Two Fluid Space-Time Discontinuous Galerkin Finite Element Method. Part I: Numerical Algorithm

W.E.H. Sollie, O. Bokhove, J.J.W. van der Vegt*

Department of Applied Mathematics, Institute of Mechanics, Processes and Control Twente, University of Twente, P.O.Box 217, 7500 AE, Enschede, The Netherlands

Abstract

A novel numerical method for two fluid flow computations is presented, which combines the space-time discontinuous Galerkin finite element discretization with the level set method and cut-cell based interface tracking. The space-time discontinuous Galerkin (STDG) finite element method offers high accuracy, an inherent ability to handle discontinuities and a very local stencil, making it relatively easy to combine with local *hp*-refinement. The front tracking is incorporated via cut-cell mesh refinement to ensure a sharp interface between the fluids. To compute the interface dynamics the level set method (LSM) is used because of its ability to deal with merging and breakup. Also, the LSM is easy to extend to higher dimensions. Small cells arising from the cut-cell refinement are merged to improve the stability and performance. The interface conditions are incorporated in the numerical flux at the interface and the STDG discretization ensures that the scheme is conservative as long as the numerical fluxes are conservative.

Keywords:
cut-cell, space-time discontinuous Galerkin, front tracking, level set, two fluid flow.

*Corresponding author.
Email addresses: w.e.h.sollie@math.utwente.nl (W.E.H. Sollie), o.bokhove@math.utwente.nl (O. Bokhove), j.j.w.vandervegt@math.utwente.nl (J.J.W. van der Vegt)

1. Introduction

Fluid flows with interfaces involve combinations of gasses, liquids and solids and have many applications in nature and industry. Examples include flows with bubbles, droplets or solid particles, wave-structure interactions, dam breaking, bed evolution, Rayleigh-Taylor and Kelvin-Helmholtz instabilities and industrial processes such as bubble columns, fluidized beds, granular flows and ink spraying. The flow patterns in these problems are complex and diverse and can be approached at various levels of complexity. Often the interface is not static but moves with the fluid flow velocity and in more complex cases interface topological changes due to breakup and coalescence processes may occur. Solutions often have a discontinuous character at the interface between different fluids, due to surface tension and other effects. In addition, the density and pressure differences across the interface can be very high, like in the case of liquid-gas flows. Also, the existence of shock or contact waves can introduce additional discontinuities into the problem. Because of the continuous advances in computer technology the numerical simulation of these problems is becoming increasingly affordable. However, there are several issues related to solving flows with interfaces numerically. These include issues regarding accuracy and conservation of the flow field quantities near the interface, robustness and stability of the interface coupling, complex geometries, unstructured mesh generation and motion, mesh topological changes and computational efficiency. A numerical method which has received much attention in recent years and which is especially suited for dealing with flows with strong discontinuities and unstructured meshes is the discontinuous Galerkin finite element method.

In this article a novel discontinuous Galerkin front tracking method for two fluid flows is presented, which is accurate, versatile and can alleviate some of the problems commonly encountered with existing methods. In order to explain and motivate the choices made for the numerical method, first the most important aspects of the space-time discontinuous Galerkin finite element method are discussed. This is followed by a discussion of important existing techniques for dealing with interfaces. Based on this discussion the interface related choices in the method are explained. Finally, the research objectives are stated.

For a complete survey of discontinuous Galerkin (DG) methods and their applications, see [6]. The main feature of DG methods is that they allow solutions to be discontinuous over element faces. The basis functions are defined

locally on each element with only a weak coupling to neighboring elements. The computational stencil is therefore very local; hence, DG methods are relatively easy to combine with parallel computation and also *hp*-refinement, where a combination of local mesh refinement (*h*-adaptation) and adjustment of polynomial order (*p*-adaptation) is used. Another important property is that DG discretizations are conservative. Near discontinuities higher order DG solutions will exhibit spurious oscillations. These oscillations may be removed by using slope limiting, shock fitting techniques or artificial dissipation in combination with discontinuity detection. Recently, Luo et al. [33] proposed the Hermite WENO limiter for DG methods, which uses Hermite reconstruction polynomials to maintain a small stencil even for higher order solutions. Krivodonova et al. [30] proposed a discontinuity detector for DG methods for hyperbolic conservation laws based on a result of strong superconvergence at the outflow boundary of each element. The disontinuity detector is used to prevent activation of the slope limiter in a smooth solution, which would otherwise reduce accuracy.

The space-time discontinuous Galerkin finite element method (STDG) introduced by van der Vegt and van der Ven ([61]) is a space-time variant of the DG method which is especially suited for handling dynamic mesh motions in space-time (See also [4, 28, 51, 62]). It features a five-stage semi-implicit Runge-Kutta scheme with coefficients optimized for stability in combination with multigrid for accelerated convergence to solve the (non)linear algebraic equations resulting from the DG discretization.

Many methods have been proposed for computing flows with interfaces or, to be more general, fronts [47]. By looking at the front representation in the mesh one can distinguish between front capturing and front tracking methods. Other methods exist, such as particle methods and boundary integral methods, but these are not relevant for the current discussion.

In front capturing methods a regular stationary mesh is used and there is no explicit front representation. Instead, the front is either described by means of marker particles, like in the marker and cell method, or by use of functions, such as in the volume of fluid and level set methods. The earliest numerical method for time dependent free surface flow problems was the marker and cell (MAC) method [9, 23]. Being a volume marker method it uses tracers or marker particles defined in a fixed mesh to locate the phases. However, the large number of markers required to obtain sufficient accuracy makes the method expensive.

In the Volume of Fluid (VoF) method [24, 40, 46, 67] a fractional volume

or color function is defined to indicate the fraction of a mesh element that covers a particular type of fluid. Algorithms for volume tracking are designed to solve the equation $\partial c/\partial t + \bar{\nabla} \cdot (c\,\mathbf{u}) = 0$, where c denotes the color function, $\mathbf{u}$ the local velocity at the front, t the time and $\bar{\nabla} = (\partial/\partial x_1, \cdots, \partial/\partial x_d)$ the spatial gradient operator in d-dimensional space. In the VoF method typically a reconstruction step is necessary to reproduce the interface geometry from the color function. More accurate VoF techniques like the Piecewise Linear Interface Construction (PLIC) method attempt to fit the interface by means of piecewise linear segments. VoF methods are easy to extend to higher dimensions and can be parallelized readily due to the local nature of the scheme. Also, they can automatically handle reconnection and breakup. Also, current VoF methods can conserve mass. However, VoF methods have difficulty in maintaining sharp boundaries between different fluids, and interfaces tend to smear. In addition, these methods can give inaccurate results when high interface curvatures occur. The computation of surface tension is not straightforward and in addition spurious bubbles and drops may be created. Recently, Greaves has combined the VoF method with Cartesian cut-cells with adapting hierarchical quadtree grids [21, 22], which alleviates some of these problems.

The Level Set Method (LSM) was introduced by Osher and Sethian in [36] and further developed in [1, 48, 52]. For a survey, see [49]. In the LSM an interface can be represented implicitly by means of the 0-level of a level set function $\psi(\mathbf{x}, t)$. The evolution of the interface is found by solving the level set equation $\partial\psi/\partial t + \mathbf{u} \cdot \bar{\nabla}\psi = 0$, with $\mathbf{u}$ the interface velocity. To reduce the computational costs a narrow band approach can be used, which limits the computations of the level set to a thin region around the interface. To enhance the level set accuracy it can be advected with the interface velocity, which for this purpose is extended from the interface into the domain. In case the level set becomes too distorted a reinitialization may be necessary. Various reinitialization algorithms are available based on solving a Hamilton-Jacobi partial differential equation [25, 37, 39]. Although the choice of the level set function is somewhat arbitrary, the signed distance to the interface tends to give the best accuracy in computing the curvature of the interface. Also, the LSM is easy to extend to higher dimensions and can automatically handle reconnection and breakup. The LSM, however, is not conservative in itself. Recent developments include the combination of the VoF method with the Level Set Method [53].

Front capturing methods have the advantage of a relatively simple for-

mulation. The main drawback of these methods lies in the need for complex interface shape restoration techniques, which often have problems in restoring the smooth and continuous interface shape, particularly in higher dimensions.

In front tracking and Lagrangian methods the front is tracked explicitly in the mesh. Front tracking was initially proposed in [43] and further developed in [16, 17, 18, 32, 35, 54, 59] and [60]. For a survey, see [26] and [44]. The evolution of the front is calculated by solving the equation $\partial \mathbf{x}/\partial t = \mathbf{u}$ at the front, where $\mathbf{x}$ is a point at the front and $\mathbf{u}$ its velocity. Glimm et al. [19] have combined front tracking with local grid based interface reconstruction using interface crossings with element edges. More recently they have proposed a fully conservative front tracking algorithm for systems of nonlinear conservation laws in [20].

Front tracking methods are often combined with either surface markers or cut-cells to define the location of the front. In the cut-cell method [3, 5, 7, 11, 27, 38, 41, 55, 56, 57, 58, 63, 64, 65] a Cartesian mesh is used for all elements except those which are intersected by the front. These elements are refined in such a way that the front coincides with the mesh. At a distance from the front the mesh remains Cartesian and computations are less expensive. A common problem with cut-cell methods is the creation of very small elements which leads to problems with the stiffness of the equations and causes numerical instability. One way to solve this problem is by element merging as proposed in [66].

In Lagrangian or moving mesh methods [8, 10, 13, 14, 15, 34, 45] the mesh is modified to follow the fluid. In these methods the mesh can become considerably distorted, which gives problems with the mesh topology and stretched elements. In the worst case, frequent remeshing may be necessary ([2, 31]). In cases of breakup and coalescence, where the interface topology changes, these methods tend to fail.

Front tracking methods are good candidates for solving problems that involve complex interface physics. They are robust and can reach high accuracy when the interface is represented using higher order polynomials, even on coarse meshes. A drawback of front tracking methods is that they require a significant effort to implement, especially in higher dimensions.

The numerical algorithm for two fluid flows presented here combines a space-time discontinuous Galerkin (STDG) discretization of the flow field with a cut-cell mesh refinement based interface tracking technique and a level set method (LSM) for computing the interface dynamics. The STDG discretization can handle interface discontinuities naturally, is conservative

and has a very compact computational stencil. The level set method has the benefit of a simple formulation which makes it easier to extend the method to higher dimensions and also provides the ability to handle topological changes automatically. The interface tracking serves to maintain a sharp interface between the two fluids. This allows for different equations to be used for each fluid, which are coupled at the interface by a numerical interface flux, based on the interface condition. In addition, front tracking methods typically have high accuracy. Cut-cell refinement is used since it has the benefit of being local in nature and also is relatively easy to extend to higher dimensions.

The outline of this article is as follows. In Section 2 the flow and level set equations are introduced. In Section 3 the background and refined meshes are discussed and the mesh refinement procedure is presented. In Section 4 the flow and level set discretizations, and the Runge-Kutta semi-implicit time integration method for the solution of the algebraic equations resulting from the numerical discretization are discussed. In Section 5 the two fluid algorithm is presented. Section 6 contains the final discussion and conclusions. In part II [50] the numerical algorithm will be applied to a number of model problems in two and three space-time dimensions.

2. Equations

2.1. Two fluid flow equations

Considered are flow problems involving two fluids as illustrated in Figure 1. The two fluids are separated in space-time by an interface S. Let $i = 1, 2$ denote the fluid index. Furthermore, let $\mathbf{x} = (t, \bar{\mathbf{x}}) = (x_0, \cdots, x_d)$ denote the space-time coordinates, with d the spatial dimension, $\bar{\mathbf{x}} = (x_1, \cdots, x_d)$ the spatial coordinates and $t \in [t_0, T]$ the time coordinate, with t_0 the initial time and T the final time. The space-time flow domain for fluid i is defined as $\mathcal{E}^i \subset \mathbb{R}^{d+1}$. The (space) flow domain for fluid i at time t is defined as $\Omega^i(t) = \{\bar{\mathbf{x}} \in \mathbb{R}^d | (t, \bar{\mathbf{x}}) \in \mathcal{E}^i\}$. The space-time domain boundary for fluid i, $\partial\mathcal{E}^i$ is composed of the initial and final flow domains $\Omega^i(t_0)$ and $\Omega^i(T)$, the interface S and the space boundaries $\mathcal{Q}^i = \{\mathbf{x} \in \partial\mathcal{E}^i | t_0 < t < T\}$. The two fluid space-time flow domain is defined as $\mathcal{E} = \cup_i \mathcal{E}^i$, the two fluid (space) flow domain at time t as $\Omega(t) = \cup_i \Omega^i(t)$ and the two fluid space-time domain boundary as $\partial\mathcal{E} = \cup_i \partial\mathcal{E}^i$. Let $\mathbf{w}^i$ denote a vector of N_w flow variables for fluid i. The bulk fluid dynamics for fluid i are assumed to be given as a

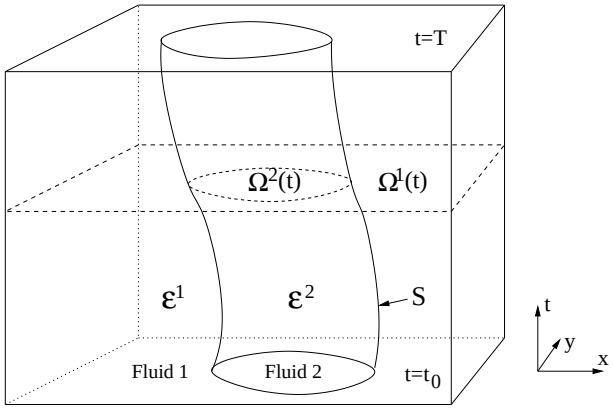

Figure 1: An example two fluid flow problem in space-time. Here $\mathcal{E}^i$ and $\Omega^i(t)$ denote the space-time and space flow domains for fluids $i = 1, 2$; and, S denotes the interface between the two fluids in space-time.

system of conservation laws:

$$\frac{\partial \mathbf{w}^i}{\partial t} + \bar{\nabla} \cdot F^i(\mathbf{w}^i) = 0, \tag{1}$$

where $\bar{\nabla} = (\partial/\partial x_1, \ldots, \partial/\partial x_d)$ denotes the spatial gradient operator and $F^i(\mathbf{w}^i) = (F_1^i, \cdots, F_d^i)$ the spatial flux tensor for fluid i with F_j^i the j-th flux vector and $j = 1, \cdots d$. Reformulated in space-time (1) becomes:

$$\begin{aligned} \nabla \cdot \mathcal{F}^i(\mathbf{w}^i) = 0, \text{ with} \\ \mathcal{F}^i(\mathbf{w}^i) = (\mathbf{w}^i, F^i(\mathbf{w}^i)), \end{aligned} \tag{2}$$

and $\nabla = (\partial/\partial t, \bar{\nabla})$ the space-time gradient operator and $\mathcal{F}^i(\mathbf{w}^i)$ the space-time flux tensor. The flow variables are subject to initial conditions:

$$\mathbf{w}^i(0, \bar{\mathbf{x}}) = \mathbf{w}_0^i(\bar{\mathbf{x}}), \tag{3}$$

boundary conditions:

$$\mathbf{w}^i(t, \bar{\mathbf{x}}) = \mathcal{B}_B^i(\mathbf{w}^i, \mathbf{w}_b^i) \text{ on } \mathcal{Q}^i/S \tag{4}$$

with $\mathbf{w}_b^i$ the prescribed boundary data at $\mathcal{Q}^i$, and interface conditions:

$$\mathbf{w}^i(t, \bar{\mathbf{x}}) = \mathcal{B}_S^i(\mathbf{w}^1, \mathbf{w}^2) \text{ on } S. \tag{5}$$

Since the actual flow variables, fluxes and initial, boundary and interface conditions are problem specific they shall be provided in part II [50] where the test cases are discussed.

2.2. Level set equation

To distinguish between the two fluids a level set function $\psi(\mathbf{x})$ is used:

$$\psi(t,\bar{\mathbf{x}}) = \begin{cases} < 0 & \text{in Fluid 1} \\ > 0 & \text{in Fluid 2} \\ = 0 & \text{at the interface.} \end{cases} \tag{6}$$

Initially, the level set function is defined as the minimum signed distance to the interface:

$$\psi(t,\bar{\mathbf{x}}) = \alpha \inf_{\forall \bar{\mathbf{x}}_S \in S(t)} \|\bar{\mathbf{x}} - \bar{\mathbf{x}}_S\|, \tag{7}$$

where $\alpha = -1$ in Fluid 1 and $\alpha = +1$ in Fluid 2, $\bar{\mathbf{x}}_S$ denotes a point on the interface $S(t)$ and $\|.\|$ is the Euclidian distance. The evolution of the level set is determined by an advection equation:

$$\frac{\partial \psi}{\partial t} + \bar{\mathbf{a}} \cdot \bar{\nabla}\psi = 0, \tag{8}$$

where $\bar{\mathbf{a}} = (a_1, \cdots, a_d)$ is a vector containing the level set velocity, which will be taken equal to the flow velocity. The level set function is subject to initial conditions:

$$\psi(0,\bar{\mathbf{x}}) = \psi_0(\bar{\mathbf{x}}), \text{ for } \bar{\mathbf{x}} \in \Omega(t_0). \tag{9}$$

At the domain boundary the level set is subject to solid wall boundary conditions:

$$\bar{\mathbf{a}}(t,\bar{\mathbf{x}}) \cdot \bar{\mathbf{n}} = 0, \text{ for } (t,\bar{\mathbf{x}}) \in \mathcal{Q}, \tag{10}$$

where $\bar{\mathbf{n}}$ denotes the space outward unit normal vector at the domain boundary.

