} \cdot (\mathbf{U}^n(\mathbf{X}^n) + \mathbf{U}^{n-1}(\mathbf{X}^{n-1})). \quad (3.21)$$

Notice that the term $\mathbf{U}^{n-1}$ can always be obtained, even after a remeshing operation since, in such situation, it can be restored via interpolation of $\mathbf{u}^{n-1}$ on the corresponding position of the Lagrangian interface.

There are two issues that arise as result of the interface advection: the growth of small undulations in some regions where the interface area tend to shrink (SOUSA et al., 2004) and the volume change inside the interface. These issues will be addressed next.

3.3.2 Volume and shape preserving after advecting the interface

A mass conservation issue in Front Tracking arise after the advection of the interface. The most common reason pointed out for this problem is the fact that the interpolation procedure for calculating the velocity field at the interface is not conservative. Another reason, which will be shown in the presentation of the results, is the use of low order schemes for advecting the in-

terface. Regardless of the origin of this issue, the volume change at a given time step is small ($O(10^{-4})\%$). However, on the overall simulation it has a cumulative behaviour and, after running tens of thousands of time steps, the volume change can easily reach $O(10^0)\%$.

This problem can be solved by applying a correction step after updating the interface position in order to correct the interface volume. Since the volume change in a given time step is small, all change can be assumed to happen in the normal direction. Moreover, the expansion or shrink is assumed to be constant along the interface, as shown in Fig. (3.8).

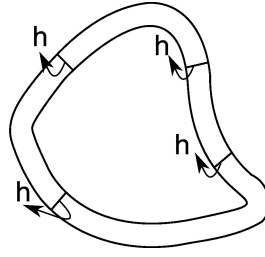


Figure 3.8: The volume change after advecting the interface is assumed to be due to an expansion/contraction of the geometry in the normal direction.

Given a mesh element on the interface, the volume associated to its expansion/shrink is given by

$$V = A_T \cdot h \quad (3.22)$$

where h is the length of the expansion in the normal direction, as shown in Fig. (3.9). A_T is the area of the triangle, which is assumed constant at both, the base and the top of the prism.

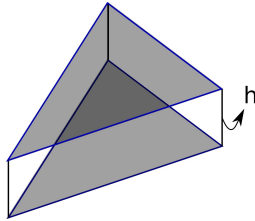


Figure 3.9: The prism conceptually formed by a typical interface element and the expansion gap h after the interface advection.

Therefore, the change in the interface volume is obtained by summing the contributions of all elements in the mesh:

$$\Delta V = \sum_{i=1}^{N_t} A_{T_i} \cdot h = A_S \cdot h \quad (3.23)$$

where A_S is the total area of the interface and N_t is the number of elements in the interface mesh. On the other hand, the volume change is simply the difference between the current and initial volume, making h the only unknown in Eq. (3.23). Therefore, the expansion gap h is given by:

$$h = \frac{\Delta V}{A_S} \quad (3.24)$$

That is, the volume correction can be achieved by just expanding or shrinking the surface in the normal direction by a factor equal to h . The volume correction is then given by Eq. (3.25):

$$\mathbf{X}_{\text{new}} = \mathbf{X}_{\text{old}} - h \cdot \mathbf{n}_v, \quad (3.25)$$

where $\mathbf{X}_{\text{old}}$ and $\mathbf{X}_{\text{new}}$ are the original and new coordinates of the mesh vertices and $\mathbf{n}_v$ is an approximation for the vertex normal, given by

$$\mathbf{n}_v = \sum_{i=1}^{N_t} \frac{\mathbf{n}_i \sin(\alpha_i)}{\|e_{i1}\| \|e_{i2}\|} \quad (3.26)$$

where N_t is the number of triangles which share the vertex. Figure (3.10) depicts the meaning of each term of the above equation: $\mathbf{n}_i$ is the normal of element i , e_{i1} and e_{i2} are the edges of element i which are employed to compute its normal, and α_i is the angle between these two edges (JIN; LEWIS; WEST, 2005).

