# Adaptive Mesh Refinement algorithm based on dual trees for cells and faces for multiphase compressible flows

Kevin Schmidmayer, Fabien Petitpas, Eric Daniel

# Adaptive Mesh Refinement algorithm based on dual trees for cells and faces for multiphase compressible flows

Kevin Schmidmayer[a,*], Fabien Petitpas[b], Eric Daniel[b]

[a] *Division of Engineering and Applied Science, California Institute of Technology*
*1200 E. California Blvd., Pasadena, CA 91125, USA*
[b] *Aix Marseille Univ, CNRS, IUSTI, Marseille, France*

**Abstract**

A novel adaptive mesh refinement method is proposed. The novelty of the method lies in using a dual data structure with two trees: A classical one for the computational cells and an extra one dedicated to computational cell faces. This new dual structure simplifies the algorithm, making the method easy to implement. It results in an efficient adaptive mesh refinement method that preserves an acceptable memory cost. This adaptive mesh refinement method is then applied to compressible multiphase flows in the framework of diffuse-interface methods. Efficiency of the method is demonstrated thanks to computational results for different applications: Transport, shock tube, surface-tension flow, cavitation and water-droplet atomization, in one and multi-dimensions. The test cases are performed with the open-source code ECOGEN and with quantitative comparisons regarding non-adaptive mesh refinement methods to analyze benefits. A discussion specific to parallel computing is also presented.

*Keywords:* adaptive mesh refinement, diffuse interface, multiphase flow, compressible flow, cell tree, face tree

## 1. Introduction

In computational fluids dynamics, the accuracy of results is conditioned by the refinement level of the computational grid. Nevertheless, the finer is the grid, the more expensive is the computational cost regarding CPU time as well as memory. For the computation of steady flows, the use of unstructured grid and mesh refinement techniques at well-defined locations can lead to very accurate results (see for example simulations around hydrofoils [19]). For the computations of strongly unsteady flows with shock waves or traveling interfaces, achievement of accurate results is conditioned by the use of a very small cell size. Thus, a large amount of computational time is wasted to compute solutions in cells where almost

*Corresponding author
*Email addresses:* kevin.schmidmayer@gmail.com (Kevin Schmidmayer), fabien.petitpas@univ-amu.fr (Fabien Petitpas), eric.daniel@univ-amu.fr (Eric Daniel)

nothing occurs. On the strength of these observations, the use of Adaptive Mesh Refinement (AMR) techniques represents an interesting option to reduce the CPU time cost when the numerical solution of complex flows is sought. When an AMR method is embedded in a computational fluid dynamics code, the computational grid is dynamically adapted in order to be refined where it is necessary and to maintain a coarse grid elsewhere.

AMR methods are suitable and already massively used for many applications: Magnetohydrodynamics [2, 10], incompressible multiphase flows as for the droplet motions in a microchannel [7] and the atomization of liquid impinging jets [32], compressible multiphase flows as for the leakage of gas from a liquefied-petroleum-gas storage cavern [28] or for bubble dynamics [40], etc.

The analysis of literature shows there are mainly three existing approaches. The first one is an approach where the entire computational domain is represented by a coarse base grid and where more resolution is required, finer and nested grids are laid over coarser grids [3, 4]. This approach is known as patch-based. An example is given on the left image of Figure 1. In the second approach, the refinement occurs on an individual cell rather than on a complete cell grid, and this directly defines a tree of cells [41]. Each cell can be refined or unrefined independently of others. The mesh refinement occurs locally where it is necessary, then at every level of the tree, the mesh may have a non-uniform and very flexible shape. This second approach is known as cell-based. An example is given on the right image of Figure 1. The third approach takes advantages of the two first in a hybrid version known as block-based.

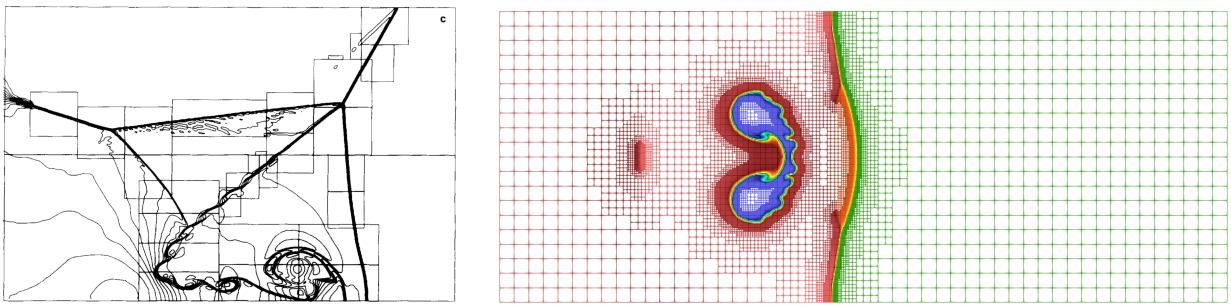

Figure 1: Representative images of two of the three main AMR approaches: On the left, shock reflection off an oblique wedge for patch-based approach (image from [3], © 1989, Journal of Computational Physics) and on the right, air shock on a helium bubble for cell-based approach.

These AMR approaches are implemented in numerous existing packages. Without being exhaustive, patch-based approach is used in codes such as Chombo [9], SAMRAI [18] or recently AMReX [43]. One can find implementations of cell-based approach in Gerris/Basilisk [31] or RAMSES [39] for example. While block-based approach is found in codes such as PARAMESH [23], FLASH [14], NIRVANA [42], ForestClaw [5] or p4est [6].

The aim of this paper is dual. The first goal is to present a new AMR method in the context of cell-tree approach. The second goal is to present how this method, when embedded in a computational code, can be used to simulate multiphase compressible flows. This last point is ensured by coupling the proposed AMR method with numerical specificities of multiphase models in the framework of the diffuse-interface theory [34].

The cell-based approach is originally appealing because of its flexibility to adapt to strongly unsteady flows. But this flexibility is paid by the fact that standard grid-based solvers cannot be used directly on a tree. When coupling with a finite volume scheme, the flux computations and the time-stepping strategy on such a tree are different from those on a grid, it means that the flow solver is slightly dependent on the level computed. In addition, accessing neighboring cells is more difficult in a tree than in an array and scanning the tree to access the nearest neighbors may be an expensive procedure. Khokhlov [20] proposed some improvements to the method: The first one consists of using a Fully Threaded Tree (FTT) where cells have not only knowledge of their child cells but also of their parent cells and neighboring cells. This thread provides an efficient parallel access to information on a tree. Besides, this FTT turns out to be useful for the dynamical refinement based on physical variations between neighboring cells; refinement generally ensured by tracking discontinuities such as shock waves or contact discontinuities. Khokhlov also improves the memory cost for maintaining the tree by regrouping cells in a so-called "oct" structure.

When dealing with compressible multiphase flows, especially in the framework of diffuse-interface methods [34], some specific features of the models have to be considered before being solved with AMR techniques:

- First, each phase is described by a set of partial differential equations (typically Navier-Stokes equations) and its own equation of state. This leads to a large number of evolution equations in conservative or non-conservative form to solve [16, 30].

- Second, split hyperbolic models may be solved, which introduce a large number of fluxes to compute between computational cells [13, 38].

Moreover, the numerical solutions of these multiphase flow equations require iterative solvers for relaxation procedures [15, 36] and expensive Riemann solvers are sometimes needed to account for real material effects [22]. These specific points complicate the writing of a code and do not facilitate the use of an AMR technique. In other words, complex compressible, multiphase, multiphysics flows (including several phases, solids, phase transition, chemical reactions, viscosity, surface tension...) are fully described by complex mathematical models and the numerical methods that solve these models involve a significant number of operations into cells, but also between cells. Embedding such numerical method in an AMR tool without precaution may lead to very bad efficiency. This is the case when using cell-tree methods, even in the fully threaded tree framework. Indeed, a lot of operations are required to find the neighboring cells. This becomes a critical point when the algorithm is used to solve these complex models coupled with high-order methods where the procedure to find the neighbors occurs many times at each time step (flux computations at each stage of the high-order method, gradient computations for numerical purpose or physical description, etc.).

In the novel cell-based AMR method presented in this paper, the AMR algorithm is simplified and the searching procedure of neighboring cells disappears. The key point is the role played by cell faces (geometrical contour): Obviously for the flux computations and also because these faces naturally define neighbors between two cells. This new method extends FTT approach on two points: The cell-tree structure is slightly modified and a second tree is

used to store information on cell faces. The addition of this second face-tree structure reduces the number of operations during the time-step integration and simplifies the algorithm. The memory involved is reasonably increased. Reasonably because the number of additional information stored for each cell face is relatively small in comparison to what is required into a cell. The AMR algorithm is then extended to take into account specific features of multiphase flow models.

The paper is organized as follows: First, the extended AMR data structure is described. Second, the general AMR algorithm in the context of finite volume scheme (coupled with Riemann solvers for flux computations) is presented. The time-stepping strategy, the advancing and the mesh-refinement procedures are detailed. Third, the extension of the AMR method to the diffuse-interface, multiphase-flow model developed in Schmidmayer et al. [38] is presented. The last part of the paper is devoted to illustrate the interests of the method on typical test cases: Transport, shock tube, surface-tension flow, cavitation and water-droplet atomization in one and multi-dimensions. Each test is performed with quantitative comparisons regarding exact solutions or results using non-AMR method in order to analyze the benefits of this new method. A discussion specific to parallel computing is also presented in this last section.

## 2. Description of AMR data structure

Let us first recall the data structure of AMR method based on cell trees. A computational cell is represented by a node at a given level in a tree. Each node of a tree is linked thanks to threads to:

- A parent node representing a computational cell at lower level. The root of a tree is a particular node with no parent.
- A given number of child nodes representing computational cells at higher level. The number of child nodes depends on the geometry and the dimension of the problem. A leaf of a tree is a particular node with no child and the computation of physical quantities (not linked to AMR) only occurs on leaf nodes.