3. Meshes

3.1. Two fluid mesh

To simplify computations, the two fluid domain is subdivided into a number of space-time slabs on which the equations are solved consecutively. Interval (t_0, T) is subdivided into N_t intervals $I_n = (t_n, t_{n+1})$, with $t_0 < t_1 <$

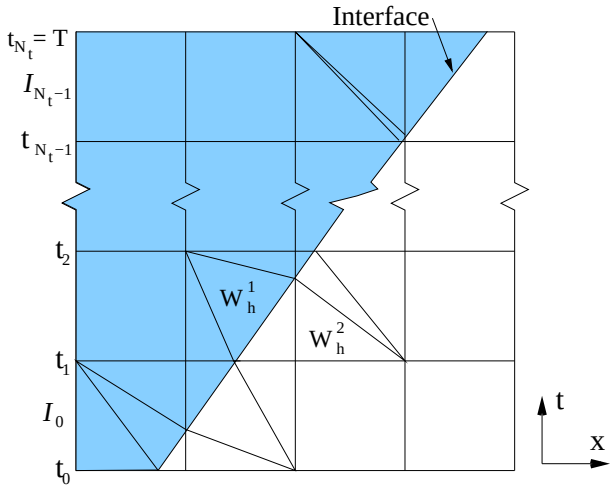

Figure 2: Two fluid mesh.

$\cdots < t_{N_t} = T$ and based on these intervals domains $\mathcal{E}^i$ are subdivided into space-time slabs $\mathcal{I}_n^i = \{\mathbf{x} \in \mathcal{E}^i | t \in I_n\}$. For every space-time slab $\mathcal{I}_n^i$ a tessellation $\mathcal{T}_h^{i,n}$ of non-overlapping space-time elements $\mathcal{K}_j^{i,n} \subset \mathbb{R}^{d+1}$ is defined:

$$\mathcal{T}_h^{i,n} = \left\{ \mathcal{K}_j^{i,n} \subset \mathbb{R}^{d+1} | \bigcup_{j=1}^{N_h^i} \bar{\mathcal{K}}_j^{i,n} = \bar{\mathcal{I}}_n^i \right. \\ \left. \text{and } \mathcal{K}_j^{i,n} \bigcap \mathcal{K}_{j'}^{i,n} = \emptyset \text{ if } j \neq j', 1 \leq j, j' \leq N_h^{i,n} \right\} \tag{11}$$

with $N_h^{i,n}$ the number of space-time elements in the space-time slab $\mathcal{I}_n^i$ for fluid i and where $\bar{\mathcal{K}}_j^{i,n} = \mathcal{K}_j^{i,n} \cup \partial\mathcal{K}_j^{i,n}$ denotes the closure of the space-time element. The tessellations $\mathcal{T}_h^{i,n}$ will be referred to as the two fluid or refined mesh (see Figure 2), since they will be constructed from a background mesh by performing local mesh refinement. The tessellations $\mathcal{T}_h^{i,n}$ define the numerical interface $S_h^{i,n}$ as a collection of finite element faces. The numerical interface is assumed to be geometrically identical in both tessellations, $S_h^{1,n} = S_h^{2,n}$. Let $\Gamma^{i,n} = \Gamma_I^{i,n} \cup \Gamma_B^{i,n} \cup \Gamma_S^n$ denote the set of all fluid i faces $\mathcal{S}_m^{i,n}$, with $\Gamma_I^{i,n}$ the set of internal faces, $\Gamma_B^{i,n}$ the set of boundary faces, and Γ_S^n the set of interfaces. Every internal face connects to exactly two elements in $\mathcal{T}_h^{i,n}$, denoted as the left element $\mathcal{K}^l$ and the right element $\mathcal{K}^r$. Every

boundary face connects to one element in $\mathcal{T}_h^{i,n}$, denoted as the element $\mathcal{K}^l$. Every interface connects to one element from $\mathcal{T}_h^{1,n}$ and also to one element from $\mathcal{T}_h^{2,n}$.

The finite element space $B_h^k(\mathcal{T}_h^{i,n})$ associated with the tessellation $\mathcal{T}_h^{i,n}$ is defined as:

$$B_h^k(\mathcal{T}_h^{i,n}) = \{\mathbf{w} \in L^2(\mathcal{E}_h^i) : \mathbf{w}|_{\mathcal{K}} \circ G_{\mathcal{K}} \in P^k(\hat{\mathcal{K}}), \forall \mathcal{K} \in \mathcal{T}_h^{i,n}\} \tag{12}$$

with $\mathcal{E}_h^i$ the discrete flow domain, $L^2(\mathcal{E}_h^i)$ the space of square integrable functions on $\mathcal{E}_h^i$, and $P^k(\hat{\mathcal{K}})$ the space of polynomials of degree at most k in the reference element $\hat{\mathcal{K}}$. The mapping $G_{\mathcal{K}_j^{i,n}}$ relates every element $\mathcal{K}_j^{i,n}$ to a reference element $\hat{\mathcal{K}} \subset \mathbb{R}^{d+1}$:

$$G_{\mathcal{K}_j^{i,n}} : \hat{\mathcal{K}} \rightarrow \mathcal{K}_j^{i,n} : \xi \mapsto \mathbf{x} = \sum_{k=1}^{N_{F,j}^{i,n}} x_k(\mathcal{K}_j^{i,n}) \chi_k(\xi) \tag{13}$$

with $N_{F,j}^{i,n}$ the number of vertices and $x_k(\mathcal{K}_j^{i,n})$ the coordinates of the vertices of space-time element $\mathcal{K}_j^{i,n}$. The finite element shape functions $\chi_k(\xi)$ are defined on the reference element $\hat{\mathcal{K}}$, with $\xi = (\xi_0, \cdots, \xi_d)$ the coordinates in the reference element. Given a set of basis functions $\hat{\phi}_m$ defined on the reference element, the basis functions $\phi_m : \mathcal{K}_j^{i,n} \rightarrow \mathbb{R}$ are defined on the space-time elements $\mathcal{K}_j^{i,n} \in \mathcal{T}_h^{i,n}$ by means of the mapping $G_{\mathcal{K}_j^{i,n}}$:

$$\phi_m = \hat{\phi}_m \circ G_{\mathcal{K}_j^{i,n}}^{-1}. \tag{14}$$

On the two fluid mesh the approximated flow variables are defined as:

$$\mathbf{w}_h^i(t, \bar{\mathbf{x}})|_{\mathcal{K}_j^{i,n}} = \sum_m \hat{\mathbf{W}}_m^i(\mathcal{K}_j^{i,n}) \phi_m(t, \bar{\mathbf{x}}) \tag{15}$$

with $\hat{\mathbf{W}}_m^i$ the expansion coefficients of fluid i. Each element in the two fluid mesh contains a single fluid. Therefore, in every element one set of flow variables is defined. Because the basis functions are defined locally in every element the space-time flow solution is discontinuous at the element faces.

3.2. Background mesh

In the construction of the two fluid mesh $\mathcal{T}_h^n$ it was assumed that every element contains exactly one fluid or equivalently that the interface is represented by a set of finite element faces. In order to define a mesh which

satisfies this requirement, a level set function ψ_h is defined on a space-time background mesh $\mathcal{T}_b^n$.

For every space-time slab $\mathcal{I}_n$ a tessellation $\mathcal{T}_b^n$ of space-time elements $\mathcal{K}_{b,\tilde{j}}^n \subset \mathbb{R}^{d+1}$ is defined:

$$\mathcal{T}_b^n = \left\{ \mathcal{K}_{b,\tilde{j}}^n \subset \mathbb{R}^{d+1} | \bigcup_{\tilde{j}=1}^{N_b} \bar{\mathcal{K}}_{b,\tilde{j}}^n = \bar{\mathcal{I}}_n \right. \\ \left. \text{and } \mathcal{K}_{b,\tilde{j}}^n \bigcap \mathcal{K}_{b,\tilde{j}'}^n = \emptyset \text{ if } \tilde{j} \neq \tilde{j}', 1 \leq \tilde{j}, \tilde{j}' \leq N_b \right\} \tag{16}$$

with N_b the number of space-time elements. The tessellation $\mathcal{T}_b^n$ will be referred to as the background mesh. In two and three space-time dimensions the background mesh is composed of square and cube shaped elements, respectively. The finite element space, mappings and basis functions are identical to those defined for the refined mesh in Section 3.1 except when dealing with the background mesh these will be denoted using a subscript b. On the background mesh a discontinuous Galerkin approximation of the level set is defined as:

$$\psi_h(t, \bar{\mathbf{x}})|_{\mathcal{K}_{b,\tilde{j}}^n} = \sum_m \hat{\mathbf{\Psi}}_m(\mathcal{K}_{b,\tilde{j}}^n) \phi_m(t, \bar{\mathbf{x}}), \tag{17}$$

with $\hat{\mathbf{\Psi}}_m$ the level set expansion coefficients. A discontinuous Galerkin discretization is used because the level set is advected with the flow velocity and will develop discontinuities in the vicinity of shock waves. In addition, a discontinuous Galerkin approximation of the level set velocity is defined as:

$$\bar{\mathbf{a}}_h(t, \bar{\mathbf{x}})|_{\mathcal{K}_{b,\tilde{j}}^n} = \sum_m \hat{\mathbf{A}}_m(\mathcal{K}_{b,\tilde{j}}^n) \phi_m(t, \bar{\mathbf{x}}), \tag{18}$$

3.3. Mesh refinement

After solving the level set equation the interface shape and position are approximately known from the 0-level set. In order to define a mesh for two fluid flow computations, the background mesh is refined by means of cut-cell mesh refinement. In the refined mesh the interface is represented by a set of faces on which the level set value is approximately zero.

The discontinuous nature of the level set approximation is not desirable for the mesh refinement, since it can result in hanging nodes. Hence the level

Algorithm 1 Mesh refinement algorithm.

FOR every element $\mathcal{K}^n_{b,\tilde{j}}$ *in* $\mathcal{T}^n_b$ *DO*
  Calculate intersection of 0*-level set* $\psi_c = 0$ *with* $\mathcal{K}^n_{b,\tilde{j}}$
  Select refinement rule
  Create and store interface physical nodes $\mathbf{x}_I$
  FOR all child elements $\hat{j}$ *defined by the refinement rule DO*
    Create $\mathcal{K}^{i,n}_{h,\hat{j}}$ *and store in* $\mathcal{T}^{i,n}_h$
  END DO
END DO
Generate faces for $\mathcal{T}^{i,n}_h$
FOR every element $\mathcal{K}^{i,n}_{h,j}$ *in* $\mathcal{T}^{i,n}_h$
  Initialize data on $\mathcal{K}^{i,n}_{h,j}$
END DO

set is smoothed before performing the mesh refinement. Assuming computations have reached time slab $\mathcal{I}_n$ the level set approximation ψ_h is smoothed by first looping over all elements in $\mathcal{I}_n$ and storing the multiplicity and the sum of the values of ψ_h in each vertex. For every vertex in $\mathcal{I}_n$ the continuous level set value ψ^c_h is calculated by dividing the sum of the ψ_h values by the vertex multiplicity. In every background element in $\mathcal{I}_n$, ψ_h is then reinitialized using the ψ^c_h values in the element vertices. To ensure continuity of the mesh only the values of the level set in the background elements belonging to the previous time slab $\mathcal{I}_{n-1}$ are used at the faces between the previous and the current time slab.

The mesh refinement algorithm is defined in Algorithm 1. The algorithm consists of a global element refinement step, in which all the elements of the background mesh are refined consecutively according to a set of refinement rules. The refinement rules define how a single element will be refined given an intersection with a 0-level set. The global refinement step is followed by a face generation step to create the connectivity between the refined elements. The face generation is straightforward and will not be discussed.

Given a smoothed level set, the element refinement is executed separately for each background element. For a given background element, it is first checked if the element contains more than one fluid by evaluating the level set at each vertex of the element. If the level set has the same sign in every

vertex, the element can contain only one fluid and it is copied directly to the refined mesh $\mathcal{T}_h^n$. Alternatively, the type of cut is determined from the level set signs. Depending on the cut type, the element is refined, based on a predefined element refinement rule for that type and the actual cut coordinates. The resulting elements are stored in $\mathcal{T}_h^n$. The element refinement rules have been designed such that for two neighboring elements the shared face is refined identically at both sides. Hence, no hanging nodes will occur in the refined mesh. The interface cut coordinates $\mathbf{x}_I$ for an edge cut by the interface are calculated as:

$$\mathbf{x}_I = \frac{\mathbf{x}_A \psi_h(\mathbf{x}_B) - \mathbf{x}_B \psi_h(\mathbf{x}_A)}{\psi_h(\mathbf{x}_A) - \psi_h(\mathbf{x}_B)}, \tag{19}$$

where $\mathbf{x}_A$ and $\mathbf{x}_B$ denote the coordinates of the edge vertices. For simplicity it is assumed that the level set is non-zero and can only be positive or negative in the vertices.

Because the refinement type is only based on the level set signs in the background element vertices, in cases where more than one interface intersects an element an ambiguity will occur where exactly the interface lies and the refinement rule will give rise to elements for which the fluid type is ambiguous. However, the fluid types of these elements can easily be found by computing the level set signs in the element midpoints.

The mesh refinement algorithm allows for freedom in choosing the element refinement rules. However, to avoid difficulties with face integration the refined mesh should have full connectivity. Element refinement rules have been developed for two and three dimensions, similar to [19], which will be discussed now.

3.4. 2D Refinement

Considered is a 2D background mesh containing only square elements. In order to define the 2D mesh refinement, first the symmetries of the square are introduced, followed by a discussion of all the relevant types of cuts in 2D and the introduction of a set of base types. Next, the square permutations are applied to the base types to find for each cut type the base type and the permutation which maps the base type to the cut type. Finally, the actual refinement rules are defined for each of the base types. In the mesh refinement algorithm, the refinement rule for a given cut type is obtained by permuting the element refinement rule of the base type to the given cut type.

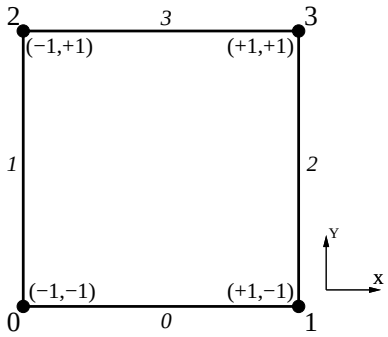

Figure 3: Vertex and edge numbering and nodal coordinates for the reference square.

Table 1: Square symmetries.

index	disjoint cycles	relation notation	Comment
0	(0)(1)(2)(3)	$\{0,1,2,3\}$	Identity
1	(0132)	$\{1,3,0,2\}$	90° right rotation
2	(03)(12)	$\{3,2,1,0\}$	180° right rotation
3	(0231)	$\{2,0,3,1\}$	270° right rotation
4	(02)(13)	$\{2,3,0,1\}$	Reflection x-axis
5	(01)(23)	$\{1,0,3,2\}$	Reflection y-axis
6	(03)(1)(2)	$\{3,1,2,0\}$	Reflection diagonal
7	(12)(0)(3)	$\{0,2,1,3\}$	Reflection diagonal

3.4.1. Square symmetries

The vertices and edges of the reference square are numbered using Local Node Indices (LNI) as shown in Figure 3. A square has a total of 8 symmetries usually referred to as the dihedral group D_4. Of these 4 are rotational symmetries and 4 are reflection symmetries. To describe the permutations there are two main notations, firstly as a decomposition in a product of disjoint cycles and secondly in relation notation. For example, in performing a counter clockwise rotation by 90 degrees, vertex 0 will move to vertex 1, vertex 1 to vertex 3, vertex 3 to vertex 2 and vertex 2 to vertex 0. In a decomposition in a product of disjoint cycles this is denoted as (0132). In relation notation it is denoted as $\{1,3,0,2\}$, where the index into the array gives the 'from' vertex and the value gives the 'to' vertex. The square symmetries are defined in Table 1. Here, permutation 0 describes the identity, permutations $1-3$ describe rotations and permutations $4-7$ describe reflections.

In the refinement algorithm permutations are needed not only of the vertices but also of nodes lying on edges. For this purpose the edge midpoints

Algorithm 2 Algorithm to determine edge permutation.

Given an edge with index i on the reference square
Get the vertex indices of the two edge vertices
Determine the permutation of the vertex indices
Find the index of the permuted edge from the permuted vertex indices

Table 2: Binary codes of the 2D base types. Each code represents a combination of level set signs for each of the 4 background element vertices, where a negative (positive) level set sign is represented by a 0 (1).

index	binary code	number
0	0111	7
1	0011	3
2	0110	6

are numbered from 4 to 7, ordered in the same way as the edges. Given the index of the edge, the index of the edge midpoint is found by adding 4, the number of vertices of the square. Hence, the permutation of an edge midpoint is found directly from the permutation of the edge. The permutation of an edge is found by looking at the permutations of its vertices. The algorithm is given in Algorithm 2. As an example, when applying permutation 1 to the edge 0, first the edge vertices are retrieved, in this case 0 and 1. Permuting these vertices gives permuted edge vertices 1 and 3; hence, the permuted edge is 2.