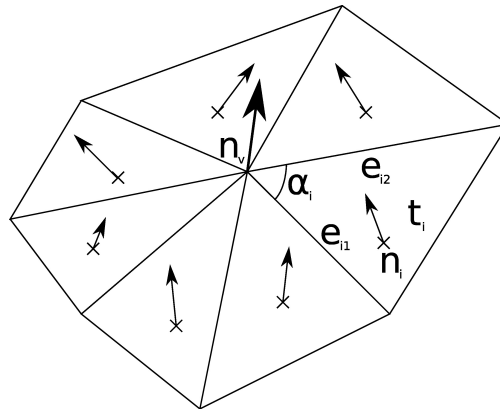


Figure 3.10: Computing a normal vector at vertex v . For each element i connected to v , $\mathbf{n}_i$ is the normal of this element, e_{i1} and e_{i2} are the edges of the element which are used for computing $\mathbf{n}_i$ and α_i is the angle between e_{i1} and e_{i2} .

is well known. Another common problem in this methodology is the development of small undulations or wrinkles that occurs at the rear of rising bubbles. According to Sousa et al. (2004), these wrinkles are a result of variations in the velocity field and may be amplified where the surface area is shrinking. Figure (3.11) shows two examples of this phenomenon, both of them depicting rising bubbles in a quiescent liquid. The image on the left shows the rear of a bubble in the spherical cap regime ($Eo = 40$, $M = 0.056$, $Re = 20.6$). The bubble on the right shows a result presented by Annaland, Deen and Kuipers (2006) and depicts the terminal shape of a bubble rising in a wobbling regime ($Eo = 10.0$, $M = 9.71 \cdot 10^{-12}$, $Re = O(10^3)$). Therefore, although the undulation issue is stronger at high Re number, it also appears at laminar flows.



Figure 3.11: Wrinkles at the rear of rising bubbles. Left: Spherical cap, $Eo = 40$, $M = 0.056$. Simulation carried out in the present Work. Right: Intermediate wobbling + spherical cap, $Eo = 10.0$, $M = 9.71 \cdot 10^{-12}$ (ANNALAND; DEEN; KUIPERS, 2006).

Sousa et al. (2004) described an algorithm named TSUR-3D (TSUR: Trapezoidal Sub-grid Undulation Removal) which eliminates such undulations by requiring that the normal of an edge, in the plane defined by the normals at the vertices of the edge, be parallel to the resultant of the normals of these vertices. This is done by computing new coordinates for the vertices of the edge while taking into account local volume preservation.

Figure (3.12) shows the result of the algorithm for a given edge e and its vertices, named $\mathbf{v}_1$ and $\mathbf{v}_2$. If $\mathbf{n}_1$ and $\mathbf{n}_2$ are the normal vectors at $\mathbf{v}_1$ and $\mathbf{v}_2$, an approximation of the normal vector for

e can be given by

$$\mathbf{n} = \frac{(\mathbf{n}_1 + \mathbf{n}_2)}{\|(\mathbf{n}_1 + \mathbf{n}_2)\|}. \quad (3.27)$$

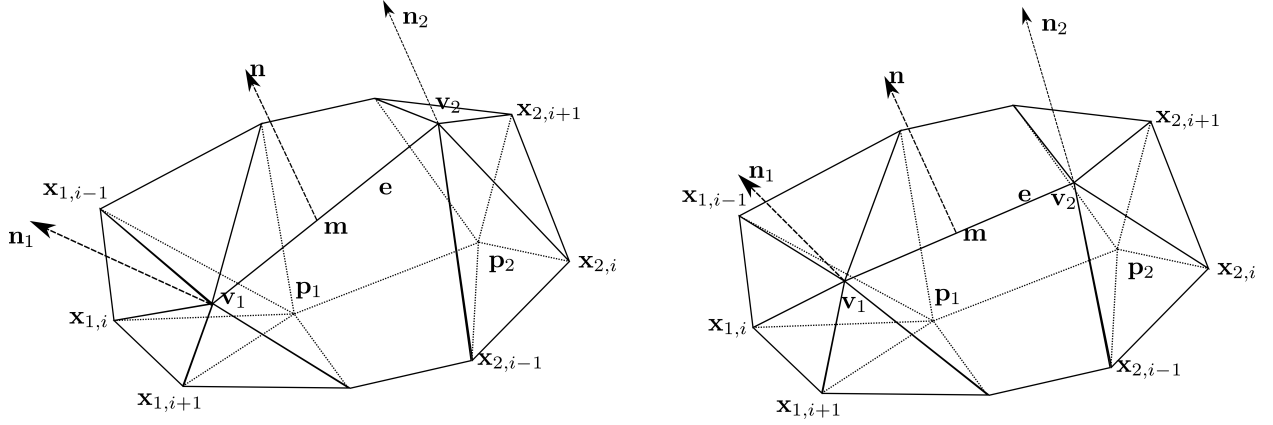


Figure 3.12: Undulation Removal procedure: repositioning a generic edge e (SOUSA et al., 2004).