Each cell may be split into a given number of child cells and the condition of having at the maximum one level of difference between each neighboring cell has to be respected, it ensures a 2:1 size ratio in each direction for neighboring cell. An example of a tree representing data for a 1D AMR scheme is shown on the left part of Figure 2 and an example of possible splitting in 1D/2D/3D Cartesian grid is shown on its right part.

### *2.1. Recall of Khokhlov's method [20]: Fully Threaded Tree*

The tree structure is one obvious structure to define and optimize the cell-data storage in an AMR method. Its flexibility allows refinement and unrefinement *via* destruction and reconstruction of chosen nodes in the tree with no need to regenerate the entire mesh. The cornerstone of such method lies in the chosen way to browse cells in the tree, which can be a complex and expensive operation depending on the links between cells. In its simplest

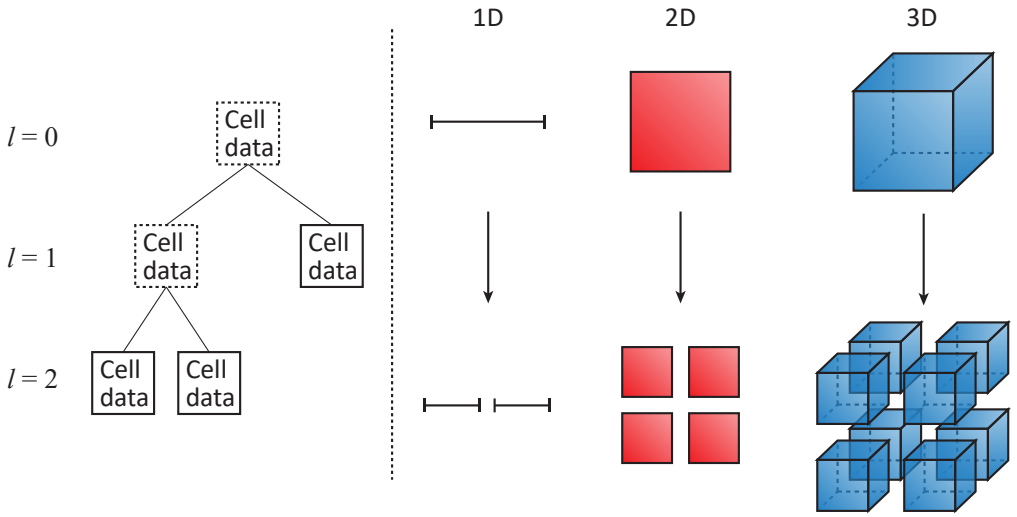


Figure 2: Figure split in two parts. On the left, an example of a tree representing the data structure for a 1D AMR scheme. "$l$" expresses the level in the tree. On the right, possible cell splitting. A 1D, 2D and 3D cell will give birth in a Cartesian grid to 2, 4 and 8 child cells, respectively.

version, a node can only be accessed from browsing the tree from its root. One can easily understand that the simple operation that consists in locating neighbors of a given cell (for example to compute inter-cell fluxes) rapidly becomes a source of computational waste. An alternative solution is the use of a so-called Fully Threaded Tree (FTT) where each node has the knowledge of its parent node, child nodes and neighbor nodes. This improvement leads to an easy browsing of the tree in every direction: From parent to child, child to parent and between neighbors. This ability to browse the tree in all directions has the drawback to considerably increase the amount of memory by the addition of multiple pointers acting as threads. It also leads to more complicated maintenance operations on the tree when refinement and unrefinement occur.

Khokhlov [20] proposed to group cells into octs in order to limit extra storage due to links between nodes and to limit maintenance operation costs. In 3D, each oct contains 8 organized cells, a pointer to its parent cell at lower level and 6 pointers to parent cells of neighboring octs. Each cell contains physical flow variables and a pointer to a child oct at higher level. Most of the pointers, as well as the geometrical properties (level, position, size, etc.), are grouped to be stored into octs rather than in cells. Consequently, the memory costs are significantly reduced (especially in 3D) in comparison to the simple FTT version without an oct structure.

The memory cost saving is undeniable when dealing with Euler equations, it is more debatable when dealing with complex models for multiphysics problems. In the latter, the ratio between physical flow variables and geometrical/AMR variables drastically increases. Khokhlov explains that the cost of his oct-FTT AMR version is 2 words of memory per cell instead of 17 words/cell for a non-oct version of FTT. These additional memory costs have to be compared with physical flow variables which must be stored (5 words/cell for Euler equations but 6xN words/cell for a general N-phase flow without extra physics or additional

variable storage). One could also note that high-order numerical methods induce other kinds of additional memory costs.

Another drawback of oct-trees is that the computational time associated with computing cell pointers from oct pointers and oct pointers from cell pointers, or in other words the necessary extra time to seek neighbor cells, is estimated at around 20% of the total computational time when computing single-phase flow with a first-order scheme while the computational time for the refinement/unrefinement procedure is only around 2% (accordingly to Khokhlov [20]). Here again, when complex models coupled with high-order numerical solvers are considered (meaning neighbor searching procedure for each flux computation of the high-order scheme, for each gradient computation, etc.), this extra computational time involved by the structure may no longer be negligible.

### *2.2. Basic idea of the new AMR data structure: The extra cell-face trees*

An efficient way to avoid increasing CPU time and difficulty in searching neighbors is to take advantage of information related to cell faces. In the finite volume framework, a face may be defined as a geometrical contour between two computational cells that is the seat of flux calculations. In programming language, it may also be defined as an object that stores two pointers, one for each neighboring cell on both side of the face. From the face point of view, they are interpreted as a "left" cell and a "right" cell. Availability of such objects, *i.e.* faces storing pointers to their neighboring cells, avoids the need to search for neighbors when solving inter-cell fluxes (*e.g.* Riemann problems) in a computational CFD code. In a more general context of unstructured grids, it also prevents from using a connectivity table. It implies that finite volume algorithms using such data structure may be easily used whatever the grid structure is.

Thus, in addition to the cell tree, we propose to define face trees. In these face trees, faces are represented by nodes that are linked to other face nodes by threads. The duality of cell tree and face trees represents a complex data structure that greatly simplifies the algorithm and reduces computational costs. Up to this remark, the new data structure is composed of:

- Cells that are organized in tree structure (oct tree or not). They may also be linked to faces.
- Faces that are also organized in tree structures. They can also be grouped in quad trees to mimic Khokhlov's oct-cell trees (a face will be split in up to 4 child faces in 3D).

Face trees in the AMR method imply additional memory costs. Nevertheless, with this new data structure, calculations at faces (fluxes, gradients, etc.) are naturally accessible without searching for neighbors. More than being interesting for Cartesian meshes, this point could be an interesting feature for application in the purpose of simulations on unstructured meshes. Moreover, the oct structure used in Khokhlov's work can be kept to group information regarding geometrical properties in the Cartesian framework.

*2.3. Detailed description of trees*

For the sake of clarity, data structure is presented for non-oct trees. The alert reader will easily extend the method to oct-trees if needed.

The main tree of the method is quite similar to the one of the classical FTT method. The computational cells constitute the nodes of the cell tree. In particular, they contain the physical flow properties (depending on the flow model under interest) as well as geometrical data. These cell nodes also include specific data for the AMR method:

- An integer for its level (0 if the cell is a root, $> 0$ otherwise),
- A pointer for each of its child cell nodes (up to 8 in 3D Cartesian mesh),
- An additional pointer for each of its faces (up to 6 in 3D Cartesian mesh),
- A pointer for the root of each new internal face tree (up to 12 in 3D Cartesian mesh). The particular case of the internal faces is presented in the following.

Compared to a classical FTT method, the novelty lies in the pointers towards the faces that represent an additional memory cost of maximum 11 pointers/cell in 3D (12 new pointers for internal faces, 1 less because the pointer to the parent cell is no longer needed in the method). Up to this point, a given cell can either be split or not (if its pointers to child cell nodes are null).

The novel second data structure is represented by new face trees. The interest of the presence of such face objects in a finite volume method lies in an improved access for flux computations between two computational cells, using the "left" and "right" cell pointers of the face. In this new AMR data structure, faces constitute nodes of new face trees. These face nodes then include additional data specific to the AMR method:

- An integer for its level,
- A pointer to each of its child face nodes (up to 4 in 3D).

This new data structure possesses some specificities. Indeed, let us consider the example of a 2D Cartesian cell represented in Figure 3. This cell is surrounded by 4 faces (blue edges). Refinement of this cell will give birth to 4 new computational cells and 12 new faces. Among these 12 faces, 8 of them (red dashed edges) are originated from the splitting of the parent faces and appear naturally as their children. Also, 4 new faces appear inside the parent cell as the result of the splitting of this cell and are considered as root of new face trees (green dash-dotted edges). Consequently, splitting of a given cell will act on face trees in two ways:

- It will increase the depth of already existing trees. "External" faces of the parent cell, that were leaves before splitting, will become parent of new faces (up to 4 in 3D), the last ones are thus leaves.
- In the same time, it will also generate new "internal" faces that are roots (and leaves) of new face trees.