3.4.2. 2D base types

The classification of the 2D cuts is based on the values of the level set in the four vertices of the square. Each type is defined as a series of four signs corresponding to the level set signs in the four vertices. For example one type is defined by $--++$. Switching to a binary representation with $-$ and $+$ corresponding to 0 and 1, respectively, we can assign the number $0011 = 3$. Since a square has 4 vertices, there are $2^4 = 16$ possibilities. In two-dimensional space-time three refinement types have been defined as given in Table 2. In Figure 4 the signs of the level set in each vertex for every type are shown. In Figure 5 the corresponding cuts are shown, where for simplicity the interface cuts at the edges midpoints only. For Type 2

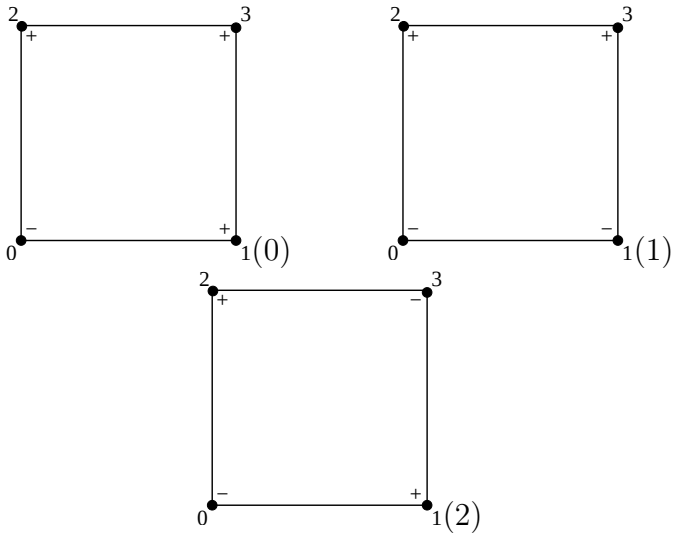

Figure 4: The vertex level set signs for the 2D base types.

two types of interface cuts are possible. The refinement rule will be able to handle both possibilities.

3.4.3. 2D base type permutations

The symmetries of the square are applied to the base cut types to find for each cut type the base type and the permutation from the base type to the cut type. For simplification, the sign of the level set in vertex 0 is assumed to be $-$ (0), meaning that the cut types need to be explicitly defined only for the indices $0-7$. To calculate the cut type for an index in the range $8-15$ the index value only has to be subtracted from the number 15. In Table 3 the base type is given for each cut type, based on the index of the cut type. The algorithm used to fill the table is given in Algorithm 3. Due to symmetries in the base types, different permutations can give equal results; hence, the permutation index is not necessarily uniquely defined.

3.4.4. 2D base type refinement

The element refinements for the 2D base types are shown in Figure 6 and the element refinements are given in Table 4. The algorithm to determine the element refinements given a level set configuration on a reference square is defined in Algorithm 4.

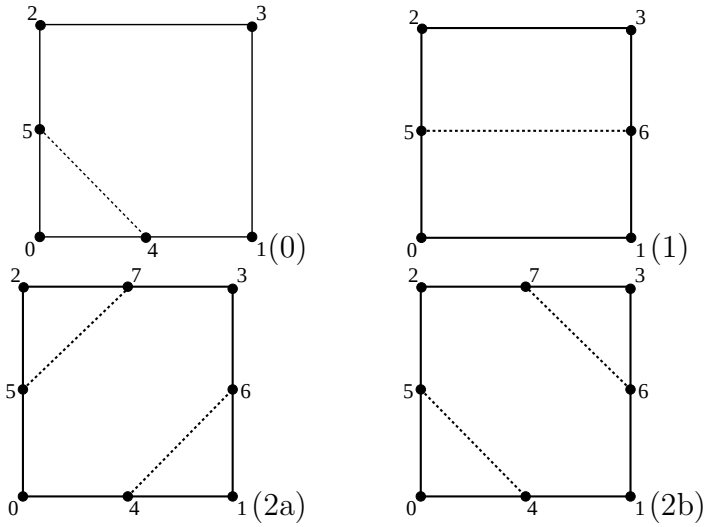

Figure 5: The interface cuts for the 2D base types. For type 2 two interface cuts are possible, which are both supported by the type 2 element refinement rule.

Table 3: 2D base types corresponding to cut type indices $0-15$.

index	base type	index	base type
0	No cut	8	Type 0
1	Type 0	9	Type 2
2	Type 0	10	Type 1
3	Type 1	11	Type 0
4	Type 0	12	Type 1
5	Type 1	13	Type 0
6	Type 2	14	Type 0
7	Type 0	15	No cut

Algorithm 3 Algorithm for filling the permutation lookup table.

Initialize permVec[8] of [type index, perm index] with [-1,-1]
FOR type index i from 0 to 3 DO
  FOR permutation index j from 0 to 7 DO
    Determine permutation j of cut type i
    Calculate index k of the permuted type
    Store [i,j] in permVec[k]
  END DO
END DO

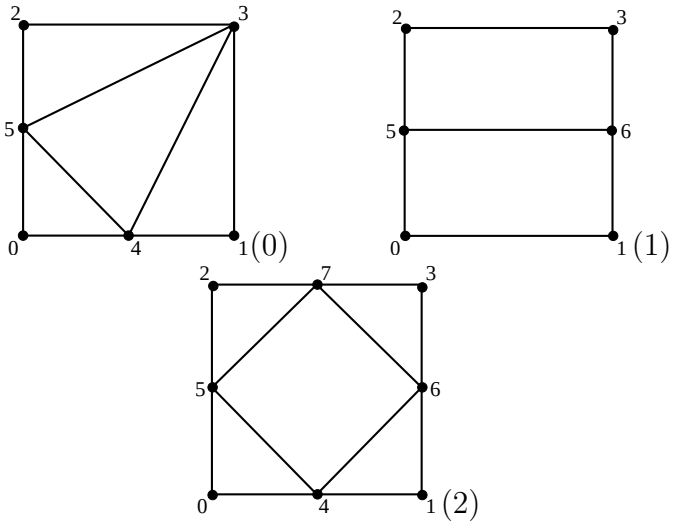

Figure 6: The element refinements for the 2D base types.

Table 4: 2D base type element refinements.

Type index	Child index	Child LNI	Fluid type
0	0	$\{0, 4, 5\}$	0
	1	$\{4, 1, 3\}$	1
	2	$\{5, 3, 2\}$	1
	3	$\{5, 4, 3\}$	1
1	0	$\{0, 1, 5, 6\}$	0
	1	$\{5, 6, 2, 3\}$	1
2	0	$\{0, 4, 5\}$	0
	1	$\{4, 1, 6\}$	1
	2	$\{6, 3, 7\}$	1
	3	$\{7, 2, 5\}$	0
	4	$\{5, 4, 7, 6\}$	0 or 1

Algorithm 4 Algorithm for determining element refinements.

Calculate index i for level set configuration
get base type from permVec[i]
IF base type does not equal -1 (unhandled type)
  FOR all child elements j DO
    FOR all local node indices k of child element j DO
      IF ($k < 4$) (square vertex)
        Get permutation index l from permVec[i]
        Determine permuted local node index k' of node k
      ELSE IF ($4 < k < 8$) (edge midpoint)
        Calculate edge index $e = k - 4$
        Find permuted edge index e'
        Calculate permuted local node index $k' = e' + 4$
      END IF
      Store k' as local node index of permuted child element j
    END DO
  END DO
END IF
Return permuted child elements local node indices

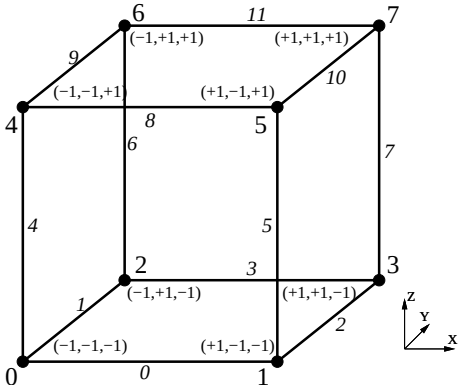

Figure 7: Vertex and edge numbering and nodal coordinates for the reference cube.

3.5. 3D Refinement

Considered is a 3D background mesh containing only cubical elements. In order to define the 3D refinement, first the symmetries of the cube are introduced, followed by a discussion of all the relevant types of cuts in 3D and the introduction of the 3D base types. Next, the cube permutations are applied to the base types to find for each cut type the base type and the permutation which maps the base type to the cut type. Finally, the actual refinement rules are defined for each of the base types.

3.5.1. Cube symmetries

The vertices and edges of the reference cube are numbered as shown in Figure 7. A cube has a total of 48 symmetries which are usually referred to as octahedral symmetries, since the symmetries of the cube are the same as those of its dual, the octahedron. Of these 24 are rotational symmetries which are orientation preserving. The remaining 24 are combinations of rotations and reflections. The cube symmetries are defined in Table 5. Here, permutation 0 describes the identity permutation, permutations $1-6$ describe a 90 degree rotation around the axis from the face center to the opposite face center, permutations $7-9$ describe 180 degree rotation around the axis from the face center to the opposite face center, permutations $10-15$ describe 180 degree rotation around the axis from the edge center to the opposite edge center and permutations $16-23$ describe 120 degree rotation around a body diagonal. Permutations $24-47$ are defined by taking permutations $0-23$ and applying inversion $(07)(16)(25)(34)$. Similarly to what was done in the

Table 5: Octahedral symmetries.

index	disjoint cycles	relation notation	Comment
0	(0)(1)(2)(3)(4)(5)(6)(7)	$\{0,1,2,3,4,5,6,7\}$	Identity
1	(0132)(4576)	$\{1,3,0,2,5,7,4,6\}$	90° rotation z-axis
2	(0231)(4675)	$\{2,0,3,1,6,4,7,5\}$	90° rotation z-axis
3	(0462)(1573)	$\{4,5,0,1,6,7,2,3\}$	90° rotation x-axis
4	(0264)(1375)	$\{2,3,6,7,0,1,4,5\}$	90° rotation x-axis
5	(0154)(2376)	$\{1,5,3,7,0,4,2,6\}$	90° rotation y-axis
6	(0451)(2673)	$\{4,0,6,2,5,1,7,3\}$	90° rotation y-axis
7	(03)(12)(47)(56)	$\{3,2,1,0,7,6,5,4\}$	180° rotation z-axis
8	(06)(24)(17)(35)	$\{6,7,4,5,2,3,0,1\}$	180° rotation x-axis
9	(05)(14)(27)(36)	$\{5,4,7,6,1,0,3,2\}$	180° rotation y-axis
10	(01)(25)(34)(67)	$\{1,0,5,4,3,2,7,6\}$	180° rotation $\{0,1\}$, $\{6,7\}$ midpoints
11	(02)(16)(34)(57)	$\{2,6,0,4,3,7,1,5\}$	180° rotation $\{0,2\}$, $\{5,7\}$ midpoints
12	(07)(13)(25)(46)	$\{7,3,5,1,6,2,4,0\}$	180° rotation $\{1,3\}$, $\{4,6\}$ midpoints
13	(07)(16)(23)(45)	$\{7,6,3,2,5,4,1,0\}$	180° rotation $\{2,3\}$, $\{4,5\}$ midpoints
14	(04)(16)(25)(37)	$\{4,6,5,7,0,2,1,3\}$	180° rotation $\{0,4\}$, $\{3,7\}$ midpoints
15	(07)(15)(26)(34)	$\{7,5,6,4,3,1,2,0\}$	180° rotation $\{1,5\}$, $\{2,6\}$ midpoints
16	(0)(7)(142)(356)	$\{0,4,1,5,2,6,3,7\}$	120° rotation body diagonal $\{0,7\}$
17	(0)(7)(124)(365)	$\{0,2,4,6,1,3,5,7\}$	120° rotation body diagonal $\{0,7\}$
18	(1)(6)(053)(247)	$\{5,1,4,0,7,3,6,2\}$	120° rotation body diagonal $\{1,6\}$
19	(1)(6)(035)(274)	$\{3,1,7,5,2,0,6,4\}$	120° rotation body diagonal $\{1,6\}$
20	(2)(5)(063)(147)	$\{6,4,2,0,7,5,3,1\}$	120° rotation body diagonal $\{2,5\}$
21	(2)(5)(036)(174)	$\{3,7,2,6,1,5,0,4\}$	120° rotation body diagonal $\{2,5\}$
22	(3)(4)(056)(172)	$\{5,7,1,3,4,6,0,2\}$	120° rotation body diagonal $\{3,4\}$
23	(3)(4)(065)(127)	$\{6,2,7,3,4,0,5,1\}$	120° rotation body diagonal $\{3,4\}$
24	(07)(16)(25)(34)	$\{7,6,5,4,3,2,1,0\}$	Inversion ((07)(16)(25)(34))
25	(0635)(1427)	$\{6,4,7,5,2,0,3,1\}$	90° rotation + Inversion
26	(0536)(1724)	$\{5,7,4,6,1,3,0,2\}$	
27	(0365)(1274)	$\{3,2,7,6,1,0,5,4\}$	
28	(0563)(1472)	$\{5,4,1,0,7,6,3,2\}$	
29	(0653)(1247)	$\{6,2,4,0,7,3,5,1\}$	
30	(0356)(1742)	$\{3,7,1,5,2,6,0,4\}$	
31	(04)(15)(26)(37)	$\{4,5,6,7,0,1,2,3\}$	180° rotation + Inversion
32	(01)(23)(45)(67)	$\{1,0,3,2,5,4,7,6\}$	
33	(02)(13)(46)(57)	$\{2,3,0,1,6,7,4,5\}$	
34	(06)(17)(2)(3)(4)(5)	$\{6,7,2,3,4,5,0,1\}$	180° rotation edge + Inversion
35	(05)(1)(27)(3)(4)(6)	$\{5,1,7,3,4,0,6,2\}$	
36	(0)(14)(2)(36)(5)(7)	$\{0,4,2,6,1,5,3,7\}$	
37	(0)(1)(24)(35)(6)(7)	$\{0,1,4,5,2,3,6,7\}$	
38	(03)(1)(2)(47)(5)(6)	$\{3,1,2,0,7,5,6,4\}$	
39	(0)(12)(3)(4)(56)(7)	$\{0,2,1,3,4,6,5,7\}$	
40	(07)(132645)	$\{7,3,6,2,5,1,4,0\}$	120° rotation body diagonal
41	(07)(154623)	$\{7,5,3,1,6,4,2,0\}$	+ Inversion
42	(023754)(16)	$\{2,6,3,7,0,4,1,5\}$	
43	(045732)(16)	$\{4,6,0,2,5,7,1,3\}$	
44	(013764)(25)	$\{1,3,5,7,0,2,4,6\}$	
45	(046731)(25)	$\{4,0,5,1,6,2,7,3\}$	
46	(026751)(34)	$\{2,0,6,4,3,1,7,5\}$	
47	(015762)(34)	$\{1,5,0,4,3,7,2,6\}$	

Table 6: Binary codes of the 3D base types. Each code represents a combination of level set signs for each of the 8 background element vertices, where a negative (positive) level set sign is represented by a 0 (1).

index	binary code	number	index	binary code	number
0	00100000	32	7	00100100	36
1	00100010	34	8	01100100	100
2	10100010	162	9	10100101	165
3	10101010	170	10	00101101	45
4	10110010	178	11	00101001	41
5	10100011	163	12	01101001	105
6	00101000	40			

2D refinement using Algorithm 2, the edge midpoints are numbered from 8 to 19 and the index of the edge midpoint is found by adding 8, the number of vertices of the cube, to the index of the edge.

3.5.2. 3D base types

Like in the 2D refinement, the 3D types are classified based on the values of the level set in the vertices. Thirteen configurations were identified, and these are given in Table 6. In Figure 8 the signs of the level set in each vertex for every base type are shown. In Figure 9 the corresponding cuts are shown, where for simplicity the interface cuts at the edges midpoints only. It should be noted that level set configurations $6-12$ allow for multiple interface cuts. This ambiguity is solved by making sure that for each level set configuration the element refinement rule is such that also multiple element cuts can be handled.

3.5.3. 3D base type permutations

The cube permutations are applied to the thirteen types of cuts to find the cut types permutations. In Figure 10 an example is shown of a permutation of the type 0 cut. To calculate the cut type for an index in the range $128-255$ the index value only has to be subtracted from the number 255. In the Table 7 the cut types for indices $0-127$ are given. The number of permuted cases for every type are given in Table 8. In the implementation a lookup table is used of size 256 which stores the type index $(0-12)$ of the cut and a permutation index $(0-47)$ from that base type. The algorithm used to fill the table is Algorithm 3, adapted to 3D.