Let $\mathbf{m}$ be the midpoint of e . The deviation between $\mathbf{n}$ and the correct normal at $\mathbf{m}$ may be computed by h_1 and h_2 as follows:

$$h_1 = (\mathbf{v}_1 - \mathbf{m}) \cdot \mathbf{n} \quad (3.28a)$$

$$h_2 = (\mathbf{v}_2 - \mathbf{m}) \cdot \mathbf{n} \quad (3.28b)$$

Next, auxiliary points $\mathbf{p}_1$ and $\mathbf{p}_2$ are calculated as

$$\mathbf{p}_1 = \mathbf{v}_1 - h_1 \mathbf{n} \quad (3.29a)$$

$$\mathbf{p}_2 = \mathbf{v}_2 - h_2 \mathbf{n} \quad (3.29b)$$

defining two polyhedra: $P_1(\mathbf{v}_1, \mathbf{x}_{10}, \dots, \mathbf{x}_{1n}, \mathbf{p}_1)$ with volume V_1 and $P_2(\mathbf{v}_2, \mathbf{x}_{20}, \dots, \mathbf{x}_{2n}, \mathbf{p}_2)$ with volume V_2 .

Using $\mathbf{p}_1$ and $\mathbf{p}_2$ as the new coordinates of $\mathbf{v}_1$ and $\mathbf{v}_2$ fulfils the objective of matching the normals. However, the local volume conservation has not been taken into account yet. To accomplish that, an average correction will be applied to the coordinates of both vertices, following the direction of $\mathbf{n}$.

Assume the volumes V_1 and V_2 are computed based on two non-intersecting plane polygons:

S_1 , with area A_1 and S_2 , with area A_2 , such that

$$V_1 = A_1 h_1 \text{ and } V_2 = A_2 h_2. \quad (3.30)$$

Let $P = P_1 \cup P_2$ be a polyhedron with volume

$$V = Ah \quad (3.31)$$

were $A = A_1 + A_2$ and $V = V_1 + V_2$. Using these relations and Eqs. (3.30) and (3.31), one may write:

$$\frac{V}{h} = \frac{V_1}{h_1} + \frac{V_2}{h_2} \quad (3.32)$$

which leads to the following expression for h :

$$h = \frac{V_1 + V_2}{\frac{V_1}{h_1} + \frac{V_2}{h_2}}. \quad (3.33)$$

Finally, the new positions of $\mathbf{v}_1$ and $\mathbf{v}_2$ are given by:

$$\mathbf{v}_1 = \mathbf{p}_1 + h\mathbf{n} \quad (3.34a)$$

$$\mathbf{v}_2 = \mathbf{p}_2 + h\mathbf{n} \quad (3.34b)$$

This procedure is summarized in algorithm 1.

3.3.3 Interfacial tension force

The force due to the surface tension acting on a surface element δA is given by

$$\delta F_\sigma = \int_{\delta A} \sigma \kappa \mathbf{n} dA \quad (3.35)$$

where σ is the surface tension coefficient, κ is the mean curvature and $\mathbf{n}$ is the outer unit normal.

The mean curvature can be written as $\kappa \mathbf{n} = (\mathbf{n} \times \nabla) \times \mathbf{n}$ (TRYGGVASON et al., 2001). Replacing

Algorithm 1 Small Undulation Removal - TSUR3D

Input:

- e : edge to be smoothed
 - $\mathbf{v}_1, \mathbf{v}_2$: vertices belonging to e .
 - $\mathbf{x}_{i1}, \mathbf{x}_{i2}$: vertices linked to $\mathbf{v}_1$ and $\mathbf{v}_2$ in the mesh.
1. Calculate $\mathbf{n}_1$ and $\mathbf{n}_2$.
 2. Calculate $\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2}{\|\mathbf{n}_1 + \mathbf{n}_2\|}$.
 3. Calculate $\mathbf{m}$.
 4. Calculate $h_1 = (\mathbf{v}_1 - \mathbf{m}) \cdot \mathbf{n}$ and $h_2 = (\mathbf{v}_2 - \mathbf{m}) \cdot \mathbf{n}$.
 5. Calculate $\mathbf{p}_1 = \mathbf{v}_1 - h_1 \mathbf{n}$ and $\mathbf{p}_2 = \mathbf{v}_2 - h_2 \mathbf{n}$.
 6. Calculate V_1 and V_2 .
 7. Calculate $h = \frac{V_1 + V_2}{V_1/h_1 + V_2/h_2}$.
 8. $\mathbf{v}_1 = \mathbf{p}_1 + h\mathbf{n}$, $\mathbf{v}_2 = \mathbf{p}_2 + h\mathbf{n}$.
-