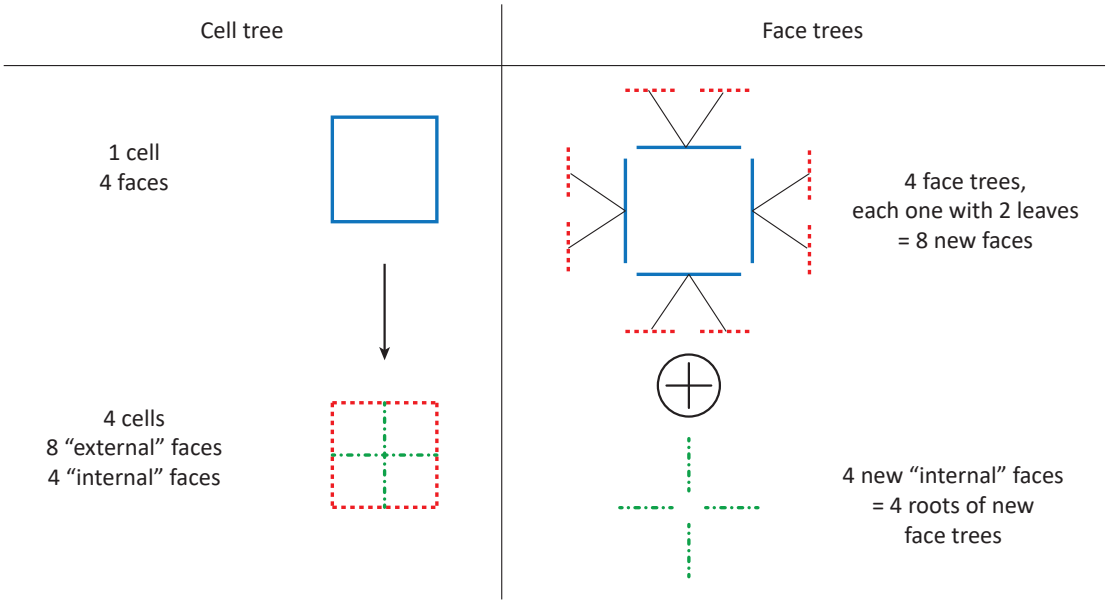


Figure 3: 2D example of the duality of the cell tree and the face trees.

A 2D example of the links between cell and face trees is illustrated in Figure 4. In this example, 3 levels are present. The figure is decomposed in three parts:

- The top part shows 2 successive refinements occurring from a given level-0 cell. The following cell and face trees correspond to this particular splitting.
- The middle part shows the corresponding cell tree represented by square nodes and composed with 4 level-1 cells and 4 level-2 cells.
- The bottom part shows the corresponding face trees represented by circle nodes. 4 level-0 face trees, 12 level-1 faces (including 4 new level-1 face trees) and 12 level-2 faces (including 4 new level-2 face trees) are generated.

The adopted numeration is ‘XYZ’ with X being C (for cell) or B (for face), Y corresponds to level number (here from 0 to 2) and Z is the letter corresponding to the entity (A to N in the present case). Pointers between cells in cell tree, as well as pointers between faces in face trees, are shown with black solid lines. These pointers are organizing the tree structure. The other pointers involved in the method, which are pointers between cells and faces, are non-exhaustively presented through examples in order to facilitate comprehension. These last pointers can be classified as follows:

- Pointers from cells to their external faces (2 in 1D, 4 in 2D and 6 in 3D). Such pointers are present for each cell and are needed for gradient computations as well as slope determination for second-order scheme. Examples of such pointers are shown with yellow solid lines, where cell C0A points to the 4 faces B0A, B0B, B0C and B0D of level 0.

- Pointers from cells to internal face trees. When a cell is refined, new faces appear between child cells (1 in 1D, 4 in 2D and 12 in 3D). These faces have the opportunity to become roots of new face trees which are linked to the parent cell through pointers. These pointers are needed for refinement/unrefinement purpose. Example of such pointers are shown with purple long-dashed lines, where cell C0A points to the 4 new faces B1K, B1L, B1M, and B1N.

- Pointers from faces to cells. Each face possesses two of these pointers which are used to compute hydrodynamic fluxes and update hydrodynamic part of the solution. These pointers are used only if the face is a leaf of a face tree. 3 examples corresponding to different situations are presented. In green dash-dotted lines are shown pointers from a face of level 1 (B1N) to two level-1 cells (C1C and C1D). In blue dotted lines, an example of a face (B2J) linked to two cells from different levels (C1C and C2D). The last example in red dashed lines shows a face (B1A) linked to a level-1 cell (C1A) and to another cell neighbor (level-0) of cell C0A or one of its child (level-1 cell) not shown in the figure.

One can note that the number of faces may be significant. Nevertheless, the face trees reasonably increase the memory involved since they only need a few additional pointers for each face. Comparison of the required memory with a classical FTT structure is presented in Table 1 in which the numbers of words per cell and per face are detailed. The total number of words reported to a cell is also given. For a given 3D hexahedron cell, the new method requires 11 additional words. Noticing that a face is common to 2 neighboring cells, a cell requires approximately 3 faces (instead of 6), each of them requiring 5 words.

| | Data type | Classical FTT AMR | New AMR |
|---|---|---|---|
| Number of words per cell | Cell level | 1 | 1 |
| | Refinement indicator | 1 | 1 |
| | Pointer to parent cell | 1 | - |
| | Pointers to child cells | 8 | 8 |
| | Pointers to neighboring cells | 6 | - |
| | Pointers to faces | - | 6 |
| | Pointers to internal faces | - | 12 |
| Number of words per face | Face level | - | 1 |
| | Pointers to child faces | - | 4 |
| | Total number of words per cell | 17 | 43 |

Table 1: Memory cost comparison between classical-FTT-AMR method and new AMR method using face trees. For a given 3D hexahedron cell, the new method requires 11 additional words. Noticing that a face is common to 2 neighboring cells, a cell requires approximately 3 faces (instead of 6), each of them requiring 5 words. The global overhead of the new method is thus 26 words/cell.

Saving may be done regarding memory costs by using Khokhlov-like oct-tree method for particular Cartesian grids. Indeed, it is possible to group faces, as it was done for cells using oct. "External" faces can be grouped in quad (group of 4 faces) and "internal" faces can

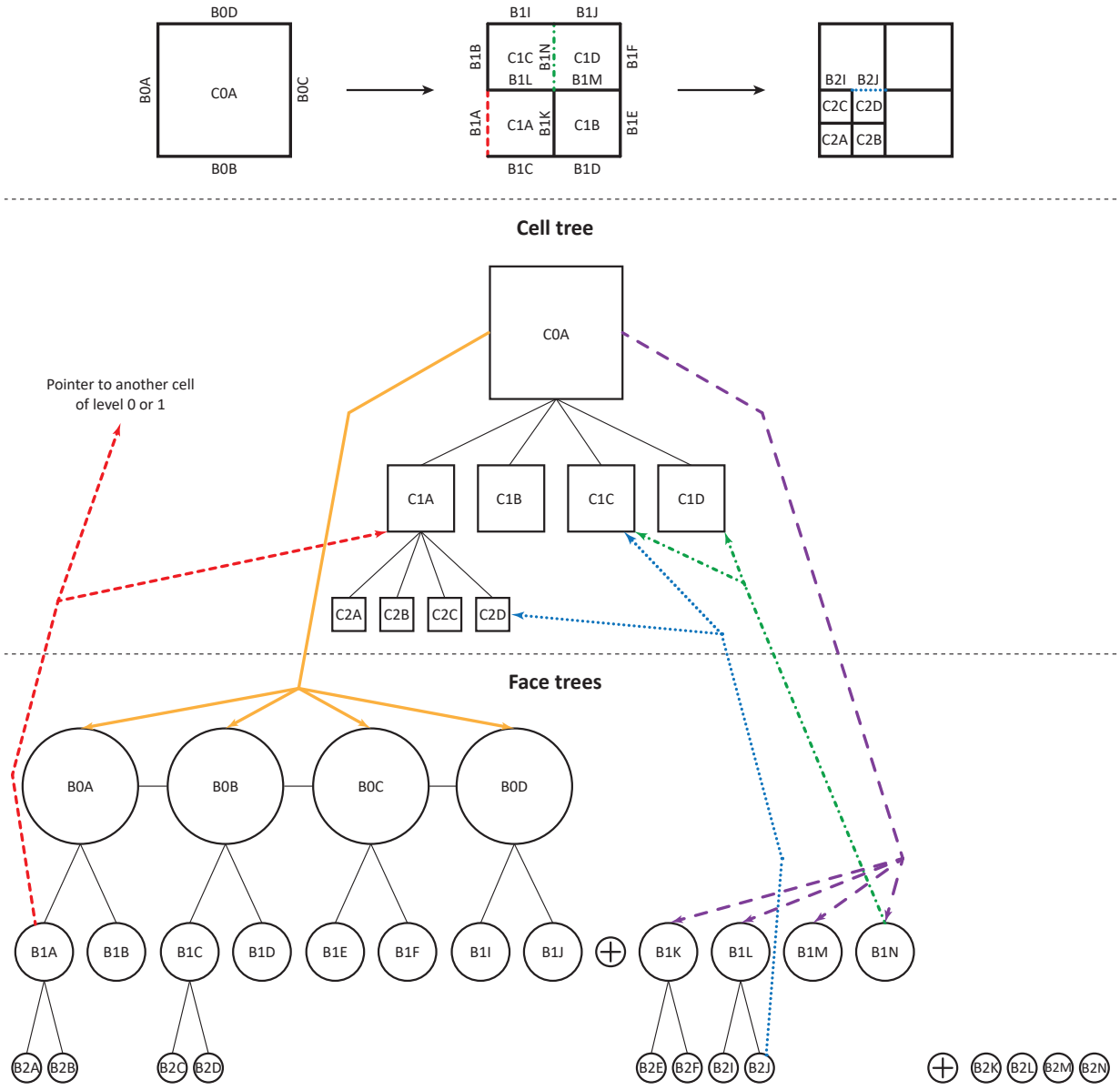


Figure 4: 2D example of links between cell and face trees. Here, some details are given for a cell and its 2 levels of refinement. The top represents the cells appearing from the two successive refinements from a level-0 parent cell. The middle and bottom sketches are representing the cell tree and face trees, respectively. Connections between both cell and face trees are presented for some typical situations.

be grouped in dodeca (group of 12 faces). This improvement in term of memory cost is possible but complicates the AMR algorithm. The global overhead of the new method may thus reduce to 3.75 words/cell.