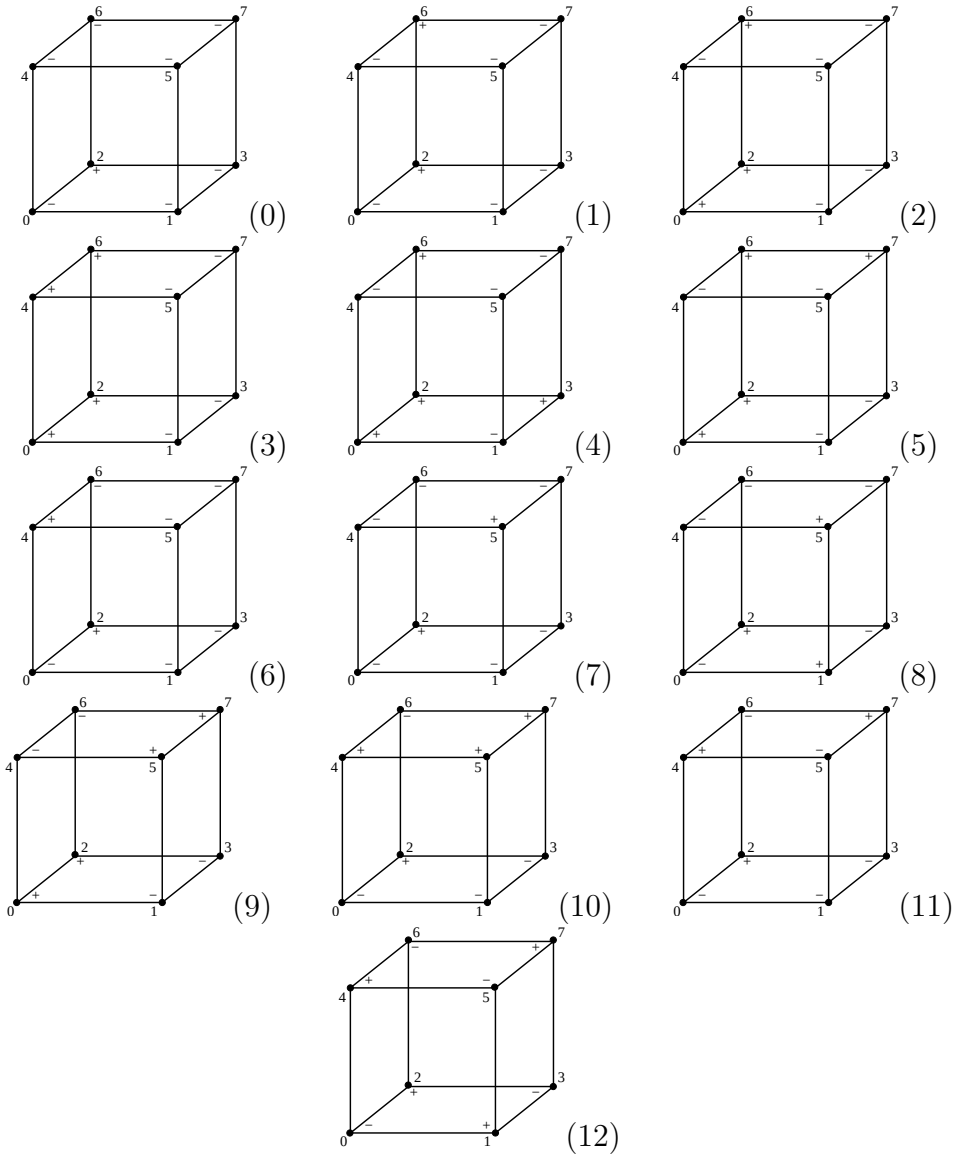

Figure 8: The vertex level set signs for the 3D base types.

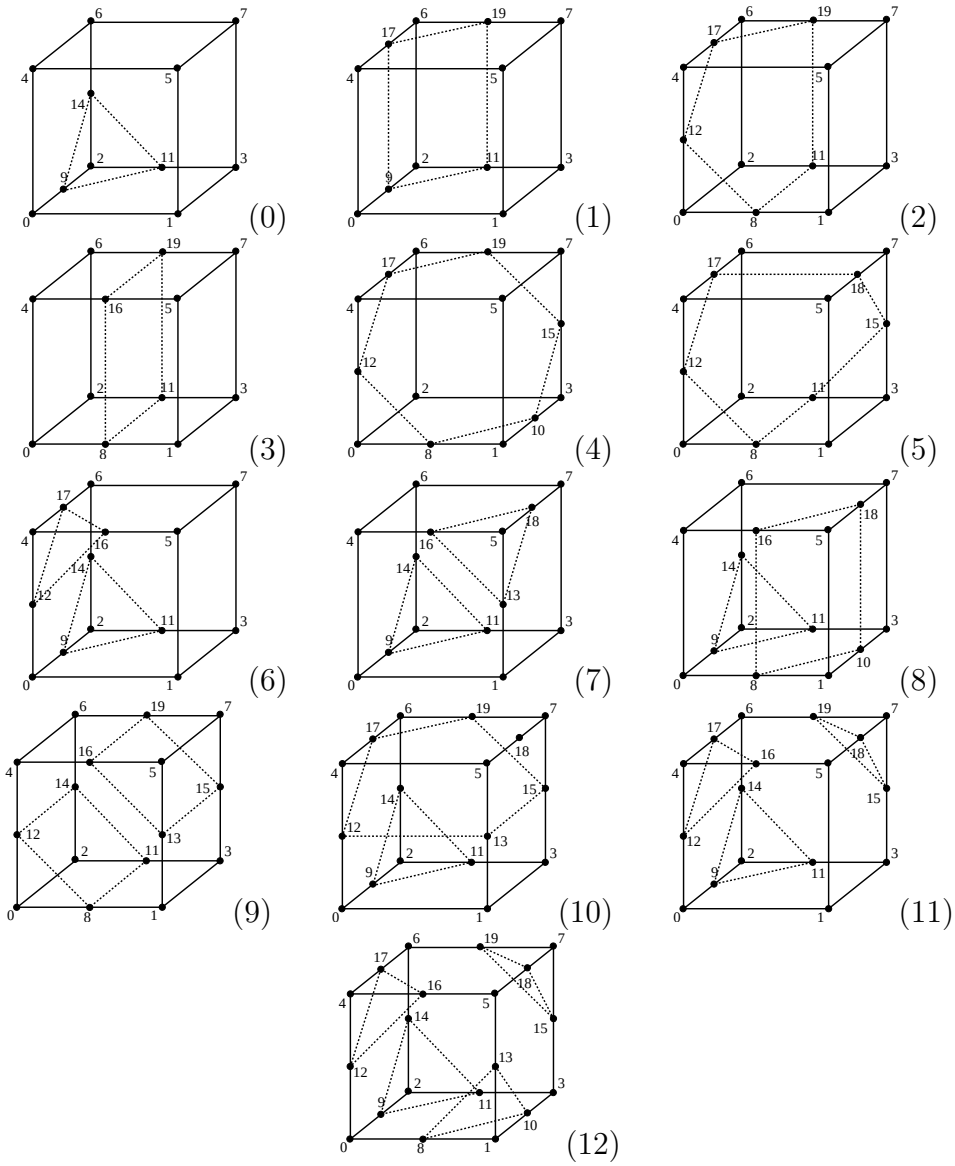

Figure 9: The interface cuts for the 3D base types. For types $6 - 12$ the level set configuration allows for alternative cuts not shown here, which are supported by the element refinement rule for that type.

Table 7: 3D base types corresponding to cut type indices $0 - 127$.

index	base type	index	base type	index	base type
0	No cut	43	Type 4	86	Type 10
1	Type 0	44	Type 8	87	Type 2
2	Type 0	45	Type 10	88	Type 8
3	Type 1	46	Type 5	89	Type 10
4	Type 0	47	Type 2	90	Type 9
5	Type 1	48	Type 1	91	Type 8
6	Type 6	49	Type 2	92	Type 5
7	Type 2	50	Type 2	93	Type 2
8	Type 0	51	Type 3	94	Type 8
9	Type 6	52	Type 8	95	Type 1
10	Type 1	53	Type 5	96	Type 6
11	Type 2	54	Type 10	97	Type 11
12	Type 1	55	Type 2	98	Type 8
13	Type 2	56	Type 8	99	Type 10
14	Type 2	57	Type 10	100	Type 8
15	Type 3	58	Type 5	101	Type 10
16	Type 0	59	Type 2	102	Type 9
17	Type 1	60	Type 9	103	Type 8
18	Type 6	61	Type 8	104	Type 11
19	Type 2	62	Type 8	105	Type 12
20	Type 6	63	Type 1	106	Type 10
21	Type 2	64	Type 0	107	Type 11
22	Type 11	65	Type 6	108	Type 10
23	Type 4	66	Type 7	109	Type 11
24	Type 7	67	Type 8	110	Type 8
25	Type 8	68	Type 1	111	Type 6
26	Type 8	69	Type 2	112	Type 2
27	Type 5	70	Type 8	113	Type 4
28	Type 8	71	Type 5	114	Type 5
29	Type 5	72	Type 6	115	Type 2
30	Type 10	73	Type 11	116	Type 5
31	Type 2	74	Type 8	117	Type 2
32	Type 0	75	Type 10	118	Type 8
33	Type 6	76	Type 2	119	Type 1
34	Type 1	77	Type 4	120	Type 10
35	Type 2	78	Type 5	121	Type 11
36	Type 7	79	Type 2	122	Type 8
37	Type 8	80	Type 1	123	Type 6
38	Type 8	81	Type 2	124	Type 8
39	Type 5	82	Type 8	125	Type 6
40	Type 6	83	Type 5	126	Type 7
41	Type 11	84	Type 2	127	Type 0
42	Type 2	85	Type 3		

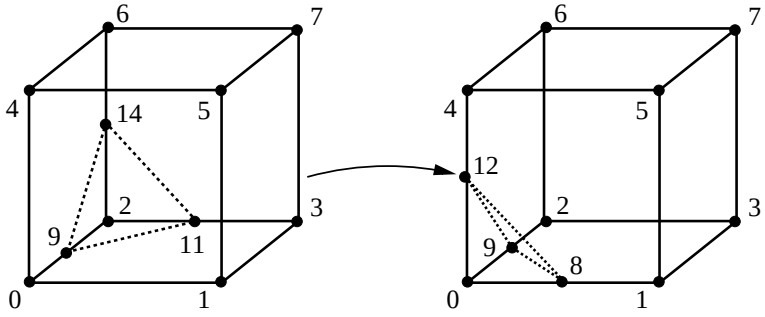

Figure 10: An example of a 3D permutation.

Table 8: Number of permutations for the 3D base types.

type	number of cases	type	number of cases
0	8	7	4
1	12	8	24
2	24	9	3
3	3	10	12
4	4	11	8
5	12	12	1
6	12		

3.5.4. 3D base type refinement

In order to define the element refinement of the 13 base types, first a surface refinement is defined, which is based on the 2D refinements illustrated in Figure 6. The surface refinements are shown in Figure 11. Element refinements have been manually devised based on the surface refinements. The element refinements for the 13 base types are given in Tables 9 and 10. In some of the refinements an additional node is used, which is located at the interface center and has LNI 20. To determine the element refinements given a level set configuration on a reference cube Algorithm 4 is used, adapted to 3D.

3.6. Merging

The occurrence of small elements in the refined mesh tends to cause numerical stability and performance problems. To solve these problems an element merging procedure was developed.

Let $\mathcal{K}_k^{i,n}, k = 0, \cdots, N_{\hat{j}}$ denote a collection of elements which need be merged, determined by means of a merging strategy to be discussed later. The merged element $\mathcal{K}_{m,\hat{j}}^{i,n}$ is defined as:

$$\mathcal{K}_{m,\hat{j}}^{i,n} = \bigcup_{k=0}^{N_{\hat{j}}} \mathcal{K}_k^{i,n}. \tag{20}$$

For each merged element $\mathcal{K}_{m,\hat{j}}^{i,n}$ the minimum and maximum bounding points $\mathbf{x}_{\hat{j}}^{min}$ and $\mathbf{x}_{\hat{j}}^{max}$ are defined componentwise as:

$$\begin{aligned} x_{\hat{j},l}^{min} &= \min_{\forall \mathbf{x} \in \mathcal{K}_{m,\hat{j}}^{i,n}} x_l, \ l = 0, \ldots, d \\ x_{\hat{j},l}^{max} &= \max_{\forall \mathbf{x} \in \mathcal{K}_{m,\hat{j}}^{i,n}} x_l, \ l = 0, \ldots, d, \end{aligned} \tag{21}$$

with d the space dimension. Let $\mathbf{x}_{\tilde{j}}^{min}$ and $\mathbf{x}_{\tilde{j}}^{max}$ denote the minimum and maximum bounding points of background element $\mathcal{K}_{b,\tilde{j}}^{n}$. It is assumed that all background mesh elements are of equal size and shape; hence, $\mathbf{x}_{\tilde{j}}^{max} - \mathbf{x}_{\tilde{j}}^{min} = \mathbf{h}_{b,\tilde{j}} = \mathbf{h}_b = constant$. For each merged element the minimum and maximum

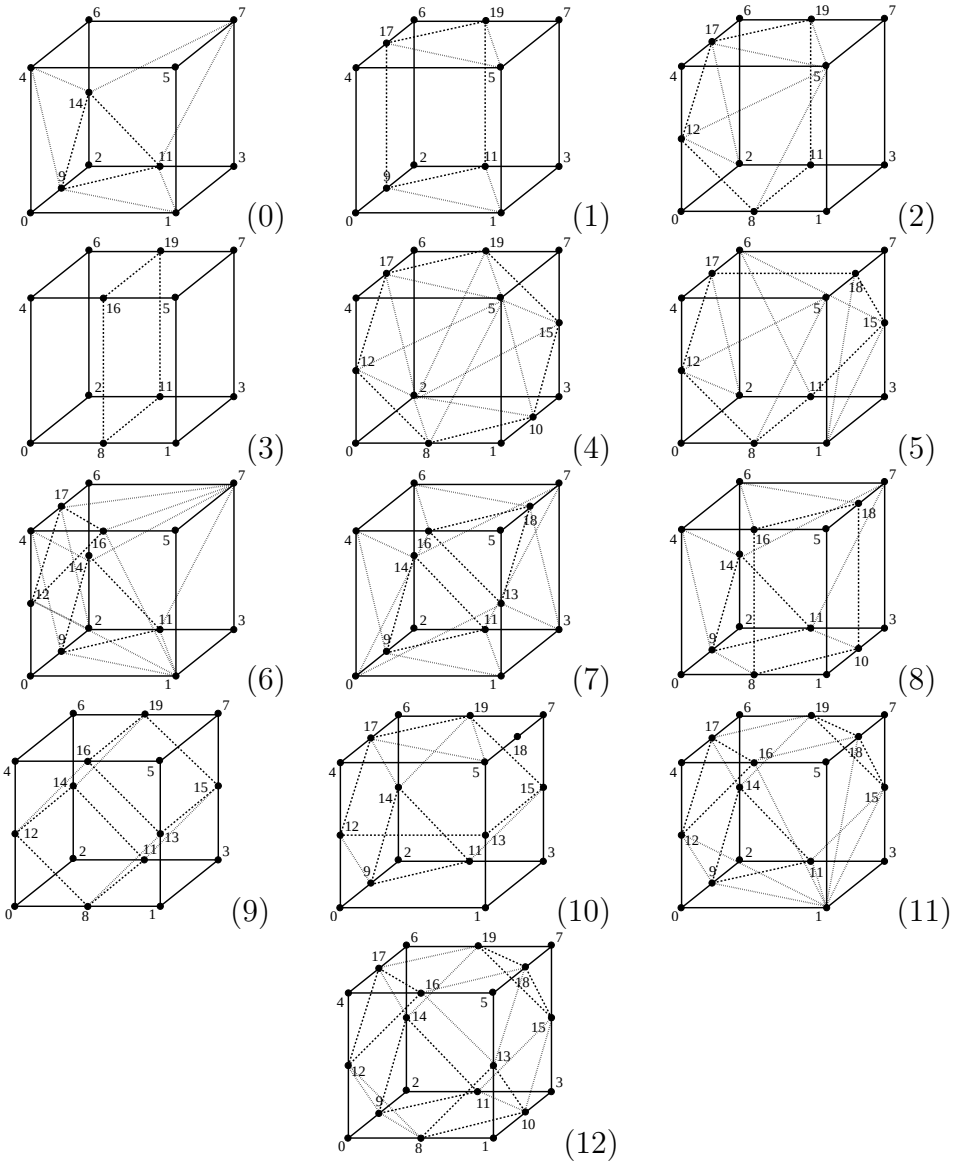

Figure 11: The surface refinements for the 3D base types.

Table 9: Element refinements for 3D base types.