this identity on Eq. (3.35) and applying the Stokes theorem yields:

$$\delta F_\sigma = \oint_{\delta\Gamma} \sigma \mathbf{t} \times \mathbf{n} d\Gamma \quad (3.36)$$

where $d\Gamma$ is the boundary of the integration element. $\mathbf{t}$ is the unit tangent and $\mathbf{n}$ is the unit normal, both computed at the element boundary, leading to the following discretized equation:

$$\mathbf{F}_m = \sum_j \sigma (\mathbf{t}_j \times \mathbf{n}_j) \quad (3.37)$$

where $\mathbf{F}_m$ is the force vector acting on element m , $\mathbf{t}_j$ is the tangent defined by edge j of the element m and $\mathbf{n}_j$ is an approximation to the unit normal vector calculated on the edge j . Following the approach of Singh and Shyy (2007), $\mathbf{n}$ is computed as the vector sum of the normals of both elements sharing edge j .

3.3.4 Surface remeshing

Surface remeshing comprises refining and coarsening procedures. Mesh refinement is a relatively simple task. After the elements are flagged for refinement, this procedure consists in splitting the existing elements or their edges until the required characteristic length is achieved. It causes no change in the geometry shape and may lead to some decrease in the mesh quality, depending on the splitting algorithm.

Mesh coarsening, on the other hand, replaces an edge or an element by a vertex, and therefore has the potential ability to change the geometry shape. Also, the volume and/or surface area of the geometry must be preserved.

In the present work, the Memoryless Simplification algorithm (MSA) of Lindstrom and Turk (1998) will be used. The main feature of this method is that it preserves volume, shape and element quality while no time history of the mesh is necessary. This is an interesting feature, especially when large meshes are used.

The method is based on edge removal rather than vertex or element removal. The advantage of this approach is that it removes always the same number of geometric entities: one vertex, two faces and three edges, as illustrated in Fig. (3.13).

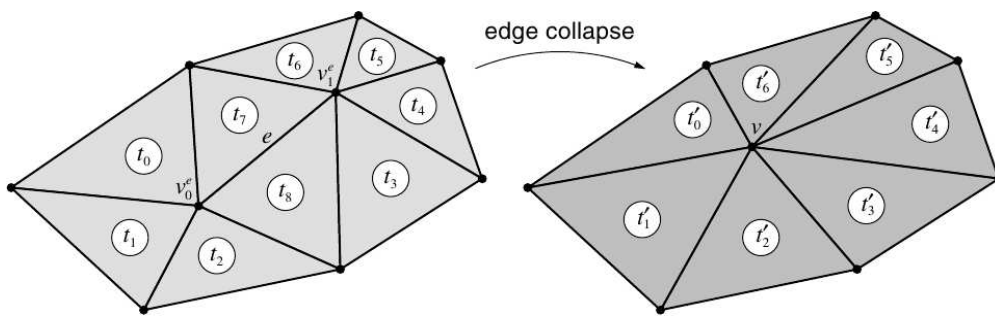


Figure 3.13: Edge collapse in the Memoryless Simplification algorithm of Lindstrom and Turk (1998).

The two most important parameters in this algorithm are the position of the new vertex and the order in which the edges are removed. MSA uses the geometry volume and area to make such decisions. The vertex position is so that the local volume is not affected by the collapse. If the new vertex is close to a geometric border, the local area is also preserved. The volume or area which are modified by the neighbouring triangles are also taken into account as well as the element quality

(LINDSTROM; TURK, 1998; LINDSTROM; TURK, 1999). Vertex positioning is treated as an optimization problem and the element shape, the local volume and area are constraints limiting the space for calculating the new coordinates. This process is illustrated in Fig. (3.14).