As mentioned in Section 2.1, one should note that the global overhead of the new method has to be compared to the memory cost for storage of geometrical, physical and high-order-scheme variables, as well as extra quantities stored for computational conveniences, that may

represent the larger part of memory costs in multiphase, multiphysics and high-accuracy computations. For example, when considering the single-phase Euler equations on a Cartesian grid at first order, 3 extra words are needed at least to store flow variables. The ratio between FTT and the present dual-tree method, in the worst configuration, is approximately equal to 0.43. When considering the complex multiphase flow model presented in Section 4.3, solved with a second-order scheme, the ratio can easily increase to 0.6 or even more if more than 2 phases are considered.

In the following, the oct-tree structure is not combined with this new AMR method to totally eliminate the neighbor-searching operations. The algorithm simplicity and the computational efficiency is thus highlighted.

## 3. General AMR algorithm

### *3.1. Finite volume scheme for conservation laws*

We consider a system of conservation laws in the following form:

$$\frac{\partial \mathbf{U}}{\partial t} + \boldsymbol{\nabla} \cdot \overline{\overline{F}}\left(\mathbf{U}\right) = \mathbf{S}, \tag{1}$$

with $\mathbf{U}$ the conservative-variable vector, $\overline{\overline{F}}$ the flux tensor and $\mathbf{S}$ the source-term vector. The finite volume scheme for time evolution of System (1) discretized on a computational cell $i$ of volume $V_i$ delimited by surfaces $A_s$ of normal unit vector $\mathbf{n}_s$ is classical:

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\Delta t}{V_i} \sum_{s=1}^{N} A_s \overline{\overline{F}}_s^{\star} \cdot \mathbf{n}_s + \Delta t \mathbf{S}_i, \tag{2}$$

where $\overline{\overline{F}}_s^{\star}$ represents the flux-tensor solution of the Riemann problem between states on both sides of cell surface $A_s$. First-order numerical scheme (2) is restricted by a CFL condition on $\Delta t$.

### *3.2. Time-stepping strategy*

The efficiency of an AMR method requires the implementation of a specific time-stepping strategy. Following the work of Khokhlov [20], the time-stepping strategy is based on two key points:

- Cells at different levels evolve with different time steps according to their level of refinement. In order to maintain the global time-step coherence for unsteady simulations, if cells of level $l$ evolve at a given time step, cells of level $l+1$ will then evolve 2 times with a time step 2 times smaller. It thus avoids to compute the smallest time step of the entire computational domain, necessary for stability, for every cell but only for the ones where it is necessary, *i.e.*, the cells at the highest level. Then, it results in a saving of CPU time.

- This time-stepping strategy allows interleaving between time integration and tree refinement. It results in a saving of memory as it limits excessive buffer layer of refinement ahead a discontinuity [20].

The global time step is determined using the minimum tree level where there are leaf cells ($l_{\min}$) and the CFL condition:

$$\Delta t = \Delta t\left(l_{\min}\right) = cfl \frac{L}{2^{l_{\min}} \max\left(\left|\left(u+a\right)_s^*\right|\right)} \ ,$$

where $cfl < 1$ is a constant, $L$ is the characteristic length of the coarser cells (at level $l = 0$) and the maximum speed is determined going through each leaf face where $u$ is the fluid velocity in the corresponding face direction and $a$ is the speed of sound waves. Time steps at various levels are:

$$\Delta t\left(l\right) = 2^{l_{\min}-l} \Delta t.$$

The general integration procedure occurs at the different levels of the tree as an interleaving of advancing and refinement procedures. It is expressed as a recursive procedure $I(l_{\min})$ with:

$$\begin{array}{lcll} I\left(l_{\min}\right) & = & A\left(l_{\min}\right) I\left(l_{\min}+1\right) R\left(l_{\min}\right), & \\ I\left(l\right) & = & A\left(l\right) I\left(l+1\right) A\left(l\right) I\left(l+1\right) R\left(l\right) & \text{for} \quad l \neq \left(l_{\min}, l_{\max}\right), \\ I\left(l_{\max}\right) & = & A\left(l_{\max}\right) A\left(l_{\max}\right) R\left(l_{\max}\right), & \end{array} \tag{3}$$

where $A(l)$ represents the advancing procedure of level $l$ described in Section 3.3 and $R(l)$ is the refinement/unrefinement procedure of level $l$ detailed in Section 3.4. All procedures in (3) are performed from right to left, *i.e.*, $R\left(l\right)$ first, and $A\left(l\right)$ last. An example of the sequence generated by (3) could be, for $l_{\min} = 0$ and $l_{\max} = 2$:

$$[R\left(0\right) \quad [R\left(1\right) \quad [R\left(2\right) A\left(2\right) A\left(2\right)] \quad A\left(1\right) \quad [R\left(2\right) A\left(2\right) A\left(2\right)] \quad A\left(1\right)] \quad A\left(0\right)].$$

One can note that the generality of the new method in the present finite volume framework simplifies the recursive integration procedure in comparison to [20] where directional time-step splitting is computed. Indeed, the procedure of the lower level $l_{\min}$ is simplified, *i.e.*, only one advancing procedure is performed, and each advancing procedure is identical (only the referred level changes). The last one has to be compared with [20] where two advancing procedures have to be carried out, one for the sequence of XYZ one-dimensional sweeps and one for its reversed.

Because of the recursive evolution algorithm, browsing of trees will lead to a significant amount of tests to detect cell and face levels. A possible option to avoid going through all the trees and then to accelerate the procedures is to add lists for cells and faces and for each level of the simulation. Then, the loops over the cells or the faces become straightforward and constantly efficient.

*3.3. Advancing procedure*

The advancing procedure $A$ is called at each time step ($\Delta t\,(l)$) to advance the solution at the time $t + \Delta t\,(l)$ using the numerical scheme (2). This advancing procedure is decomposed in 3 steps:

- The first step is solving the hyperbolic part of System 2. A loop is first performed on leaf faces of level $l$ where fluxes are estimated (using Riemann solvers) and stacked in a flux buffer variable (initially set to 0) in each of "left" ($L$ subscript) and "right" ($R$ subscript) neighboring cells. These flux buffers are denoted by $\widetilde{\mathbf{F}}$. It is important to notice that possible contribution to these buffers comes from higher face level (during preceding advancing procedures at higher level).
  At the end of this face loop, buffers for cells of level $l$ are complete. Indeed, whatever the neighbor levels are, fluxes have been stacked either during this advancing procedure or during those of higher level. Then, conservative variables for leaf cells of level $l$ should be evolved and corresponding flux buffers reset to 0 for next time step.

- The second step consists in upgrading leaf cells of level $l$ through the source-term integration.

- The third step is an averaging procedure consisting in updating split cells of level $l$. This step is useful for the correct computation of the refinement procedure (presented in details in Section 3.4).

The $A\,(l)$ procedure is described in a form of the following pseudocode:

1. — Hyperbolic computation —

   for (leaf faces $f$ of level $l$) {
   Compute the hyperbolic flux tensor $\overline{\overline{F}}^{\star}_f = \overline{\overline{F}}^{\star}_f\,(\mathbf{U}^n_L, \mathbf{U}^n_R)$;
   $\widetilde{\mathbf{F}}_L = \widetilde{\mathbf{F}}_L - l_{\text{diff},L} L_f \overline{\overline{F}}^{\star}_f \cdot \mathbf{n}_f$;
   $\widetilde{\mathbf{F}}_R = \widetilde{\mathbf{F}}_R + l_{\text{diff},R} L_f \overline{\overline{F}}^{\star}_f \cdot \mathbf{n}_f$;
   }
   for (leaf cells $i$ of level $l$) {
   $\mathbf{U}^1_i = \mathbf{U}^n_i + \frac{\Delta t}{V_i} \widetilde{\mathbf{F}}_i$;
   $\widetilde{\mathbf{F}}_i = \mathbf{0}$;
   }

2. — Source-term computation —

   for (leaf cells $i$ of level $l$) {
   $\mathbf{U}^{n+1}_i = \mathbf{U}^1_i + \Delta t \mathbf{S}_i\,(\mathbf{U}^1_i)$;
   }

3. — Averaging of the child cells for each parent cells —

for (parent cells $i$ of level $l$) {
    for (child cells $j$ of parent cell $i$) {
        $\widetilde{\mathbf{F}}_i = \widetilde{\mathbf{F}}_i + \mathbf{U}_j^{n+1}$;
    }
    $\mathbf{U}_i^{n+1} = \widetilde{\mathbf{F}}_i /$ Number of child cells;
    $\widetilde{\mathbf{F}}_i = \mathbf{0}$;
}

The $l_{\text{diff}}$ factor appearing in the stacking of the flux buffers takes into account potential level differences between "left" and "right" cells on the considered face. It would take the value $l_{\text{diff}} = 1$ if the two neighboring cells have the same level or if the flux is applied to the cell with the higher level, and the value $l_{\text{diff}} = 0.5$ if the flux is applied to the cell with the lower level. In the example of cells C2D and C1B in Figure 4, the one with the higher level (C2D) will have 2 time-step integrations while the one with the lower level (C1B) will have just 1. In that case, for the flux computation between these two cells, $l_{\text{diff},L} = 1$ for cell C2D and $l_{\text{diff},L} = 0.5$ for cell C1B. In that way, it makes a time-averaged flux in the cell with the lower level.

Extension of this advancing procedure will be done in Section 4.3 for non-conservative system of multiphase flows.

### *3.4. Mesh-refinement procedure*

If the data structure and integration algorithm represent key points to ensure efficiency of an AMR simulation, the ability to refine or unrefine at required locations is another key point that is obviously linked to the quality of numerical results. This is also the most problem-dependent part and the choice of the refinement criteria is undeniably the most difficult point for the user of an AMR method. This point will be discussed on a practical test case in Section 5. The cell refinement is always linked to a refinement indicator $0 \leqslant \xi \leqslant 1$ which is computed and stored for every computational cell. This indicator will be used to detect which cells need to be refined or unrefined:

- If a leaf cell has $\xi \geqslant \xi_{\text{split}}$, it indicates that the corresponding cell must be refined,
- If a split cell has $\xi < \xi_{\text{join}}$, the corresponding cell can be unrefined,

where $\xi_{\text{split}}$ and $\xi_{\text{join}}$ are two predefined constant parameters controlling the cell refinement dynamics forward and backward a discontinuity. An extra condition to control refinement is also imposed: The difference of levels between two neighboring cells cannot be larger than 1.