Type index	Child index	Child LNI	Fluid type
0	0	{11, 1, 3, 5, 7}	0
	1	{9, 0, 1, 4, 5}	0
	2	{1, 5, 9, 11}	0
	3	{2, 9, 11, 14}	1
	4	{14, 4, 5, 6, 7}	0
	5	{4, 5, 9, 14}	0
	6	{5, 7, 11, 14}	0
	7	{5, 9, 11, 14}	0
1	0	{0, 1, 9, 4, 5, 17}	0
	1	{1, 3, 11, 5, 7, 19}	0
	2	{1, 11, 9, 5, 19, 17}	0
	3	{2, 9, 11, 6, 17, 19}	1
2	0	{20, 8, 1, 11, 3}	0
	1	{20, 0, 8, 2, 11}	1
	2	{4, 12, 17, 20}	0
	3	{12, 0, 2, 20}	1
	4	{6, 17, 2, 20}	1
	5	{12, 2, 17, 20}	1
	6	{12, 8, 0, 20}	1
	7	{8, 5, 1, 20}	0
	8	{12, 5, 8, 20}	0
	9	{4, 5, 12, 20}	0
	10	{20, 1, 5, 3, 7}	0
	11	{20, 2, 11, 6, 19}	1
	12	{20, 11, 3, 19, 7}	0
	13	{17, 5, 4, 20}	0
	14	{19, 7, 5, 20}	0
	15	{6, 19, 17, 20}	1
	16	{17, 19, 5, 20}	0
3	0	{0, 8, 2, 11, 4, 16, 6, 19}	1
	1	{8, 1, 11, 3, 16, 5, 19, 7}	0
4	0	{0, 8, 2, 12}	1
	1	{1, 8, 5, 10}	0
	2	{2, 10, 3, 15}	1
	3	{2, 6, 17, 19}	1
	4	{2, 19, 15, 8, 10}	1
	5	{2, 17, 19, 12, 8}	1
	6	{4, 5, 12, 17}	0
	7	{5, 7, 15, 19}	0
	8	{5, 8, 10, 19, 15}	0
	9	{5, 12, 8, 17, 19}	0
5	0	{20, 0, 8, 2, 11}	1
	1	{20, 8, 1, 11}	0
	2	{0, 2, 12, 20}	1
	3	{6, 17, 2, 20}	1
	4	{4, 12, 17, 20}	0
	5	{12, 2, 17, 20}	1
	6	{0, 12, 8, 20}	1
	7	{1, 8, 5, 20}	0
	8	{4, 5, 12, 20}	0
	9	{8, 12, 5, 20}	0
	10	{7, 15, 18, 20}	1
	11	{3, 1, 15, 20}	0
	12	{5, 18, 1, 20}	0
	13	{18, 15, 1, 20}	0
	14	{2, 11, 6, 20}	1
	15	{3, 15, 11, 20}	0
	16	{7, 6, 15, 20}	1
	17	{11, 15, 6, 20}	1
	18	{20, 7, 18, 6, 17}	1
	19	{20, 18, 5, 17, 4}	0
6	0	{1, 11, 9, 20}	0
	1	{1, 3, 11, 20}	0
	2	{20, 9, 14, 12, 17}	0 or 1
	3	{5, 1, 16, 20}	0
	4	{12, 16, 1, 20}	0
	5	{20, 5, 7, 1, 3}	0
	6	{3, 7, 11, 20}	0
	7	{14, 11, 7, 20}	0
	8	{5, 16, 7, 20}	0
	9	{16, 17, 7, 20}	0
	10	{9, 14, 11, 2}	1
	11	{9, 11, 14, 20}	0 or 1
	12	{12, 16, 17, 4}	1
	13	{12, 17, 16, 20}	0 or 1
	14	{7, 14, 17, 6}	0
	15	{7, 17, 14, 20}	0
	16	{1, 12, 9, 0}	0
	17	{1, 9, 12, 20}	0
7	0	{0, 1, 9, 20}	0
	1	{1, 11, 9, 20}	0
	2	{1, 3, 11, 20}	0
	3	{0, 9, 4, 20}	0
	4	{9, 14, 4, 20}	0
	5	{14, 6, 4, 20}	0
	6	{0, 1, 13, 20}	0
	7	{13, 16, 0, 20}	0
	8	{4, 0, 16, 20}	0
	9	{1, 13, 3, 20}	0
	10	{18, 3, 13, 20}	0
	11	{7, 3, 18, 20}	0
	12	{3, 11, 7, 20}	0
	13	{6, 7, 14, 20}	0
	14	{14, 7, 11, 20}	0
	15	{4, 6, 16, 20}	0
	16	{16, 6, 18, 20}	0
	17	{18, 6, 7, 20}	0
	18	{9, 11, 14, 20}	0
	19	{9, 14, 11, 2}	0 or 1
	20	{13, 16, 18, 20}	1
	21	{13, 18, 16, 5}	0 or 1

Table 10: Element refinements for 3D base types (continued).

Type index	Child index	Child LNI	Fluid type
8	0	{1, 10, 8, 5, 18, 16}	1
	1	{14, 16, 18, 8, 10}	0 or 1
	2	{16, 18, 14, 6}	0
	3	{14, 8, 10, 9, 11}	0 or 1
	4	{4, 16, 14, 6}	0
	5	{4, 14, 16, 9}	0
	6	{14, 8, 16, 9}	0 or 1
	7	{9, 4, 16, 0, 8}	0
	8	{18, 7, 14, 6}	0
	9	{18, 14, 7, 11}	0
	10	{14, 10, 11, 18}	0 or 1
	11	{11, 18, 7, 10, 3}	0
	12	{2, 9, 11, 14}	1
9	0	{2, 11, 14, 0, 8, 12}	1
	1	{3, 15, 11, 1, 13, 8}	0
	2	{7, 19, 15, 5, 16, 13}	1
	3	{6, 14, 19, 4, 12, 16}	0
	4	{11, 15, 14, 19, 8, 13, 12, 16}	0 or 1
10	0	{0, 1, 9, 20}	0
	1	{9, 1, 11, 20}	0
	2	{3, 11, 1, 20}	0
	3	{0, 9, 12, 20}	0
	4	{6, 17, 14, 20}	0
	5	{20, 12, 9, 17, 14}	0 or 1
	6	{20, 12, 13, 0}	0
	7	{20, 13, 15, 1, 3}	0
	8	{3, 15, 11, 20}	0
	9	{6, 14, 19, 20}	0
	10	{20, 11, 15, 14, 19}	0 or 1
	11	{6, 17, 19, 20}	0
	12	{9, 11, 14, 20}	0 or 1
	13	{9, 14, 11, 2}	1
	14	{12, 17, 13, 20}	0 or 1
	15	{17, 19, 13, 20}	0 or 1
	16	{19, 15, 13, 20}	0 or 1
	17	{19, 17, 13, 5}	1
	18	{17, 4, 5, 12, 13}	1
	19	{19, 5, 7, 13, 15}	1
11	0	{0, 1, 9, 20}	0
	1	{9, 1, 11, 20}	0
	2	{3, 11, 1, 20}	0
	3	{0, 9, 12, 20}	0
	4	{6, 17, 14, 20}	0
	5	{20, 12, 9, 17, 14}	0 or 1
	6	{0, 12, 1, 20}	0
	7	{12, 16, 1, 20}	0
	8	{16, 5, 1, 20}	0
	9	{5, 18, 1, 20}	0
	10	{18, 15, 1, 20}	0
	11	{15, 3, 1, 20}	0
	12	{3, 15, 11, 20}	0
	13	{6, 14, 19, 20}	0
	14	{20, 11, 15, 14, 19}	0 or 1
	15	{5, 16, 18, 20}	0
	16	{6, 19, 17, 20}	0
	17	{20, 17, 19, 16, 18}	0 or 1
	18	{9, 11, 14, 20}	0 or 1
	19	{12, 17, 16, 20}	0 or 1
	20	{18, 19, 15, 20}	0 or 1
	21	{9, 14, 11, 2}	1
	22	{12, 16, 17, 4}	1
	23	{18, 15, 19, 7}	1
12	0	{0, 8, 9, 20}	0
	1	{3, 11, 10, 20}	0
	2	{20, 8, 10, 9, 11}	0 or 1
	3	{0, 9, 12, 20}	0
	4	{6, 17, 14, 20}	0
	5	{20, 12, 9, 17, 14}	0 or 1
	6	{0, 12, 8, 20}	0
	7	{5, 13, 16, 20}	0
	8	{20, 16, 13, 12, 8}	0 or 1
	9	{3, 10, 15, 20}	0
	10	{5, 18, 13, 20}	0
	11	{20, 18, 15, 13, 10}	0 or 1
	12	{3, 15, 11, 20}	0
	13	{6, 14, 19, 20}	0
	14	{20, 11, 15, 14, 19}	0 or 1
	15	{6, 19, 17, 20}	0
	16	{5, 16, 18, 20}	0
	17	{20, 17, 19, 16, 18}	0 or 1
	18	{9, 11, 14, 20}	0 or 1
	19	{12, 17, 16, 20}	0 or 1
	20	{8, 13, 10, 20}	0 or 1
	21	{18, 19, 15, 20}	0 or 1
	22	{9, 14, 11, 2}	1
	23	{12, 16, 17, 4}	1
	24	{8, 10, 13, 1}	1
	25	{18, 15, 19, 7}	1

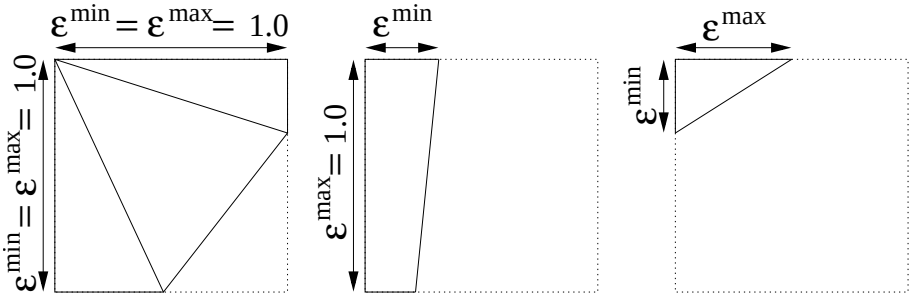

Figure 12: Illustration of the first step in the merging strategy. The dotted lines represent the background element and the solid lines represent the collection of child elements of one of the fluid types. The collection on the left has an $\epsilon^{min} > \epsilon^{MIN}$ and hence is considered a valid merged element in itself. The collections in the middle and on the right are have a small ϵ^{min} and require merging with a neigboring element.

lengths relative to the background element are defined as:

$$\epsilon_{\hat{j}}^{min} = \min_{l=0,\cdots,d} \frac{x_{\hat{j},l}^{max} - x_{\hat{j},l}^{min}}{h_{b,l}}$$
$$\epsilon_{\hat{j}}^{max} = \min_{l=0,\cdots,d} \frac{x_{\hat{j},l}^{max} - x_{\hat{j},l}^{min}}{h_{b,l}}. \tag{22}$$

In addition two predefined parameters, $\epsilon^{MIN} = 0.9$ and $\epsilon^{MAX} = 1.9$, are introduced. The merging strategy is defined for each fluid i individually as follows:

- Step 1: For each background element $\mathcal{K}_{b,\tilde{j}}^n$ retrieve the collection of all child elements that contain fluid i. For this collection of elements compute ϵ^{min} and ϵ^{max} and store these values on the background element. If the background element does not contain fluid i elements it is unavailable for merging and $\epsilon^{min} = \epsilon^{max} = 0.0$. If $\epsilon^{min} < \epsilon^{MIN}$ the collection defines a small or thin merged element and requires merging involving one or more neighboring background elements. If $\epsilon^{min} > \epsilon^{MIN}$ the collection itself defines a valid merged element. Step 1 is illustrated in Figure 12.

- Step 2: Using a loop over the faces in the background mesh, it is determined for each background element $\mathcal{K}_{b,\tilde{j}}^n$ which neighboring elements $\mathcal{K}_{b,k}^n, k = 0, \ldots, N_{\tilde{j}}$ are usable for merging, which is the case if

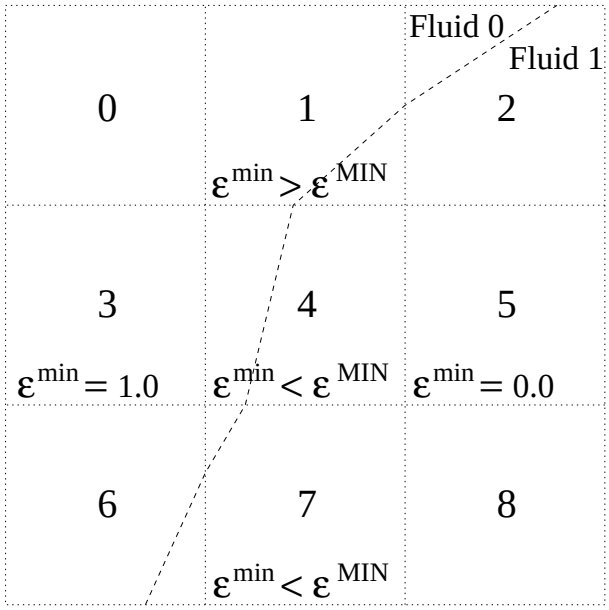

Figure 13: Illustration of the second step in the merging strategy for fluid type 0 and background element 4. The background elements are shown in dotted lines and the 0-level set is shown as a dashed line. Background element 4 has an $\epsilon^{min} < \epsilon^{MIN}$ and hence requires merging with one or more of the neighboring elements $1, 3, 5$ and 7. Elements 1 and 3 both contain enough fluid 0 ($\epsilon^{min} > \epsilon^{MIN}$) and hence are valid candidates for merging, while element 5 and element 7 do not contain enough fluid 0 ($\epsilon^{min} < \epsilon^{MIN}$) and hence are invalid candidates for merging.

the neighboring element contains a collection of fluid i elements with $\epsilon^{min} > \epsilon^{MIN}$. Step 2 is illustrated in Figure 13.

- Step 3: The merged elements are determined in three steps. Each step corresponds to a different type of merging, and these are illustrated in Figure 14. After a background element has been used in merging it is marked as UNAVAILABLE.

  – Type 1: For each available individual background element $\mathcal{K}^n_{b,\tilde{j}}$ check if $\epsilon^{min} > \epsilon^{MIN}$ and if it has at least two available neighboring elements for which $\epsilon^{min} < \epsilon^{MIN}$. If so, merge all refined elements $\mathcal{K}^{i,n}_j$ with the correct fluid type i contained in these background elements.

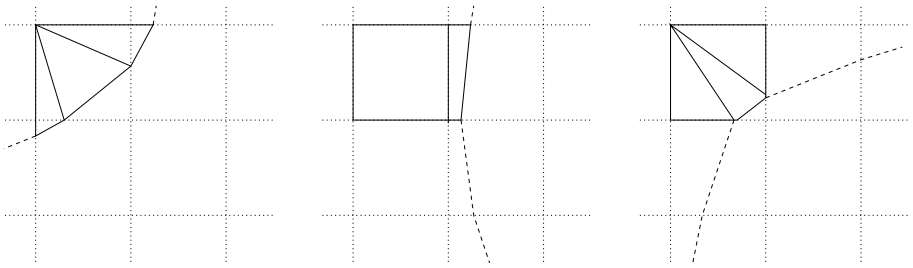

Figure 14: The three types of merged elements. The solid lines represent the refined elements that will be combined into a single merged element. The dotted lines represent the background mesh and the dashed lines represent the interface at positions not occupied by the merged element.

- Type 2: For each available individual background element $\mathcal{K}^n_{b,\tilde{j}}$ check if $\epsilon^{min} < \epsilon^{MIN}$. If so loop over all available neighboring elements $\mathcal{K}^n_{b,k}, k = 0, \cdots, N_{\tilde{j}}$ with $N_{\tilde{j}}$ the number of available neighboring elements. For each combination of the background element $\mathcal{K}^n_{b,\tilde{j}}$ and a neighboring element $\mathcal{K}^n_{b,k}$ determine ϵ^{min}_k. Find the $\tilde{k}$ for the combination which has the largest size, $\epsilon^{min}_{\tilde{k}} > \epsilon^{min}_k, k = 0, \cdots, N_{\tilde{j}}$. Merge all refined elements $\mathcal{K}^{i,n}_j$ with the correct fluid type i contained in the background elements $\mathcal{K}^n_{b,\tilde{j}}$ and $\mathcal{K}^n_{b,\tilde{k}}$.
- Type 3: For each available individual background element $\mathcal{K}^n_{b,\tilde{j}}$ check if $\epsilon^{min} > \epsilon^{MIN}$. If so check if it contains more than one element $\mathcal{K}^{i,n}_j$ with the correct fluid type i and if so merge these elements.

The merged elements tend to have complex shapes which makes it difficult to find suitable reference elements and basis functions. To alleviate this problem a bounding box element is introduced ([12]), which is simple shaped and contains the merged element. This merging procedure is illustrated for two dimensions in Figure 15 and an example of a mesh with merged elements in two dimensions is shown in Figure 16.

Let $\mathcal{K}^{i,n}_{M,\hat{j}}$ denote the bounding box of the merging element $\mathcal{K}^{i,n}_{m,\hat{j}}$. The finite element space, mappings and basis functions used for the bounding box elements are identical to those defined for the refined mesh but will be de-

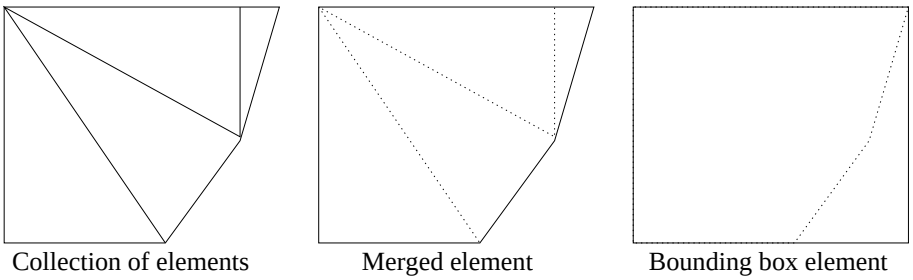

Figure 15: A collection of elements, their merged element and its bounding box element, in physical space.