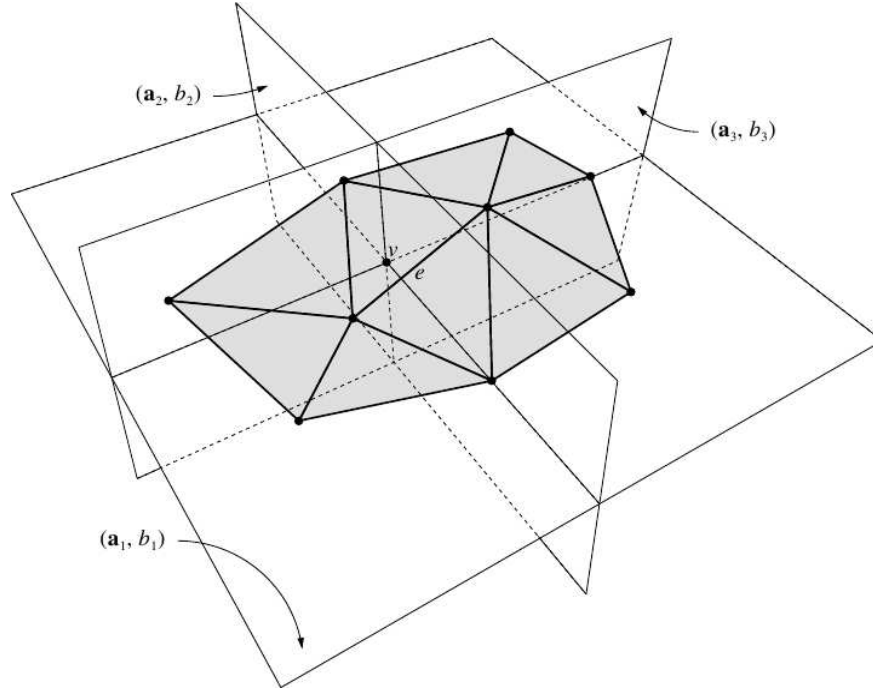


Figure 3.14: Constraints for calculating the coordinates of a new vertex in the Memoryless Simplification algorithm of Lindstrom and Turk (1998).

Remeshing criteria

Since the interface is modelled with a Dirac delta function and requires interpolation or spreading of some variables, it must be ensured that there is at least one Lagrangian vertex inside a given Eulerian cell at the level that contains the interface. Therefore, the trigger that starts the remeshing process is a comparison of the characteristic length in both meshes. In the Eulerian framework this is defined as the minimum grid spacing between the x-, y- and z- directions. In the Lagrangian framework, it may be the mean or maximum edge length, depending on the remeshing operation. Mesh refinement is performed based on the maximum value found for the edge length in the lag mesh. Small elements, on the other hand, are not as harmful. Therefore, mesh coarsening is triggered by the mean edge length.

3.4 The complete algorithm

The steps described previously in this chapter are grouped now in a procedure for solving the Navier-Stokes equations using the front-tracking method of Tryggvason. The result is depicted in algorithm (2).

Algorithm 2 Navier Stokes solver

Input:

- Eulerian domain.
 - Number of levels of refinement.
 - Physical properties.
 - Lagrangian domain.
 - Initial and boundary conditions.
 - Initial time step and total simulation time.
1. Apply initial and boundary conditions.
 2. Calculate the indicator function and the initial position of the physical properties.
 3. Calculate the interface tension force field and distribute it to the Eulerian domain.
 4. Solve the Navier-Stokes equations.
 5. Interpolate the velocity field to the Lagrangian interface.
 6. Update the position of the Lagrangian interface.
 7. Apply the TSUR-3D algorithm on the interface.
 8. Apply the volume recovery algorithm on the interface.
 9. Remesh the interface if necessary.
 10. Calculate the indicator function and the new position of the physical properties.
 11. Calculate the interface tension force and distribute it to the Eulerian domain.
 12. Solve the Navier-Stokes equations.
 13. Calculate the new time step
 14. Update elapsed time. If elapsed time is greater than the total simulation time stop.
 15. Return to step 5
-

3.5 Closure

Using a one-fluid formulation is a good way to separate the Navier-Stokes solver from the fluid-fluid interface modelling. Based on that, in this work an existing code was employed as the solver for the Navier-Stokes equations and emphasis was kept on the development of a framework for representing the fluid-fluid interface in a front-tracking sense. The GTS library provided the basic tools for fulfilling such tasks, especially regarding the conservative remeshing algorithm. The C-Fortran computational interface developed for coupling both sets of computer codes provided the means for the Fortran portion of the code to access complex, C-native data structures of GTS.