#### *3.4.1. Setup of $\xi$ indicator*

The approach we use to calculate the refinement $\xi$ indicator is based on locations of significant gradients [1, 8, 25] and it proceeds in two steps:

- For each computational cell, $\xi$ is calculated by:

$$\begin{aligned} \xi &= 1 \quad \text{if:} \quad \frac{|(X)_{Nb(i,j)} - (X)_i|}{\min\left((X)_{Nb(i,j)}, (X)_i\right)} > \epsilon, \\ \xi &= 0 \quad \text{otherwise,} \end{aligned} \tag{4}$$

  where $X$ can be any pertinent physical variable (for example pressure $p$, velocity magnitude $\|\mathbf{u}\|$, density $\rho$ or volume fraction $\alpha$). $Nb(i,j)$ represents neighboring cells ($j$ accounts for a corresponding neighbor of cell $i$). The choice of the variable will discriminate shocks, contact discontinuities, interfaces or any kind of gradients. $\epsilon$ is a constant parameter that controls the limit in term of stiffness of the detected gradients. Attention should be paid to the velocity magnitude to avoid division by zero. A combination of several gradients can also be used to improve detection.

- The second step consists in smoothing the refinement indicator. This operation is very important for several reasons. First, it prevents cells from being falsely refined. Secondly, smoothing allows cells forward a discontinuity to be refined before the discontinuity arrival and by this way prevents oscillations as well as a loss of precision. To perform smoothing, we assume $\xi$ can be modeled using a diffusion equation:

$$\frac{\delta \xi}{\delta \tilde{t}} = K \nabla^2 \xi, \tag{5}$$

  where $\tilde{t}$ is a fictive diffusion time only used to advance the solution for the diffusion of $\xi$ into the domain. So, this diffusion has no link or impact on the treated physical characteristic. $K = 2^{-2l} L^2$ is a constant diffusion coefficient. This equation is solved with an explicit time advancement where the time step is chosen to preserve the diffusion stability ($dt = cfl_{\text{diff}} K/2$, where $cfl_{\text{diff}}$ corresponds to the CFL condition of the diffusion equation). Note that when this equation is solved, the number of fictive time iterations indirectly gives the number of cells where the indicator will be diffused. Typically, 3 or 4 time iterations are enough.

The splitting and joining criteria are then used to determine if the cell has to be refined or unrefined. The refinement around a density discontinuity is presented in Figure 5 as a typical example.

*3.4.2. Refinement and unrefinement of cells and faces*

Due to the dual data structure, refinement (unrefinement) proceeds in two steps: First the refinement (unrefinement) of the cells and second of the faces.

The refinement of a cell occurs if $\xi \geqslant \xi_{\text{split}}$ once the $\xi$ indicator of every cell of the current level $l$ is smoothed. The two steps are:

- First, the cell refinement: It does not involve special difficulties as it follows the scheme in Figure 2. A refined cell will give birth to up to 8 child cells (in Cartesian 3D) of level $l+1$ and each child will be built with the same physical characteristics than its parent cell.

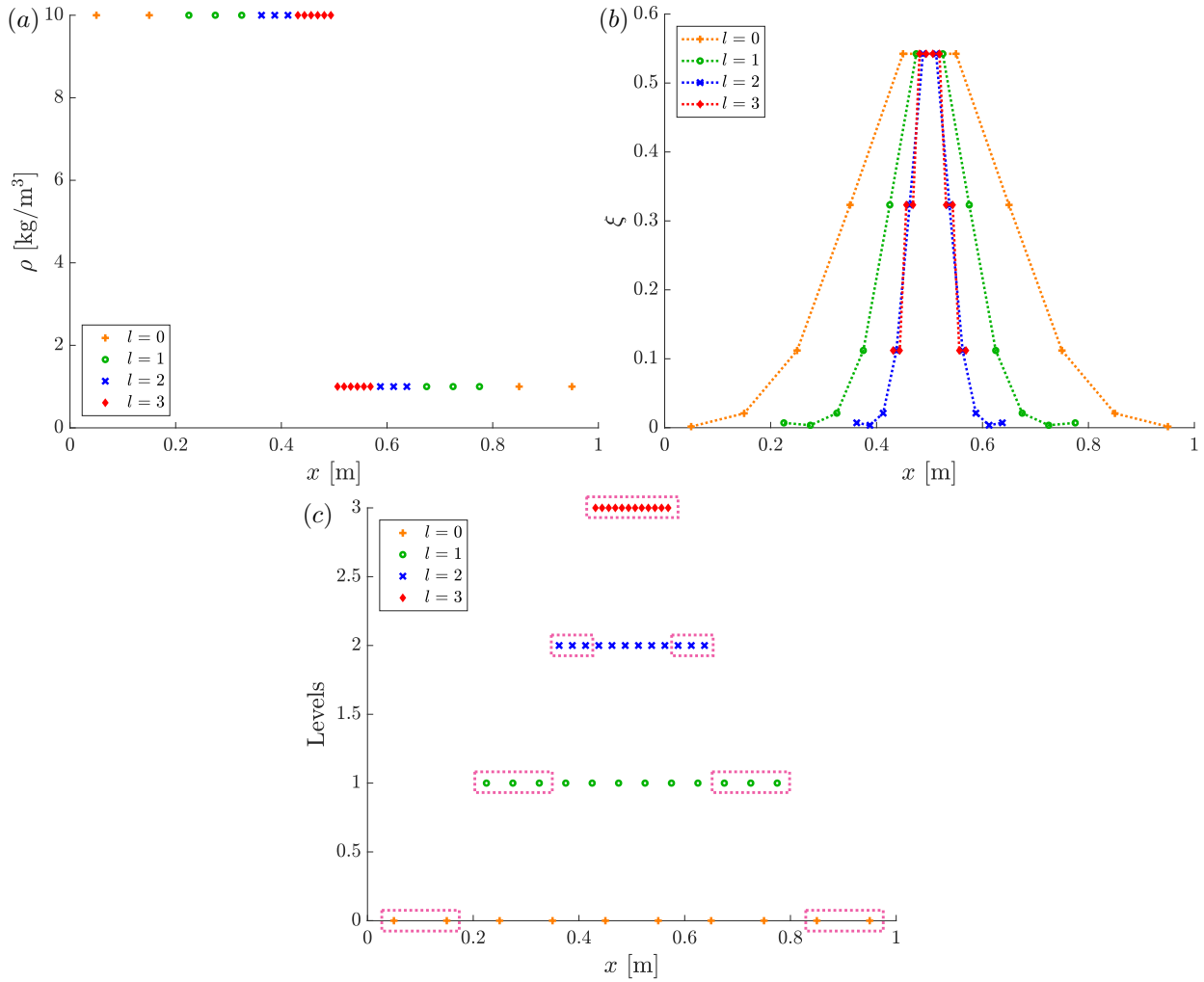


Figure 5: Result of the successive mesh refinement procedures around a density discontinuity. (a) The density discontinuity, (b) the values of the indicator for each level and (c) the distribution of cell levels (the pink-surrounded leaf cells are the cells where the integration occurs) are presented.

- Second, the face refinement: It is also performed in two steps for each split cell. The first one is the creation of the internal child faces of level $l+1$ that belong to the parent cell (it then creates up to 12 new face trees in Cartesian 3D). In the second step, the creation of the external child faces of level $l+1$ for each of the faces of the parent cell (level $l$) is achieved only if it was not already done by the corresponding neighboring cell. The child faces belong to their parent face (see Figure 3).

The unrefinement occurs if $\xi < \xi_{\text{join}}$ and is performed according the two steps:

- First, the cell unrefinement: The physical variables of the child cells are averaged and overwrite the ones of the parent cell (see point 3 of the pseudocode of Section 3.3). Then, child cells are removed.

- Second, the face unrefinement: It is performed in two steps. The first one is the removal of the internal child faces that belong to the parent cell. In the second step, the removal of the external child faces is done only if the corresponding neighboring cell is not split.

All the pointers are obviously redirected to the corresponding cell or face if necessary.

*3.4.3. Pseudocode of the mesh-refinement procedure*

The $R\,(l)$ refinement procedure is thus described as the following pseudocode:

1. — $\xi$ setup —

   for (cells $i$ of level $l$) {
   $\xi = 0$;
   if (one of the gradient criteria is respected) { $\xi = 1$; }
   }

2. — Smoothing of $\xi$ —

   for (x diffusion iterations) {
   for (cells $i$ of level $l$) {
   Compute the diffusion equation for $\xi$ (Eq. (5));
   }
   }

3. — Refinement —

   if ($l < l_{\max}$) {
   for (non-split cells $i$ of level $l$) {
   if ($\xi \geqslant \xi_{\text{split}}$ & level of each neighboring cell $> l - 1$) { Cell $i$ is refined; }
   }
   }

4. — Unrefinement —

   if ($l < l_{\max}$) {
   for (split cells $i$ of level $l$) {
   if ($\xi < \xi_{\text{join}}$ & level of each neighboring cell $\leqslant l + 1$ & child cells are not split) {
   Cell $i$ is unrefined with children averaging (see point 3 of the pseudocode of Section 3.3) to overwrite the values of cell $i$;
   }
   }
   }

If the general AMR algorithm presented in this section is applied to compute the solution of a simple conservative model (*i.e.* Euler equations) on a single core, one can expect only little computational-time saving in comparison to Khokhlov's algorithm [20], because of the few neighbor-searching operations in the AMR procedure. On the other hand, if the

mathematical model considers multiphase compressible flows, embedding complex additional physical effects (surface tension, viscosity, heat transfers, etc.) in a parallel architecture, the need to search for neighboring cells to compute inter-cell fluxes, gradients and slopes for second-order scheme are significantly increased. The computational time spent to access this information will necessarily explode and the advantages of this new AMR method will naturally appear. This is why the next section is devoted to the extension of the general AMR algorithm to the multiphase flow model of Schmidmayer et al. [38] for diffuse-interface problems including surface tension.