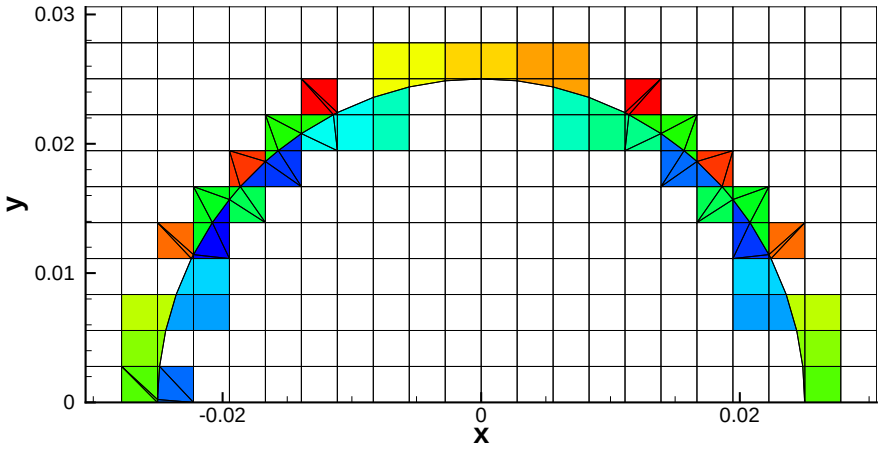

Figure 16: Refined mesh showing the merged elements as colored collections of child elements.

noted using a subscript M. On the bounding box element the approximated flow variables are defined as:

$$\mathbf{w}_h^i(t,\bar{\mathbf{x}})|_{\mathcal{K}_{M,\hat{j}}^{i,n}} = \sum_m \hat{\mathbf{W}}_m^i(\mathcal{K}_{M,\hat{j}}^{i,n})\phi_m(t,\bar{\mathbf{x}}) \tag{23}$$

with $\hat{\mathbf{W}}_{M,\hat{j}}^i$ the flow coefficients of fluid i. Each merged element contains exactly one fluid. For all elements $\mathcal{K}_k^{i,n} \subset \mathcal{K}_{m,\hat{j}}^{i,n}$ the flow evaluation is redefined as an evaluation in the bounding box element:

$$\mathbf{w}_h^i(\mathbf{x})|_{\mathcal{K}_k^{i,n}} = \mathbf{w}_h^i(\mathbf{x})|_{\mathcal{K}_{M,\hat{j}}^{i,n}}. \tag{24}$$

Integration of a function $f(\mathbf{w_h^i})$ over a merged element $\mathcal{K}_{m,\hat{j}}^{i,n}$ is performed by integrating over all the individual elements and summing the contributions:

$$\int_{\mathcal{K}_{m,\hat{j}}^{i,n}} f(\mathbf{w}_h^i)d\mathcal{K} = \sum_{k=0}^{N_{\hat{j}}} \int_{\mathcal{K}_k^{i,n}} f(\mathbf{w}_h^i)d\mathcal{K}. \tag{25}$$

4. Space-time discontinuous Galerkin discretization

4.1. Flow discretization

The discontinuous Galerkin finite element approximation for two fluid flows on the refined mesh $\mathcal{T}_h^{i,n}$ is found by multiplying (2) with an arbitrary test function $\mathbf{v} \in B_h^k(\mathcal{T}_h^{i,n})$ and integrating over all elements in the domains $\mathcal{E}^1$ and $\mathcal{E}^2$:

$$\sum_{\mathcal{K}_j^{i,n} \in \mathcal{T}_h^{i,n}} \int_{\mathcal{K}_j^{i,n}} \mathbf{v}\nabla \cdot \mathcal{F}^i(\mathbf{w}^i)\, d\mathcal{K} = 0. \tag{26}$$

Applying Gauss' theorem results in:

$$
\begin{aligned}
&- \sum_{\mathcal{K}_j^{i,n} \in \mathcal{T}_h^{i,n}} \int_{\mathcal{K}_j^{i,n}} \nabla \mathbf{v} \cdot \mathcal{F}^i(\mathbf{w}^i)\, d\mathcal{K} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_I^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{F}^{i,l}(\mathbf{w}^{i,l}) \cdot \mathbf{n}_{\mathcal{K}}^l\, \mathbf{v}^l + \mathcal{F}^{i,r}(\mathbf{w}^{i,r}) \cdot \mathbf{n}_{\mathcal{K}}^r\, \mathbf{v}^r\, d\mathcal{S} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_B^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{F}^{i,l}(\mathbf{w}^{i,l}) \cdot \mathbf{n}_{\mathcal{K}}^l\, \mathbf{v}^l\, d\mathcal{S} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_S^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{F}^{i,l}(\mathbf{w}^{i,l}) \cdot \mathbf{n}_{\mathcal{K}}^l\, \mathbf{v}^l\, d\mathcal{S} = 0,
\end{aligned} \tag{27}
$$

where $\mathcal{F}^{i,K}$ and $\mathbf{w}^{i,K}$ are the limiting trace values at the face $\mathcal{S}$ of element $\mathcal{K}^{i,K}$, $K = l, r$.

Let the trace v_h^K of a function v_h on a face $\mathcal{S}$ with respect to the element $\mathcal{K}^K, K = l, r$ be defined as $v_h^K = \lim_{\epsilon \downarrow 0} v_h(\mathbf{x} - \epsilon \mathbf{n}_{\mathcal{K}}^K)$, where $\mathbf{n}_{\mathcal{K}}^K = (n_0, \ldots, n_d)$ is the space-time outward unit normal vector at the face $\mathcal{S}$ with respect to element $\mathcal{K}^K$. Left and right normal vectors of a face are related as $\mathbf{n}_{\mathcal{K}}^l = -\mathbf{n}_{\mathcal{K}}^r$. The element local trace $v_h^{\pm}$ of a function v_h on a face $\mathcal{S}$ is defined as $v_h^{\pm} = \lim_{\epsilon \downarrow 0} v_h(\mathbf{x} \pm \epsilon \mathbf{n}_{\mathcal{K}})$. The average $\{\!\{F\}\!\}$ of a scalar or vector function F on the face $\mathcal{S}_m \in \Gamma_I$ is defined as $\{\!\{F\}\!\} := \frac{1}{2}(F^l + F^r)$, where l and r denote the traces at elements $\mathcal{K}^l$ and $\mathcal{K}^r$, respectively. The jump $[[F]]$ of a scalar function F on the face $\mathcal{S}_m \in \Gamma_I$ is defined as $[[F]] := F^l \mathbf{n}^l + F^r \mathbf{n}^r$ and the jump $[[\mathbf{G}]]$ of a vector function $\mathbf{G}$ on the face $\mathcal{S}_m \in \Gamma_I$ is defined as $[[\mathbf{G}]] := \mathbf{G}^l \cdot \mathbf{n}^l + \mathbf{G}^r \cdot \mathbf{n}^r$. The jump operator satisfies on Γ_I the product rule $[[F\mathbf{G}]] = \{\!\{F\}\!\}[[\mathbf{G}]] + [[F]]\{\!\{\mathbf{G}\}\!\}$.

By using a conservative flux, $\mathcal{F}^l(\mathbf{w}^l) \cdot \mathbf{n}_{\mathcal{K}}^l = -\mathcal{F}^r(\mathbf{w}^r) \cdot \mathbf{n}_{\mathcal{K}}^r$; hence, $[[\mathcal{F}(\mathbf{w})]] = 0$, the integration over the internal faces is rewritten as:

$$
\begin{aligned}
&\sum_{\mathcal{S}_m^{i,n} \in \Gamma_I^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{F}^{i,l}(\mathbf{w}^{i,l}) \cdot \mathbf{n}_{\mathcal{K}}^l\, \mathbf{v}^l + \mathcal{F}^{i,r}(\mathbf{w}^{i,r}) \cdot \mathbf{n}_{\mathcal{K}}^r\, \mathbf{v}^r\, d\mathcal{S} \\
&= \sum_{\mathcal{S}_m^{i,n} \in \Gamma_I^{i,n}} \int_{\mathcal{S}_m^{i,n}} \{\!\{\mathcal{F}^i(\mathbf{w}^i)\}\!\} \cdot [[\mathbf{v}]]\, d\mathcal{S}.
\end{aligned} \tag{28}
$$

So far the formulation has been strictly local, in the sense that neighboring elements and also the initial, boundary and interface conditions are not incorporated. In order to do this, numerical fluxes are introduced. At internal

faces the flux is replaced by a numerical flux $\mathcal{H}_I^i(\mathbf{w}^l, \mathbf{w}^r, \mathbf{n}_\mathcal{K})$, which is consistent: $\mathcal{H}(\mathbf{w}, \mathbf{w}, \mathbf{n}_\mathcal{K}) = \mathcal{F}(\mathbf{w}) \cdot \mathbf{n}_\mathcal{K}^l$, and conservative. Likewise at the boundary faces the flux is replaced by a numerical flux $\mathcal{H}_B^i(\mathbf{w}^l, \mathbf{w}^r, \mathbf{n}_\mathcal{K})$, which is also consistent. At the interface the flux is replaced by a numerical interface flux $\mathcal{H}_S^i(\mathbf{w}^{i,l}, \mathbf{w}_s^i, \mathbf{n}_\mathcal{K})$, with $\mathbf{w}_s^i$ the ghost state at the interface for fluid i. Using the fact that for a conservative flux $\{\!\{\mathcal{H}(\mathbf{w}^l, \mathbf{w}^r, \mathbf{n}_\mathcal{K})\}\!\} = \mathcal{H}(\mathbf{w}^l, \mathbf{w}^r, \mathbf{n}_\mathcal{K})$ and replacing the trial and test functions by their approximations in the finite element space $B_h^k(\mathcal{T}_h^{i,n})$, the weak formulation is defined as:

Find $\mathbf{w}_h^i \in B_h^k(\mathcal{T}_h^{i,n})$ such that for all $\mathbf{v}_h \in B_h^k(\mathcal{T}_h^{i,n})$:

$$
\begin{aligned}
&- \sum_{\mathcal{K}_j^{i,n} \in \mathcal{T}_h^{i,n}} \int_{\mathcal{K}_j^{i,n}} \nabla \mathbf{v}_h \cdot \mathcal{F}^i(\mathbf{w}_h^i)\, d\mathcal{K} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_I^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_I^i(\mathbf{w}_h^{i,l}, \mathbf{w}_h^{i,r}, \mathbf{n}_\mathcal{K})\, (\mathbf{v}_h^l - \mathbf{v}_h^r)\, d\mathcal{S} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_B^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_B^i(\mathbf{w}_h^{i,l}, \mathbf{w}_b^i, \mathbf{n}_\mathcal{K})\, \mathbf{v}_h^l\, d\mathcal{S} \\
&+ \sum_{\mathcal{S}_m^{i,n} \in \Gamma_S^{i,n}} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_S^i(\mathbf{w}_h^i, \mathbf{w}_s^i, \mathbf{n}_\mathcal{K})\, \mathbf{v}_h^l\, d\mathcal{S} = 0, \\
&\qquad\qquad i = 1, 2,\ n = 0, \cdots, N_t - 1.
\end{aligned} \tag{29}
$$

Introduction of the polynomial expansion (15) in (29) and using the basis functions ϕ_l for the test functions gives the following discretization in each space-time element $\mathcal{K}_j^{i,n}$:

$$
\begin{aligned}
&\mathcal{L}_{kl}^{i,n}(\hat{\mathbf{W}}^n, \hat{\mathbf{W}}^{n-1}) = 0,\ i = 1, 2,\ n = 0, \cdots, N_t - 1, \\
&\qquad\qquad k = 0, \cdots, N_w - 1,\ l = 0, \ldots, N_{B,j}^{i,n} - 1
\end{aligned} \tag{30}
$$

with N_t the number of time slabs, N_w the number of flow variables, $N_{B,j}^{i,n}$ the

number of basis functions. The nonlinear operator $\mathcal{L}_{kl}^{i,n}$ is defined as:

$$\begin{aligned}\mathcal{L}_{kl}^{i,n} = &- \int_{\mathcal{K}_j^{i,n}} (\nabla \phi_l)_j \cdot \mathcal{F}_{kj}^i(\mathbf{w}_h^i) d\mathcal{K} \\ &+ \sum_{\mathcal{S}_m^{i,n} \in \partial\mathcal{K}_j^{i,n} \cap \Gamma_I^i} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_{I,k}(\mathbf{w}_h^{i,-}, \mathbf{w}_h^{i,+}, \mathbf{n}_\mathcal{K})\, \phi_l \, d\mathcal{S} \\ &+ \sum_{\mathcal{S}_m^{i,n} \in \partial\mathcal{K}_j^{i,n} \cap \Gamma_B^i} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_{B,k}(\mathbf{w}_h^{i,-}, \mathbf{w}_b^{i,+}, \mathbf{n}_\mathcal{K})\, \phi_l \, d\mathcal{S} \\ &+ \sum_{\mathcal{S}_m^{i,n} \in \partial\mathcal{K}_j^{i,n} \cap \Gamma_S^i} \int_{\mathcal{S}_m^{i,n}} \mathcal{H}_{S,k}(\mathbf{w}_h^{i,-}, \mathbf{w}_s^{i,+}, \mathbf{n}_\mathcal{K})\, \phi_l \, d\mathcal{S}. \end{aligned} \tag{31}$$

In equation (30) the dependency of $\mathcal{L}_{kl}^{i,n}$ on $\hat{\mathbf{W}}^{n-1}$ stems from the integrals over the internal faces connecting the current and previous time slabs. The numerical fluxes are problem dependent and will be discussed in part II [50] for specific test problems.

4.2. Level set discretization

The level set equation can be characterized as a hyperbolic partial differential equation containing an intrinsic nonconservative product, meaning that it cannot be transformed into divergence form. This causes problems when the level set becomes discontinuous, because the weak solution in the classical sense of distributions does not exist. Thus, no classical Rankine-Hugoniot shock conditions can be defined. Although the level set is initially smooth, it can become discontinuous over time due to discontinuities in the global flow velocity advecting the level set. In order to find a disontinuous Galerkin discretization for the level set equation, valid even when level set solution and velocity become discontinuous, the theory presented in [42] is applied. For simplicity the same notation will be used as in [42].

In general, a hyperbolic system of m partial differential equations in nonconservative form in q space dimensions can be defined as:

$$U_{i,0} + F_{ik,k} + G_{ikr} U_{r,k} = 0,\ i, r = 1, \cdots, m \tag{32}$$

with U the vector of variables, F the conservative spatial flux tensor, G the nonconservative spatial flux tensor and where $(.)_{,0}$ and $(.)_{,k}, k = 1, \cdots, q$ denote partial differentation with respect to time and spatial coordinates,

respectively. The space-time DGFEM weak nonconservative formulation of this system is defined as:

$$
\begin{aligned}
0 = & \sum_{\mathcal{K}\in\mathcal{T}_h^n} \int_{\mathcal{K}} (-V_{i,0}U_i - V_{i,k}F_{ik} + V_i G_{ikr} U_{r,k}) d\mathcal{K} \\
& + \sum_{\mathcal{K}\in\mathcal{T}_h^n} \left(\int_{K(t_{n+1}^-)} V_i^L U_i^L dK - \int_{K(t_n^+)} V_i^L U_i^R dK \right) \\
& + \sum_{\mathcal{S}\in\mathcal{S}^n} \int_{\mathcal{S}} (V_i^L - V_i^R) \hat{P}_i^{nc} d\mathcal{S} \\
& + \sum_{\mathcal{S}\in\mathcal{S}^n} \int_{\mathcal{S}} \{\!\{V_i\}\!\} \left(\int_0^1 G_{ikr}(\chi(\tau; U^L, U^R)) \frac{d\chi_r}{d\tau}(\tau; U^L, U^R) d\tau \bar{n}_k^L \right) d\mathcal{S}, \qquad (33)
\end{aligned}
$$

where V denotes the vector of trial functions and χ denotes the path function. The nonconservative flux is defined as:

$$
\begin{aligned}
& \hat{P}^{nc}(U_L, U_R, v, \bar{n}^L) = \qquad (34) \\
& \begin{cases}
F_{ik}^L - \frac{1}{2}\int_0^1 G_{ikr}(\chi(\tau; U^L, U^R)) \frac{d\chi_r}{d\tau}(\tau; U^L, U^R) d\tau \bar{n}_k^L, & \text{if } S_L > v \\
\{\!\{F_{ik}\}\!\} \bar{n}_k^L + \frac{1}{2}((S_R - v)\bar{U}_i^* + (S_L - v)\bar{U}_i^* - S_L U_i^L - S_R U_i^R), & \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } S_L < v < S_R \\
F_{ik}^L + \frac{1}{2}\int_0^1 G_{ikr}(\chi(\tau; U^L, U^R)) \frac{d\chi_r}{d\tau}(\tau; U^L, U^R) d\tau \bar{n}_k^L, & \text{if } S_R < v,
\end{cases}
\end{aligned}
$$

where S_L and S_R denote the minimum and maximum wavespeeds, v denotes the grid velocity and $\bar{U}_i^*$ denotes the average star state solution. When using a linear path $\chi = U_L + \tau(U_R - U_L)$, $\frac{d\chi_r}{d\tau}(\tau; U^L, U^R) = (U^R - U^L)$.