## 4. Extension to multiphase flow model of Schmidmayer et al. [38]

The AMR method presented in this paper is intended for diffuse-interface models for multiphase compressible flow. The retained model to illustrate the use of the AMR method is detailed in Schmidmayer et al. [38] in which surface-tension effects are taken into account. Though, the AMR method can be easily adapted to treat extra physics as for example phase transition [24, 35], cavitation [29], detonation in high-energetic materials [30], solid-fluid interaction and compaction of granular media [11, 12], and low Mach number flows [27].

We recall here the main properties of the Schmidmayer et al. model [38].

### *4.1. Multiphase system of equations*

The pressure-relaxation model with surface tension of [38] is:

$$\begin{cases} \frac{\partial \alpha_1}{\partial t} + \mathbf{u}\cdot\nabla\alpha_1 & = \mu\left(p_1 - p_2\right), \\ \frac{\partial \alpha_1\rho_1}{\partial t} + \boldsymbol{\nabla}\cdot\left(\alpha_1\rho_1\mathbf{u}\right) & = 0, \\ \frac{\partial \alpha_2\rho_2}{\partial t} + \boldsymbol{\nabla}\cdot\left(\alpha_2\rho_2\mathbf{u}\right) & = 0, \\ \frac{\partial \rho\mathbf{u}}{\partial t} + \boldsymbol{\nabla}\cdot\left(\rho\mathbf{u}\otimes\mathbf{u} + p\overline{\overline{I}} + \overline{\overline{\Omega}}\right) & = \mathbf{0}, \\ \frac{\partial \alpha_1\rho_1 e_1}{\partial t} + \boldsymbol{\nabla}\cdot\left(\alpha_1\rho_1 e_1\mathbf{u}\right) + \alpha_1 p_1\boldsymbol{\nabla}\cdot\mathbf{u} & = -\mu p_I\left(p_1 - p_2\right), \\ \frac{\partial \alpha_2\rho_2 e_2}{\partial t} + \boldsymbol{\nabla}\cdot\left(\alpha_2\rho_2 e_2\mathbf{u}\right) + \alpha_2 p_2\boldsymbol{\nabla}\cdot\mathbf{u} & = \mu p_I\left(p_1 - p_2\right), \\ \frac{\partial c}{\partial t} + \mathbf{u}\cdot\boldsymbol{\nabla}c & = 0, \end{cases} \tag{6}$$

where $\alpha_k$, $\rho_k$, $e_k$ and $p_k$ are the volume fraction, the density, the internal energy and the pressure of phase $k$, and each fluid is governed by its own equation of state (EOS) $e_k = e_k\left(\rho_k, p_k\right)$. $\rho$, $p$ and $\mathbf{u}$ are the mixture variables for density, pressure and velocity. Concerning the surface-tension terms, $\sigma$ is the surface-tension coefficient, $c$ is a color function and $\overline{\overline{\Omega}}$ is the capillary tensor given by:

$$\overline{\overline{\Omega}} = -\sigma\left(\|\boldsymbol{\nabla}c\|\overline{\overline{I}} - \frac{\boldsymbol{\nabla}c\otimes\boldsymbol{\nabla}c}{\|\boldsymbol{\nabla}c\|}\right).$$

$\mu$ is the pressure relaxation coefficient, $p_I = \frac{z_2 p_1 + z_1 p_2}{z_1 + z_2}$ (see [33] for details) and $z_k = \rho_k a_k$ is the acoustic impedance of the phase $k$ with $a_k$ being the speed of sound of the corresponding phase. The mixture pressure is given by:

$$p = \alpha_1 p_1 + \alpha_2 p_2.$$

Due to the condition $p_1 \neq p_2$ in this model, the total-energy equation of the mixture is replaced by the internal-energy equation for each phase. Nevertheless, the mixture-total-energy equation of the system can be written in usual form:

$$\frac{\partial \rho E + \varepsilon_\sigma}{\partial t} + \boldsymbol{\nabla} \cdot \Big( (\rho E + \varepsilon_\sigma + p)\, \mathbf{u} + \overline{\overline{\Omega}} \cdot \mathbf{u} \Big) = 0, \tag{7}$$

where $E = e + \frac{1}{2}\|\mathbf{u}\|^2$ and $e$ are the mixture variables for total energy and internal energy. And the capillary energy is equal to $\varepsilon_\sigma = \sigma \|\boldsymbol{\nabla} c\|$.

The equation (7) is redundant when both phase internal-energy equations are solved, but it will appear to be an important ingredient for numerical method to ensure the energy conservation and to preserve a correct treatment of shock waves.

*4.2. Numerical Method without AMR*

Model (6) without the relaxation terms is split into two submodels. The first submodel does not take into account the surface-tension terms which are included in the second one. The numerical method is then presented as a 3-step method fully detailed in [38]. Each step is successively performed in order to circumvent specific numerical problems.

- First, the hyperbolic non-equilibrium-pressure model — Model (6) without the surface-tension and relaxation terms — is solved using a Godunov-type method. This system describes the transport and the compression waves.
- Second, the hyperbolic surface-tension (capillary) model — Model (6) with the surface-tension terms — is solved. A specific attention is paid to the choice of the flux terms in order to ensure the momentum and energy conservation.
- Third, a relaxation procedure leads to the pressure equilibrium.

The unknown vector $\mathbf{U}^{n+1}$ is obtained from the initial condition $\mathbf{U}^n$ by application of the three successive operators according to the sequence:

$$\mathbf{U}^{n+1} = L_{\text{relax}} L_{\text{cap}} L_{\text{hyper}} \left( \mathbf{U}^n \right),$$

where the vector $\mathbf{U}$ contains the unknown quantities defined in the system:

$$\mathbf{U} = [\alpha_1, \alpha_1\rho_1, \alpha_2\rho_2, \rho u, \rho v, \rho w, \alpha_1\rho_1 e_1, \alpha_2\rho_2 e_2, c, \rho E + \varepsilon_\sigma]^T$$

This chain of operators must remain the basis of the numerical solution but required some modification due to the AMR procedure.

*4.3. Extension of the AMR algorithm for multiphase flow*

The AMR algorithm presented in Section 3 is now extended to treat multiphase flow model (6) that implies several modifications:

1. Model (6) is non-conservative. It is therefore necessary to take non-conservative terms into account in the advancing procedure,

2. The global time-step of integration is slightly modified to add a cell-gradient procedure $G$ which computes the color-function gradients required in the surface-tension-effect formulation. It is done before going to the higher tree level and the integration procedure now reads:

$$\begin{array}{lcll} I\left(l_{\min}\right) & = & A\left(l_{\min}\right) I\left(l_{\min}+1\right) G\left(l_{\min}\right) R\left(l_{\min}\right), & \\ I\left(l\right) & = & A\left(l\right) I\left(l+1\right) A\left(l\right) I\left(l+1\right) G\left(l\right) R\left(l\right) & \text{for} \quad l \neq \left(l_{\min}, l_{\max}\right), \\ I\left(l_{\max}\right) & = & A\left(l_{\max}\right) A\left(l_{\max}\right) R\left(l_{\max}\right). & \end{array} \tag{8}$$

Thus, the sequence generated by (8) for the same precedent example ($l_{\min} = 0$ and $l_{\max} = 2$) now gives:

$$[R\left(0\right) G\left(0\right) \quad [R\left(1\right) G\left(1\right) \quad [R\left(2\right) A\left(2\right) A\left(2\right)]$$

$$A\left(1\right) \quad [R\left(2\right) A\left(2\right) A\left(2\right)] \quad A\left(1\right)] \quad A\left(0\right)].$$

3. Model (6) also contains relaxation terms that implies modification in the algorithm.

In the following part, the modification of the advancing procedure as well as the cell-gradient and relaxation procedures specific to multiphase compressible flows with surface tension are detailed.

*4.3.1. Cell-gradient procedure*

The different cell gradients which could be required to treat a specific physic, *e.g.* the surface tension, are compute *via* the cell-gradient procedure $G$. When the computation of the surface-tension effects is done, the flux computation on a face uses the color-function cell gradients of each neighboring cell of this face. To avoid unnecessary computations, the $G$ procedure is proceeded in a loop going through each cell, and not when the computation of the fluxes are involved. Indeed, in the case where one of the two neighboring cells of a face has a smaller level than this face, the cell gradient in this cell has to be computed before the flux computation. But, in the recursive integration procedure (Equation (8)), if the cell-gradient procedure $G\left(l_{\min}\right)$ is not done before going to the integration procedure of the higher level $I\left(l+1\right)$, the needed cell gradient for the flux computation of the advancing procedure of this higher level $A\left(l+1\right)$ would not have been computed. So, the procedure is done at the level $l$ before going to the recursive integration procedure $I\left(l+1\right)$ and it follows the pseudocode:

```
if (l < l_max) {
  for (leaf cells i of level l) {
      Compute the color-function gradients G_i(U_i^n);
  }
}
```

*4.3.2. Advancing procedure*

For the solution of Model (6), the advancing procedure has to take into account three additional points: One is related to the additional physics, another for the relaxation step and a last one for non-conservative terms. Source terms are not considered in the model. If there were some, they would have been added in the following pseudocode between the point 2 and 3, and under the same formulation than in the pseudocode of Section 3.3. The pseudocode of the $A\,(l)$ procedure is:

1. — Hyperbolic computation —

   for (leaf faces $f$ of level $l$) {
   Compute the hyperbolic flux tensor $\overline{\overline{F}}^{\star}_{f} = \overline{\overline{F}}^{\star}_{f}\,(\mathbf{U}^{n}_{L}, \mathbf{U}^{n}_{R})$ and its corresponding contact-discontinuity velocity $\mathbf{u}^{\star}_{f}$;
   $\widetilde{\mathbf{F}}_{L} = \widetilde{\mathbf{F}}_{L} - l_{\text{diff},L} L_{f} \left( \overline{\overline{F}}^{\star}_{f} + \mathbf{H}_{\text{nc},f}\,(\mathbf{U}^{n}_{L})\,\mathbf{u}^{\star}_{f} \right) \cdot \mathbf{n}_{f}$;
   $\widetilde{\mathbf{F}}_{R} = \widetilde{\mathbf{F}}_{R} + l_{\text{diff},R} L_{f} \left( \overline{\overline{F}}^{\star}_{f} + \mathbf{H}_{\text{nc},f}\,(\mathbf{U}^{n}_{R})\,\mathbf{u}^{\star}_{f} \right) \cdot \mathbf{n}_{f}$;
   }
   for (leaf cells $i$ of level $l$) {
   $\mathbf{U}^{1}_{i} = \mathbf{U}^{n}_{i} + \frac{\Delta t}{V_{i}} \widetilde{\mathbf{F}}_{i}$;
   $\widetilde{\mathbf{F}}_{i} = \mathbf{0}$;
   }

2. — Surface-tension computation —

   for (leaf cells $i$ of level $l$) {
   Compute the color-function gradients $\mathbf{G}_{i} = \mathbf{G}_{i}\,(\mathbf{U}^{1}_{i})$;
   }
   for (leaf faces $f$ of level $l$) {
   Compute the capillary flux tensor $\overline{\overline{F}}^{\text{cap}}_{f}\,(\mathbf{U}^{1}_{L}, \mathbf{U}^{1}_{R}, \mathbf{G}_{L}, \mathbf{G}_{R})$;
   $\widetilde{\mathbf{F}}_{L} = \widetilde{\mathbf{F}}_{L} - l_{\text{diff},L} L_{f} \overline{\overline{F}}^{\text{cap}}_{f} \cdot \mathbf{n}_{f}$;
   $\widetilde{\mathbf{F}}_{R} = \widetilde{\mathbf{F}}_{R} + l_{\text{diff},R} L_{f} \overline{\overline{F}}^{\text{cap}}_{f} \cdot \mathbf{n}_{f}$;
   }

   for (leaf cells $i$ of level $l$) {
   $\mathbf{U}^{2}_{i} = \mathbf{U}^{1}_{i} + \frac{\Delta t}{V_{i}} \widetilde{\mathbf{F}}_{i}$;
   $\widetilde{\mathbf{F}}_{i} = \mathbf{0}$;
   }

3. — Relaxation computation —

   for (leaf cells $i$ of level $l$) {
   Compute the relaxation procedure to obtain $\mathbf{U}^{n+1}_{i}$ from $\mathbf{U}^{2}_{i}$;
   }

4. — Averaging of the child cells for each parent cells —

for (parent cells $i$ of level $l$) {
  for (child cells $j$ of parent cell $i$) {
    $\widetilde{\mathbf{F}}_i = \widetilde{\mathbf{F}}_i + \mathbf{U}_j^{n+1}$;
  }
  $\widetilde{\mathbf{F}}_i = \widetilde{\mathbf{F}}_i$/ Number of child cells;
  Compute the relaxation procedure to obtain $\mathbf{U}_i^{n+1}$ from $\widetilde{\mathbf{F}}_i$;
  $\widetilde{\mathbf{F}}_i = \mathbf{0}$;
}

where the hyperbolic flux tensor $\overline{\overline{F}}^{\star} = \left(\mathbf{F}_x^{\star}, \mathbf{F}_y^{\star}, \mathbf{F}_z^{\star}\right)$, the non-conservative vector $\mathbf{H}_{\text{nc}}$ and the capillary flux tensor $\overline{\overline{F}}^{\text{cap}} = \left(\mathbf{F}_x^{\text{cap}}, \mathbf{F}_y^{\text{cap}}, \mathbf{F}_z^{\text{cap}}\right)$ are given in a Cartesian expression by:

$$\mathbf{F}_x^{\star}(\mathbf{U}) = \begin{bmatrix} \alpha_1 u \\ \alpha_1\rho_1 u \\ \alpha_2\rho_2 u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \alpha_1\rho_1 e_1 u \\ \alpha_2\rho_2 e_2 u \\ cu \\ (\rho E + p)\, u \end{bmatrix} \qquad \mathbf{F}_y^{\star}(\mathbf{U}) = \begin{bmatrix} \alpha_1 v \\ \alpha_1\rho_1 v \\ \alpha_2\rho_2 v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \alpha_1\rho_1 e_1 v \\ \alpha_2\rho_2 e_2 v \\ cv \\ (\rho E + p)\, v \end{bmatrix} \qquad \mathbf{F}_z^{\star}(\mathbf{U}) = \begin{bmatrix} \alpha_1 w \\ \alpha_1\rho_1 w \\ \alpha_2\rho_2 w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \alpha_1\rho_1 e_1 w \\ \alpha_2\rho_2 e_2 w \\ cw \\ (\rho E + p)\, w \end{bmatrix}$$

$$\mathbf{F}_x^{\text{cap}}(\mathbf{U}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Omega_{11} \\ \Omega_{12} \\ \Omega_{13} \\ 0 \\ 0 \\ 0 \\ \varepsilon_\sigma u + \Omega_{11} u + \Omega_{12} v + \Omega_{13} w \end{bmatrix} \qquad \mathbf{F}_y^{\text{cap}}(\mathbf{U}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Omega_{21} \\ \Omega_{22} \\ \Omega_{23} \\ 0 \\ 0 \\ 0 \\ \varepsilon_\sigma u + \Omega_{21} u + \Omega_{22} v + \Omega_{23} w \end{bmatrix}$$

$$\mathbf{F}_z^{\text{cap}}(\mathbf{U}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Omega_{31} \\ \Omega_{32} \\ \Omega_{33} \\ 0 \\ 0 \\ 0 \\ \varepsilon_\sigma u + \Omega_{31} u + \Omega_{32} v + \Omega_{33} w \end{bmatrix} \qquad \mathbf{H}_{\text{nc}} = \begin{bmatrix} -\alpha_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \alpha_1 p_1 \\ \alpha_2 p_2 \\ -c \\ 0 \end{bmatrix}$$

*4.3.3. Relaxation procedure*

The relaxation procedure is present in two locations of the multiphase AMR algorithm, *i.e.*, at the end of each time step (reference to the relaxation computation in point 3 of the advancing procedure Section 4.3.2) and in the unrefinement procedure (see next paragraph). This relaxation procedure is fully detailed in [36] and its aim is to take into account the mechanical equilibrium *via* pressure relaxation and its respective impact on the volume fraction and density of each phase. If pressure relaxation is activated, a mechanical-equilibrium model is solved using the pressure-disequilibrium model (6) (see [38] for details).

When cells are joined during an unrefinement procedure, an averaging procedure has to be performed before removing child cells (see point 4 of Section 3.4.3). When dealing with a multiphase model as Model (6), a supplementary relaxation procedure also has to be added to keep thermodynamical consistency in mixture regions. The point 4 of Section 3.4.3 is then replaced by the pseudocode:

```
if (l < l_max) {
  for (split cells i of level l) {
    if (ξ < ξ_join & level of each neighboring cell ⩽ l + 1 & child cells are not split) {
      Cell i is unrefined with children averaging and relaxation procedure to
      overwrite the values of cell i;
    }
  }
}
```

## 5. Numerical results

The interests of the new AMR method to solve multiphase compressible flows are presented through 5 typical configurations involving compressible flows of water and air: Transport, shock tube, surface-tension flow, cavitation and water-droplet atomization in 1D and

3D. Each test is performed with quantitative comparisons regarding exact solutions or results using non-AMR method and has been carefully chosen to highlight a particular aspect of the method:

- 1D transport test: A very simple test of motion of a contact discontinuity between two gases. This test proposes a deep discussion about the influence of the refinement criteria on AMR results.
- 1D liquid/gas shock tube: This is a typical test showing that the method is efficient and provides physically good results for multiphase compressible flows. Exact solutions are available for such test that shows an interface between liquid and gas (large density ratio), a shock wave and strong rarefaction waves.
- 3D surface-tension test: This is a typical example that uses the model of Schmidmayer et al. [38] presented in Section 4.3.
- 3D cavitation test (parallel scaling test): The possibilities of the method and its efficiency on parallel architecture are study in this test.
- 3D water-droplet atomization: This test shows the method possibilities to treat a real application.

All computational results are obtained using the open-source code ECOGEN [37] where this new AMR method is implemented. The flow solver is based on a MUSCL-like scheme (second-order in space and time) and the Harten-Lax-van Leer Contact (HLLC) approximate Riemann solver [17, 36]. Note that for the second-order in space, the scheme requires slopes that are determined at faces. At the coarse/fine boundaries, the slopes of the child faces of the same parent face are arithmetically averaged to obtain a unique slope used on the coarse side. The equation of state (EOS) for the air obeys to the ideal-gas law:

$$p_{\text{air}} = (\gamma_{\text{air}} - 1)\, \rho_{\text{air}} e_{\text{air}},$$

with $\gamma_{\text{air}} = 1.4$. The water obeys the stiffened-gas EOS:

$$p_{\text{water}} = (\gamma_{\text{water}} - 1)\, \rho_{\text{water}} e_{\text{water}} - \gamma_{\text{water}} p_{\infty,\text{water}},$$

where the stiffened-gas-EOS parameters are $\gamma_{\text{water}} = 4.4$ and $p_{\infty,\text{water}} = 6.10^8$ Pa. For details about the thermodynamic closure and EOS-parameter determination, one can refer to [21].

### *5.1. 1D transport test*

The goal of the first test is to show the influence of the different criteria of refinement in comparison to a fully-refined, non-AMR mesh on a very simple 1D transport test case. To avoid any potential complex interaction with a multiphase model, this first test is done for a single-phase flow governed by Euler equations. Note that for equivalent comparisons, the cell size for the non-AMR mesh is the same than for the cells at the highest level ($l_{\max}$) of the AMR method.