The level set equation can be considered a special case of (32), where the state and fluxes are defined as $U = \psi_h, F = 0, G = \bar{\mathbf{a}}_h$. The following simplification can be made:

$$
\begin{aligned}
& \int_0^1 G_{ikr}(\chi(\tau; U^L, U^R)) \frac{d\chi_r}{d\tau}(\tau; U^L, U^R) d\tau \bar{n}_k^L \\
& = \int_0^1 \bar{\mathbf{a}}(\chi(\tau; \psi_h^L, \psi_h^R))(\psi_h^R - \psi_h^L) d\tau \bar{n}_k^L \\
& = - \{\!\{\bar{\mathbf{a}}_h\}\!\} [[\psi_h]]. \qquad (35)
\end{aligned}
$$

Hence, the nonconservative level set discretization becomes:

$$\begin{aligned}
&\sum_{\mathcal{K}^n_{b,\tilde{j}} \in \mathcal{I}^n_b} \int_{\mathcal{K}^n_{b,\tilde{j}}} -\frac{\partial \phi_l}{\partial t} \psi_h + \phi_l \, \bar{\mathbf{a}}_h \cdot \bar{\nabla} \psi_h \, d\mathcal{K} \\
&+ \sum_{\mathcal{K}^n_{b,\tilde{j}} \in \mathcal{I}^n_b} \left(\int_{\mathcal{K}^n_{b,\tilde{j}}(t_{n+1})} \phi_l^l \, \psi_h^l \, d\mathcal{S} - \int_{\mathcal{K}^n_{b,\tilde{j}}(t_n)} \phi_l^l \, \psi_h^r \, d\mathcal{S} \right) \\
&+ \sum_{\mathcal{S}^n_{b,\tilde{m}} \in \Gamma^n_b} \int_{\mathcal{S}^n_{b,\tilde{m}}} (\phi_l^l - \phi_l^r) \, \hat{P}^{nc} \, d\mathcal{S} \\
&- \sum_{\mathcal{S}^n_{b,\tilde{m}} \in \Gamma^n_b} \int_{\mathcal{S}^n_{b,\tilde{m}}} \{\!\{ \phi_l \}\!\} \, [\![\psi_h]\!] \, \{\!\{ \bar{\mathbf{a}}_h \}\!\} \, d\mathcal{S} = 0,
\end{aligned} \tag{36}$$

with

$$\hat{P}^{nc} = \begin{cases} +\frac{1}{2} [\![\psi_h]\!] \, \{\!\{ \bar{\mathbf{a}}_h \}\!\} & \text{if } S_L > 0 \\ +\frac{1}{2} (S_R(\psi_h^* - \psi_h^R) + S_L(\psi_h^* - \psi_h^L)) & \text{if } S_L < 0 < S_R \\ -\frac{1}{2} [\![\psi_h]\!] \, \{\!\{ \bar{\mathbf{a}}_h \}\!\} & \text{if } S_R < 0 \end{cases} \tag{37}$$

where $S_L = \min\{\bar{\mathbf{a}}_h^L \cdot \bar{\mathbf{n}}_K^L, \bar{\mathbf{a}}_h^R \cdot \bar{\mathbf{n}}_K^L\}$ and $S_R = \max\{\bar{\mathbf{a}}_h^L \cdot \bar{\mathbf{n}}_K^L, \bar{\mathbf{a}}_h^R \cdot \bar{\mathbf{n}}_K^L\}$ the minimum and maximum wavespeeds and where the star state level set value is defined as:

$$\psi_h^* = \begin{cases} \psi_L & \text{if } (S_L + S_R)/2 > 0 \\ \psi_R & \text{if } (S_L + S_R)/2 < 0. \end{cases} \tag{38}$$

At boundary faces the level set boundary conditions (10) are enforced by specifying the right state as:

$$\begin{aligned}
\psi^r(t, \bar{\mathbf{x}}) &= \psi^l(t, \bar{\mathbf{x}}) \\
\bar{\mathbf{a}}^r(t, \bar{\mathbf{x}}) &= \bar{\mathbf{a}}^l(t, \bar{\mathbf{x}}) - 2(\bar{\mathbf{a}}^l(t, \bar{\mathbf{x}}) \cdot \mathbf{n}_{\mathcal{K}}) \mathbf{n}_{\mathcal{K}}, \text{ for } (t, \bar{\mathbf{x}}) \in \mathcal{Q}.
\end{aligned} \tag{39}$$

4.3. Pseudo-time integration

By augmenting the flow equations with a pseudo-time derivative, the discretized equations (30) are extended into pseudo-time, resulting in:

$$M_{ml}^{i,n} \frac{\partial \hat{W}_{km}^{i,n}}{\partial \tau} + \mathcal{L}_{kl}^{i,n}(\hat{\mathbf{W}}^n, \hat{\mathbf{W}}^{n-1}) = 0, \tag{40}$$

Algorithm 5 Pseudo-time integration method for solving the non-linear algebraic equations in the space-time discretization.

1. *Initialize first Runge-Kutta stage:* $\bar{\mathbf{W}}^{i,(0)} = \hat{\mathbf{W}}^{i,n}$.
2. *Calculate* $\bar{\mathbf{W}}^{i,(s)}, s = 1, \cdots, 5$*:*

$$(1 + \alpha_s \lambda)\bar{\mathbf{W}}^{i,(s)} = \bar{\mathbf{W}}^{i,(0)} + \alpha_s \lambda \left(\bar{\mathbf{W}}^{i,(s-1)} - \Delta t \, (M^{i,n})^{-1} \, \mathcal{L}(\bar{\mathbf{W}}^{i,(s-1)}, \bar{\mathbf{W}}^{i,n-1}) \right)$$

3. *Update solution:* $\hat{\mathbf{W}}^{i,n} = \bar{\mathbf{W}}^{i,(5)}$.

using the summation convention on repeated indices, and with

$$M_{ml}^{i,n} = \int_{\mathcal{K}_j^{i,n}} \phi_l \phi_m \, d\mathcal{K} \tag{41}$$

the mass matrix. To solve (40) a five stage semi-implicit Runge-Kutta iterative scheme is used [29, 61] as defined in Algorithm 5. Starting from a guess for the initial solution, the solution is iterated in pseudo-time until a steady state is reached, which is the real time solution of the space-time discretization. Here $\lambda = \Delta\tau/\Delta t$ denotes the ratio of pseudo time and physical time step, and the coefficients α_s are defined as: $\alpha_1 = 0.0791451$, $\alpha_2 = 0.163551$, $\alpha_3 = 0.283663$, $\alpha_4 = 0.5$, $\alpha_5 = 1.0$. The physical time step Δt is defined globally by using a Courant-Friedrichs-Levy (CFL) condition:

$$\Delta t = \frac{CFL_{\Delta t} \, h}{S_{max}}, \tag{42}$$

with $CFL_{\Delta t}$ the physical CFL number, h the inradius of the space projection of the element and S_{max} the maximum value of the wave speed on the faces. The five stage semi-implicit Runge-Kutta iterative scheme is also used for solving the discretized level set equation.

5. Two fluid algorithm

The two fluid algorithm is defined in Algorithm 6. The operations at the initialization, in the inner iteration and at the time slab update are illustrated for two space-time dimensions in Figures 17, 18 and 19, respectively. In

Algorithm 6 Computational steps in the two fluid method. Lines 1-6 detail the initialization, lines 13-22 the inner iteration and lines 8-12 time slab update. The initialization, inner iteration and time slab update are illustrated for two space-time dimensions in Figures 17, 18 and 19.

1. $n = 0$
2. *Create background mesh* $\mathcal{T}_b^{n-1}$
3. *Initialize level set* $\psi_h^{n-1}(\mathbf{x})$ *on* $\mathcal{T}_b^{n-1}$
4. *Initialize level set velocity* $\bar{\mathbf{a}}_h^{n-1}(\mathbf{x})$ *on* $\mathcal{T}_b^{n-1}$
5. *Create refined mesh* $\mathcal{T}_h^{i,n-1}$ *based on* $\psi_h^{n-1} = 0$
6. *Initialize flow field* $\mathbf{w}_h^{i,n-1}(\mathbf{x})$ *on* $\mathcal{T}_h^{i,n-1}$
7. *WHILE* $n < N_t$ *DO*
8. *Create background mesh* $\mathcal{T}_b^n$
9. *Initialize level set* $\psi_h^n(\mathbf{x})$ *on* $\mathcal{T}_b^n$ *as* $\psi_h^{n-1}(t_n, \bar{\mathbf{x}})$ *on* $\mathcal{T}_b^{n-1}$ *(46)*
10. *Initialize level set velocity* $\bar{\mathbf{a}}_h^n(\mathbf{x})$ *on* $\mathcal{T}_b^n$ *as* $\bar{\mathbf{a}}_h^{n-1}(t_n, \bar{\mathbf{x}})$ *on* $\mathcal{T}_b^{n-1}$ *(47)*
11. *Create refined mesh* $\mathcal{T}_{h,0}^{i,n}$ *based on* $\psi_h^n = 0$
12. *Initialize flow field* $\mathbf{w}_{h,0}^{i,n}(\mathbf{x})$ *on* $\mathcal{T}_{h,0}^{i,n}$ *as* $\mathbf{w}_{h,0}^{i,n-1}(t_n, \bar{\mathbf{x}})$ *on* $\mathcal{T}_h^{i,n-1}$ *(43)*
13. $k = 0$
14. *WHILE two fluid mesh has not converged:* $|e_k - e_{k-1}| > \epsilon_{IF}$ *DO*
15. *Solve* ψ_h^n *on* $\mathcal{T}_b^n$
16. *Calculate level set interface error* $e_k = \|\psi_h^n\|_2^{IF}$
17. *Create refined mesh* $\mathcal{T}_{h,k}^{i,n}$ *based on* $\psi_h^n = 0$
18. *Initialize flow field* $\mathbf{w}_{h,k}^{i,n}(\mathbf{x})$ *on* $\mathcal{T}_{h,k}^{i,n}$ *as* $\mathbf{w}_h^{i,n-1}(t_n, \bar{\mathbf{x}})$ *on* $\mathcal{T}_h^{i,n-1}$ *(43)*
19. *Solve* $\mathbf{w}_{h,k}^{i,n}(t, \bar{\mathbf{x}})$ *on* $\mathcal{T}_{h,k}^{i,n}$
20. *Initialize level set velocity* $\bar{\mathbf{a}}_h^n(\mathbf{x})$ *on* $\mathcal{T}_b^n$ *as* $\mathbf{u}_{h,k}^{i,n}(\mathbf{x})$ *on* $\mathcal{T}_h^{i,n}$ *(44)*
21. $k = k + 1$
22. *END DO*
23. $n = n + 1$
24. *END DO*

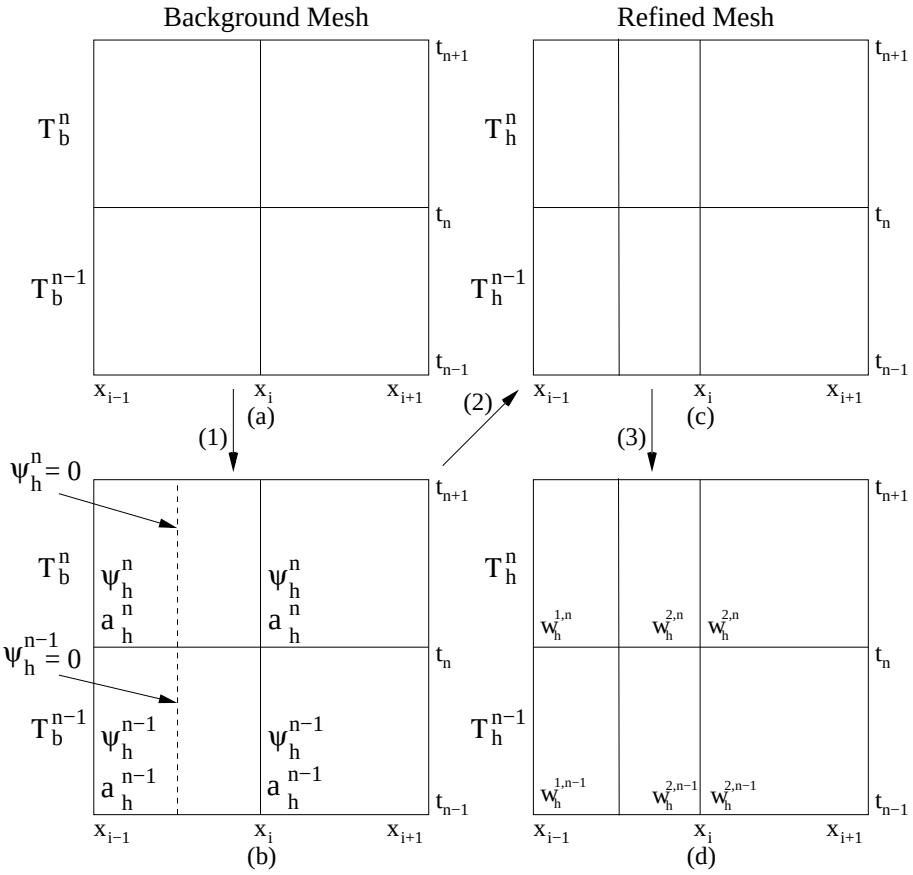

Figure 17: At initialization, first the background mesh is created. Because the solution from the previous time step is required in the evalution of the numerical flux at the time slab face, the background mesh is conveniently composed of a current (n) and a previous ($n-1$) time slab (a). Next the level set is initialized on the background mesh (b). Based on the 0-level set, the background mesh is refined to obtain the refined mesh (c). Finally, in all elements of the refined mesh the flow variables are initialized (d). The initialization is performed on the current as well as a previous time slab.

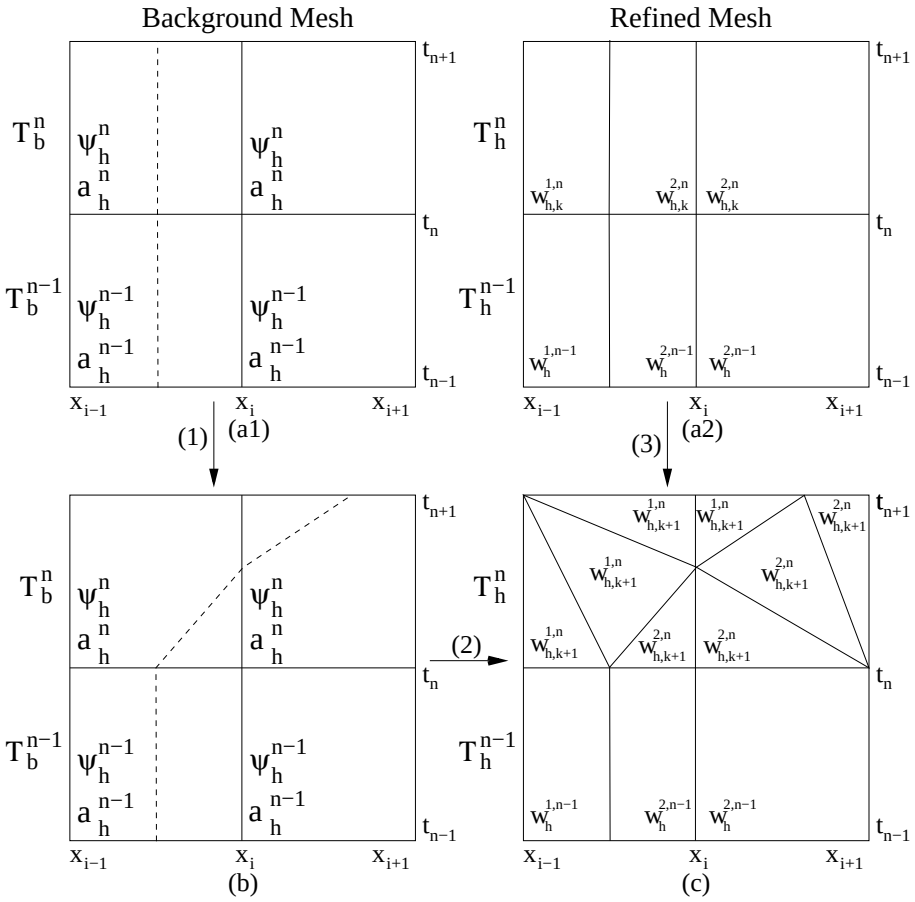

Figure 18: In the inner iteration, given level set and flow solutions on the background and refined meshes (a1, a2), first the level set is solved on $\mathcal{T}_b^n$ (b). Based on the 0-level set the background mesh is refined to obtain a new two fluid mesh $\mathcal{T}_h^n$, on which the flow field is reinitialized and solved (c). Finally, the level set velocity is reinitialized with the flow velocity.

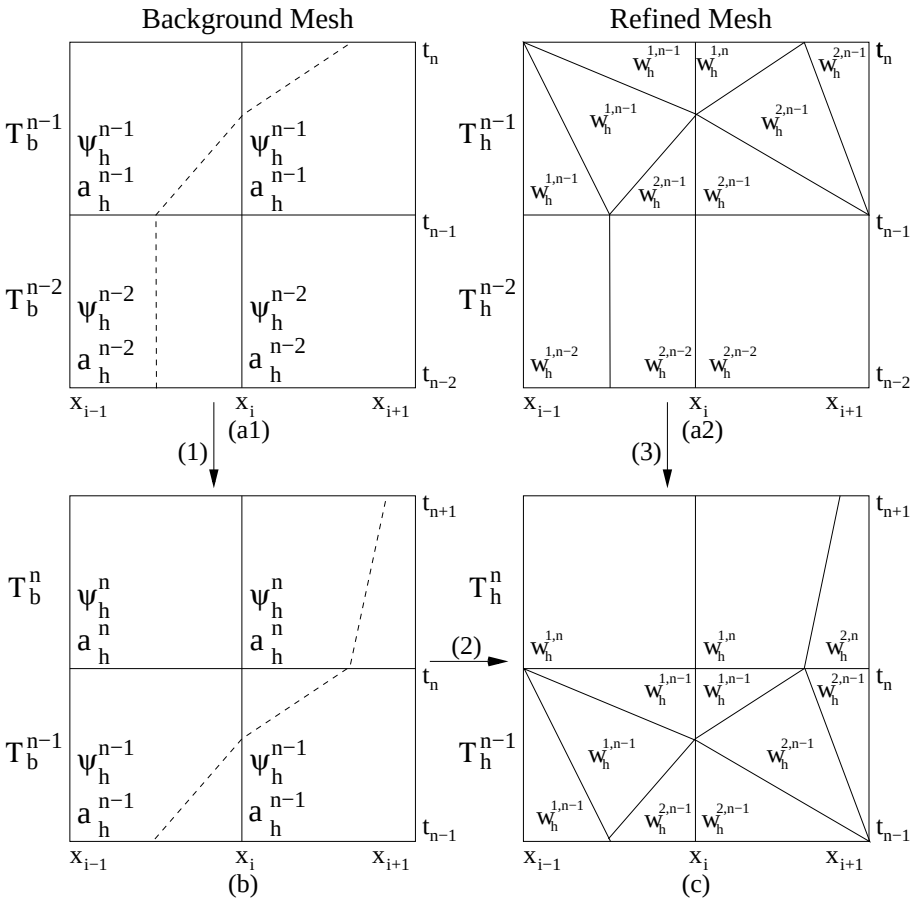

Figure 19: When moving to the next time slab, given level set and flow solutions on the background and refined meshes (a1, a2), first a new background mesh $\mathcal{T}_b^n$ is created, on which a level set is initialized and solved (b). Based on the 0-level set, the background mesh is refined to obtain the two fluid mesh $\mathcal{T}_h^n$, on which the flow field is initialized (c).

the inner iteration and at the time slab update the flow approximation $\mathbf{w}_h^{i,n}$ is reinitialized with the solution average from the previous time slab:

$$\mathbf{w}_h^{i,n}(t, \bar{\mathbf{x}}) = \bar{\mathbf{w}}_h^{i,n-1}(t_n, \bar{\mathbf{x}}). \tag{43}$$

When, for a fluid type, no solution exists in the previous time slab, the element is marked as such and is reinitialized at a later stage by using the reinitialized solution from a neighboring element in the new timeslab. To make the flow reinitialization compatible with the element merging it is preceded by a projection step, in which the solution in each merged element is projected onto the refined elements of which it is composed. After solving the flow equations the level set velocity $\mathbf{a}_h^n$ is reinitialized as:

$$\int_{\mathcal{K}_{b,\tilde{j}}^n} \bar{\mathbf{a}}_h^n(\mathbf{x})\phi_l(\mathbf{x})\, d\mathcal{K} = \int_{\mathcal{K}_{b,\tilde{j}}^n} \mathbf{u}_{h,k}^n(\mathbf{x})\phi_l(\mathbf{x})\, d\mathcal{K}. \tag{44}$$

In order to evaluate the flow velocity $\mathbf{u}_{h,k}^n$ on the background mesh, for every element $\mathcal{K}_j^{i,n}$ in the refined mesh $\mathcal{T}_h^n$, a child to parent mapping $H_{\mathcal{K}_j^{i,n}}$ is defined:

$$H_{\mathcal{K}_j^{i,n}} = G_{\mathcal{K}_j^{i,n}}^{-1} \circ G_{\mathcal{K}_{b,\tilde{j}}^n}, \tag{45}$$

where $G_{\mathcal{K}_{b,\tilde{j}}^n}$ and $G_{\mathcal{K}_j^{i,n}}$ are the mappings from the reference element to the physical space of the background and the child element, respectively. The mapping $H_{\mathcal{K}_j^{i,n}}$ maps the element $\mathcal{K}_j^{i,n}$ to its parent element $\mathcal{K}_{b,\tilde{j}}^n$ in the background mesh $\mathcal{T}_b^n$. The inverse mappings $G_{\mathcal{K}_{b,\tilde{j}}^n}^{-1}$ always exists, since the background elements are by construction never degenerate. The child to parent mapping is illustrated in Figure 20. At the time slab update the level set approximation ψ_h^n is reinitialized as:

$$\psi_h^n(t, \bar{\mathbf{x}}) = \psi_h^{n-1}(t_n, \bar{\mathbf{x}}) \tag{46}$$

and the level set velocity approximation $\mathbf{a}_h^n$ is reinitialized as:

$$\bar{\mathbf{a}}_h^n(t, \bar{\mathbf{x}}) = \bar{\mathbf{a}}_h^{n-1}(t_n, \bar{\mathbf{x}}). \tag{47}$$

6. Discussion

A space-time discontinuous Galerkin finite element method for two fluid flows has been presented which combines aspects of front tracking and front

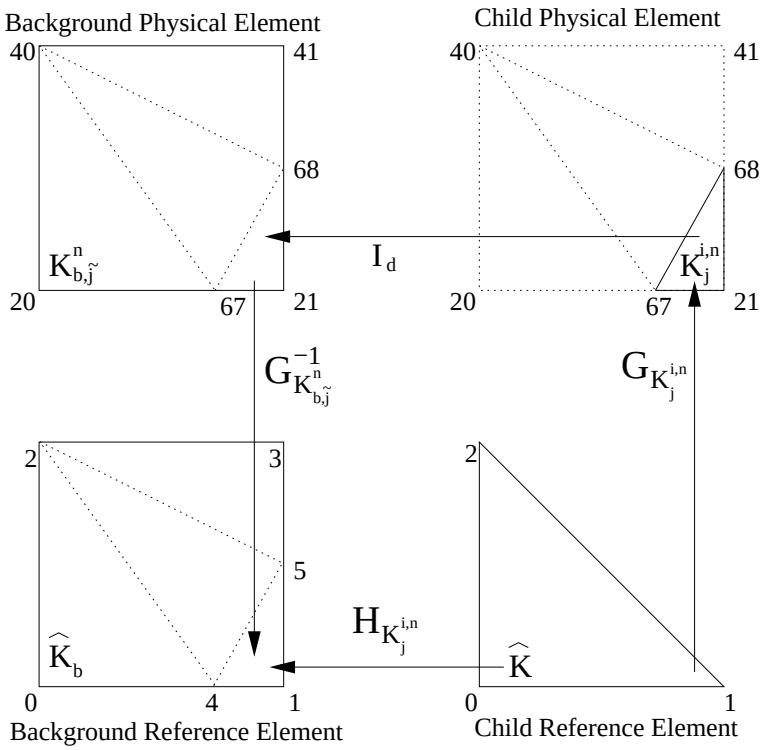

Figure 20: The child to parent mapping $H_{\mathcal{K}_j^{i,n}}$ is composed of the mapping $G_{\mathcal{K}_j^{i,n}}$ from the child reference element to child physical element and the inverse mapping $G^{-1}_{\mathcal{K}^n_{b,\tilde{j}}}$ from background physical element to the background reference element. The child physical element is connected to the background physical element through the identity mapping I_d.

capturing methods with cut-cell mesh refinement and a STDG discretization. It is anticipated that this scheme can accurately solve smaller scale problems where the interface shape is of importance and where complex interface physics are involved. Special attention has been paid to making the scheme as generic as possible to allow for future implementations in higher dimensions. The STDG discretization ensures that the scheme is conservative as long as the numerical fluxes are conservative. The problem with cut-cell mesh refinement with small cells is solved by using element merging. Topological changes such as merging and coalescence are handled in the method due to the level set method. Care must, however, be taken since topological changes may conflict with the conservativity of the scheme especially on coarse meshes and may cause non-convergence of the flow solution. In part II [50] the method will be tested on a number of test problems.

Acknowledgments

The author is kindly indebted to V.R.Ambati, A.Bell and L.Pesch for the valueable discussions, suggestions and support.

References

[1] D. Adalsteinsson & J.A. Sethian, A fast level set method for propagating interfaces, J. Comp. Phys. 118 (1995) 269–277.

[2] H.T. Ahn, M. Shashkov, Adaptive moment-of-fluid method, J. Comp. Phys. 228 (2009) 2792–2821.

[3] A.S. Almgren et al., A Cartesian mesh projection method for the incompressible Euler equations in complex geometries, SIAM J. Sci. Comp. 18 (1997) 1289–1309.

[4] V.R. Ambati, O. Bokhove, Space-time discontinuous Galerkin finite element method for shallow water flows, J. Comp. Appl. Math., 204 (2007) 452-462.

[5] D.M. Causon et al., Calculation of shallow water flows using a Cartesian cut element approach, Adv. Water Resour. 23 (2000) 545–562.

[6] B. Cockburn, G.E. Karniadakis, C.W. Shu, Discontinuous Galerkin methods theory, computation & applications, Lecture Notes in Computational Science & Engineering. Vol.11, Springer, Berlin, 2000.

[7] W.J. Coirier, K.G. Powell, An accuracy assessment of Cartesian-mesh approaches for Euler equations, J. Comp. Phys. 117 (1995) 121–131.

[8] B. Cuenot, J. Magnaudet, B. Spennato, The effects of slightly soluble surfactants on the flow around a spherical bubble, J. Fluid Mech. 339 (1997) 25–53.

[9] B.J. Daly, Numerical study of the effect of surface tension on interface instablility, Phys. Fluids 12 (1969) 1340.

[10] D.S. Dandy, L.G. Leal, Buoyancy-driven motion of a deformable drop through a quiescent liquid at intermediate Reynolds numbers, J. Fluid Mech. 339 (1989) 161–192.

[11] D. de Zeeuw, K.G. Powell, An adaptively refined Cartesian mesh solver for the Euler equations, J. Comp. Phys. 104 (1993) 56–68.

[12] K.J. Fidkowski, D.L. Darmofal, A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations, J. Comp. Phys. 225 (2007) 1653–1672.

[13] M.J. Fritts, W. Cowley, H.E. Trease (eds.), The Free Lagrange Method, Lect. Notes Phys. Vol. 238, Springer-Verlag, New York, 1985.

[14] J. Fukai et al., Wetting effects on the spreading of a liquid droplets colliding with a flat surface: experiment & modeling, Phys. Fluids A. 5 (1993) 2588–2599.

[15] D.E. Fyfe, E.S. Oran, M.J. Fritts, Surface tension & viscosity with Lagrangian hydrodynamics on a triangular mesh, J. Comp. Phys. 76 (1988) 349–384.

[16] J. Glimm et al., Front tracking for hyperbolic systems, Adv. Appl. Math. 2 (1981) 91-119.

[17] J. Glimm, O. McBryan, A computational model for interfaces, Adv. Appl. Math. 6 (1985) 422–435.

[18] J. Glimm et al., Three-dimensional front tracking, SIAM J. Sci. Comp. 19 (1998) 201–225.

[19] J. Glimm et al., Simple Front Tracking, Contemp. Math. 238 (1999) 133–149.

[20] J. Glimm et al., Conservative front tracking with improved accuracy, SIAM J. Num. Anal. 41-5 (2003) 1926–1947.

[21] D.M. Greaves, Simulation of viscous water column collapse using adapting hierarchial grids, Int. J. Num. Meth. Fluids 50 (2005) 693–711.

[22] D.M. Greaves, Viscous wave interaction with structures using adapting quadtree grids & cartesian cut cells, ECCOMAS CFD 2006, Delft.

[23] F.H. Harlow, J.E. Welch, Numerical calculation of time-dependent viscous incompressible flow, Phys. Fluids 8 (1965) 2182.

[24] C.V. Hirt, B.D. Nichols, Volume of fluid (VOF) methods for the dynamics of free boundaries, J. Comp. Phys 39 (1981) 201–255.

[25] C. Hu, C.W.- Shu, A Discontinuous Galerkin Finite Element Method for Hamilton-Jacobi Equations, NASA/CR-1998-206903 (1998), Hampton.

[26] J.M. Hyman, Numerical methods for tracking interfaces, Phys. D. 12 (1984) 396–407.

[27] H. Johansen, P. Colella, A Cartesian mesh embedded boundary method for Poisson's equation on irregular domains, J. Comp. Phys. 147 (1998) 60–85.

[28] C.M. Klaij, J.J.W. van der Vegt, H. van der Ven, Space-time discontinuous Galerkin method for the compressible Navier-Stokes equations, J. Comp. Phys. 217 (2006) 589–611.

[29] C.M. Klaij, J.J.W. van der Vegt, H. van der Ven, Pseudo-time stepping methods for space-time discontinuous Galerkin discretizations of the compressible Navier-Stokes equations, J. Comp. Phys. 219 (2006) 622–643.

[30] L. Krivodonova et al., Shock detection & limiting with discontinuous Galerkin methods for hyperbolic conservation laws, Appl. Num. Math. 48 (2004) 323–338.

[31] M. Kucharik, J. Limpouch, R. Liska, Laser plasma simulations by Arbitrary Lagrangian Eulerian method, J. de Phys. 133 (2006) 167–169.

[32] R.J. LeVeque, K.-M. Shyue, 2-dimensional front tracking based on high resolution wave propagation methods, J. Comp. Phys. 123 (1996) 354–368.

[33] H. Luo, J.D. Baum, R. Lohner, A Hermite WENO-based limiter for discontinuous Galerkin method on unstructured grids, J. Comp. Phys. 225 (2007) 686–713.

[34] J. Magnaudet, M. Rivero, J. Fabre, Accelerated flows around a rigid sphere or a spherical bubble. Part I: Steady straining flow, J. Fluid Mech. 284 (1995) 97–136.

[35] G. Moretti, B. Grossman, F. Marconi, A complete numerical technique for the calculation of three dimensional inviscid supersonic flow, Rep. No. 72-192 (1972), American Institute for Aerodynamics & Astronautics, New York.

[36] S.J. Osher, J.A. Sethian, Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations, J. Comp. Phys. 79 (1988) 12–49.

[37] S.J. Osher, C.W. Shu, High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations, SIAM J. Num. Anal. 28 (1991) 907–922.

[38] R.B. Pember et al., An adaptive Cartesian mesh method for unsteady compressible flow in irregular regions, J. Comp. Phys. 120 (1994) 278-304.

[39] D. Peng et al., A PDE-Based Fast Local Level Set Method, J. Comp. Phys. 155 (1999) 410–438.

[40] E.G. Puckett, J.S. Saltzman, A 3D adaptive mesh refinement algorithm for interfacial gas dynamics, Physica D. 60 (1992) 84-93.

[41] J. Quirk, An alternative to unstructured meshes for computing gas dynamic flows around arbitrarily complex two-dimensional bodies, Comp.Fluids 23 (1994) 125–142.

[42] S. Rhebergen, O. Bokhove, J.J.W. van der Vegt, Discontinuous Galerkin finite element methods for hyperbolic nonconservative partial differential equations, J. Comp. Phys. 227 (2008) 1887–1922.

[43] R.D. Richtmyer, K.W. Morton, Difference Methods for Initial-Value Problems, Inter-science, New york, 1967.

[44] J.M. Rudman, Volume-tracking methods for interfacial flow calculations, Int. J. Numer. Heat Fluid Flow 24 (1997) 671–691.

[45] G. Ryskin, L.G. Leal, Numerical solution of free-boundary problems in fluid mechanics, Part 2: Buoyancy-driven motion of a gas bubble through a quiescent liquid, em J. Fluid Mech. 148 (1984) 19–36.

[46] R. Saurell, R. Abgrall, A simple method for compressible multifluid flows, SIAM J. Sci. Comp. 21 (1998) 1115-1145.

[47] R. Scardovelli, S. Zaleski, Direct numerical simulation of free-surface & interfacial flow, Annu. Rev. Fluid Mech. 31 (1999) 567–603.

[48] J.A. Sethian, Level Set Methods, Cambridge Univ. Press, 1996.

[49] J.A. Sethian, P. Smereka, Level set methods for fluid interfaces, Annu. Rev. Fluid Mech. 35 (2003) 341–372.

[50] W.E.H. Sollie, J.J.W. van der Vegt, Two Fluid Space-Time Discontinuous Galerkin Finite Element Method, Part II: Applications, (2009).

[51] J.J. Sudirham, J.J.W. van der Vegt, R.M.J. van Damme, Space-time discontinuous Galerkin method for advection-diffusion problems on time-dependent domains, Appl. Num. Math. 56 (2006) 1491–1518.

[52] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible two-phase flow, J. Comp. Phys. 114 (1994) 146–159.

[53] M. Sussman, E.G. Puckett, A coupled level set & volume-of-fluid method for computing 3D & axisymmetric incompressible two-phase flows, J. Comp. Phys. 162 (2000) 301–337.

[54] G. Tryggvason, S.O. Unverdi, Computations of three-dimensional Rayleigh-Taylor instability, Phys. Fluids A. 5 (1990) 656–659.

[55] P.G. Tucker, Z. Pan, A Cartesian cut element method for incompressible viscous flow, Appl. Math. Modell. 24 (2000) 591–606.

[56] H.S. Udaykumar et al., Multiphase dynamics in arbitrary geometries on fixed Cartesian meshes, J. Comp. Phys. 137 (1997) 366–405.

[57] H.S. Udaykumar et al., A sharp interface Cartesian mesh method for simulating flows with complex moving boundaries, J. Comp. Phys. 174 (2001) 345–380.

[58] H.S. Udaykumar, R. Mittal, P. Rampunggoon, Interface tracking finite volume method for complex solid-fluid interactions on fixed meshes, Comm. Nummer. Meth. Eng. 18 (2002) 89–97.

[59] S.O. Unverdi, G. Tryggvason, Computations of multi-fluid flows, Phys. Fluids D 60 (1992) 70–83.

[60] S.O. Unverdi, G. Tryggvason, A front-tracking method for viscous, incompressible multi-fluid flows, J. Comp. Phys. 100 (1992) 25–37.

[61] J.J.W. van der Vegt, H. van der Ven, Space-time discontinuous Galerkin finite element method with dynamic mesh motion for Inviscid Compressible Flows, J. Comp. Phys. 182 (2002) 546–585.

[62] J.J.W. van der Vegt, Y. Xu, Space-time discontinuous Galerkin method for nonlinear water waves, J. Comp. Phys. 224 (2007) 17–39.

[63] G. Yang et al., A Cartesian cut element method for compressible flows. Part A: Static body problems, Aeronaut. J. 2 (1997) 47–56.

[64] G. Yang et al., A Cartesian cut element method for compressible flows. Part B: Moving body problems, Aeronaut. J. 2 (1997) 57–65.

[65] G. Yang, D.M. Causon, Ingram, Calculation of compressible flows about complex moving geometries using a three-dimensional Cartesian cut element method, Int. J. Numer. Meth. Fluids 33 (2000) 1121–1151.

[66] T. Ye et al., An accurate Cartesian mesh method for viscous incompressible flows with complex immersed boundaries, J. Comp. Phys. 156 (1999) 209–240.

[67] S.S. Young et al., A basic experimental study of sandwich injection moulding with sequential injection, Pol. Eng. Sci. 20 (1980) 798–804.