The initial condition consists in a segment of air at high density ($\rho_{\text{discontinuity}} = 10$ kg.m$^{-3}$) while the surrounding air is at lower density ($\rho_{\text{environment}} = 1$ kg.m$^{-3}$). The flow velocity in the whole domain is set to $u = 50$ m.s$^{-1}$, the pressure is uniform and the Neumann boundary condition is used. The center of the high-density segment is initially set at the coordinate 0.3 m and has a length of 0.2 m. The simulation time is $t = 8$ ms. Only the density profile is presented since the pressure and the velocity remain uniforms.

Concerning the AMR method, 4 refinement levels ($l_{\max} = 4$) are involved which means there are 5 levels in total including the initial one. The mesh is initialized with $N = 10$ cells and then the corresponding number of cells for a full refinement is $N$x$2^{l_{\max}} = 160$. The choice of the refinement criteria is one of the most difficult part in an AMR method and it is completely case-dependent. The gradient refinement criterion in this first test is obviously based on the density variation, then, the analysis is restricted to other important parameters involved in the refinement criterion ($\epsilon$, $\xi_{\text{split}}$ and $\xi_{\text{join}}$)

The information concerning the AMR data is given in Table 2 for 4 different sets of AMR criterion values.

| Test case | Initial number of cells | $l_{\max}$ | Equivalent mesh | $\epsilon$ | $\xi_{\text{split}}$ | $\xi_{\text{join}}$ | Max number of cells involved |
|---|---|---|---|---|---|---|---|
| Case 1 | 10 | 4 | 160 | 0.1 | 0.5 | 0.5 | 50 |
| Case 2 | 10 | 4 | 160 | 0.1 | 0.5 | 0.1 | 61 |
| Case 3 | 10 | 4 | 160 | 0.1 | 0.1 | 0.1 | 73 |
| Case 4 | 10 | 4 | 160 | 1 | 0.1 | 0.1 | 56 |

Table 2: AMR data for the 1D transport test case using the Euler model.

Figure 6 shows the results obtained with the different criteria of Table 2. The initialization using AMR method is shown and is identical for each test case. The non-AMR result is also shown but is partially hidden by the AMR result of the third test (Case 3) since this last gives as good result as the non-AMR case.

On the left image of Figure 6, results using three combinations of $\xi_{\text{split}}$ and $\xi_{\text{join}}$ are compared:

- One can observe the difference concerning the shape of the results for the tests Case 1, where $\xi_{\text{split}} = \xi_{\text{join}} = 0.5$, and Case 2, with $\xi_{\text{split}} = 0.5$ and $\xi_{\text{join}} = 0.1$. Note for the case when $\xi_{\text{split}}$ and $\xi_{\text{join}}$ are equal, the diffusion of the variable $\xi$ plays the role of a buffer which indirectly avoids most of the effects of refinement-unrefinement of the cells at very close times. The result with a lower $\xi_{\text{join}}$ is closer to the non-AMR one at the head of this heaviside function. At the rear, the results are similar between the two AMR tests and they have a lower density than the non-AMR result. In the two cases, the matching with the non-AMR result is better at the head of the discontinuity. One can conclude that the refinement and unrefinement processes give better results in the upwind direction and that having similar values of the criteria yield to better results in regards to symmetrical aspect. In the following tests, to limit this non-symmetrical aspect, the values of those two criteria are always taken equals. One can also note that

the maximum number of cells involved is higher in the second test case than in the first one (see Table 2) because its joining criterion is lower.

- In the case where the two criteria are taken with a lower value ($\xi_{\text{split}} = \xi_{\text{join}} = 0.1$), the result (Case 3) is in better agreement with the non-AMR result, not only at the head of the discontinuity but also at the rear. Indeed, the values of these criteria indirectly give the number of refined cells around a detected discontinuity (detected through the gradient-criterion limited value $\epsilon$). The lower the values of $\xi_{\text{split}}$ and $\xi_{\text{join}}$, the greater the number of refined cells around the discontinuity.

The results on the right image of Figure 6 shows the importance of the gradient-criterion limited value $\epsilon$ with the comparison between Cases 3 and 4 where $\epsilon = 0.1$ and $\epsilon = 1$, respectively. The smaller the criterion value, the closer the result is to the non-AMR method. However, as shown in Table 2 with the maximum number of cells involved, it is important to note that this value has to be well chosen, not only to be close to the equivalent non-AMR solution, but also to not refine all the mesh and thus guarantee computational efficiency. Furthermore, this criterion compares the normalized variation of the chosen physical variable, here density, with the value of $\epsilon$ (Equation (4)). Because it is normalized with the minimum density of the cell where the calculation is done or of its neighboring cells, plus because the absolute numerical diffusion is the same on the two sides of the discontinuity, the normalized variation is higher on the side of the lower density and then this side is more refined. Thus, the head of the heaviside discontinuity propagates through a mesh containing more cells at the highest level than at the rear of the discontinuity, and it explains the non-symmetrical aspect that is clearly observable for high values of $\epsilon$.

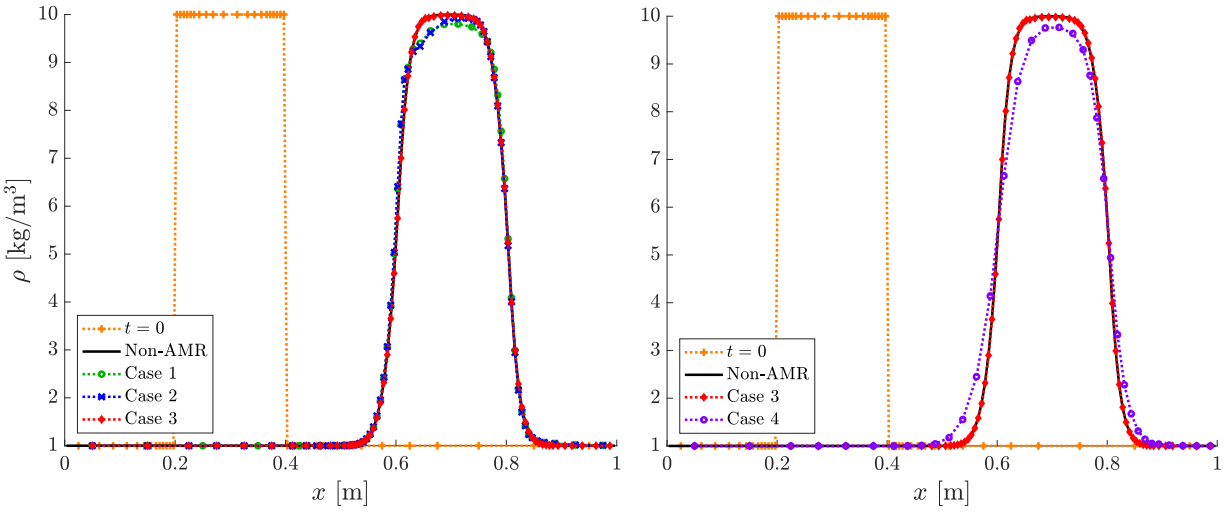


Figure 6: Density $\rho$ along the $x$-direction for the 1D transport test case using the Euler model. Initialization setup ($t = 0$) is shown using the AMR method and results are given for different test cases: One non-AMR and four AMR with different sets of AMR criteria.

The variation of the initial number of cells and the number of levels for a constant equivalent non-AMR mesh is not shown here because it leads to close results, even if the

total number of cells involved are different. Then, having the lowest initial number of cells with the highest number of refinement levels seems the best option since the results are similar and the computational time should be lower. "Should be" because of the balance between the computational time lost in the recursive integration and refinement procedures with a high number of levels and the gain between the computation of two different initial meshes. Moreover, using high-order methods reduce the number of cells involved in the computation due to the sharper discontinuities. This last point partially counterbalances the additional computational time involved by these high-order methods.

In the following, the impact of the criteria is no longer shown but it is important to keep in mind that this choice is crucial to obtain good results and simultaneously guarantee computational efficiency.

### *5.2. Liquid/gas shock tube*

The shock-tube test for multiphase flow is the flow generated by the initial contact of a high and a low-pressure chamber. The high-pressure chamber is filled with water at the pressure $p = 1.10^9$ Pa and with a density of $\rho = 1,000$ kg.m$^{-3}$. In the low-pressure chamber, there is air with a pressure of $p = 1.10^5$ Pa and a density of $\rho = 50$ kg.m$^{-3}$. In both chambers, the initial fluid velocity is $u = 0$ m.s$^{-1}$. The length of the tube is 1 m and the initial discontinuity is located at 0.7 m with the high-pressure chamber on the left and the low-pressure one on the right. The simulation time is $t_{\text{final}} = 241$ $\mu$s.

In the shock-tube test, because of the physics involved, the mixture-density and mixture-pressure variations are chosen to define the refinement criteria. The other information concerning the AMR data is given in Table 3 where the equivalent mesh indicates the number of cells of a fully-refined mesh (which also indicates the number of cells of the non-AMR method).

| Initial number of cells | $l_{\max}$ | Equivalent mesh | $\epsilon$ | $\xi_{\text{split}}$ | $\xi_{\text{join}}$ | Max number of cells involved |
|---|---|---|---|---|---|---|
| 10 | 8 | 2,560 | 0.1 | 0.1 | 0.1 | 210 |

Table 3: AMR data for the 1D shock-tube test case.

Figure 7 shows the exact solution and the simulation solution of the AMR method. A shock is propagating from the left to the right, followed by the contact discontinuity while the expansion waves propagate in the opposite direction. In that case, the AMR mesh starts with 10 cells and the value of $l_{\max}$ is chosen equal to 8. The maximum number of cells involved using the AMR method is equal to 210 instead of $2,560$ for the equivalent non-AMR mesh (see Table 3).

Results analysis shows that:

- The shock wave and the contact discontinuity are in very good agreement with the exact solution.

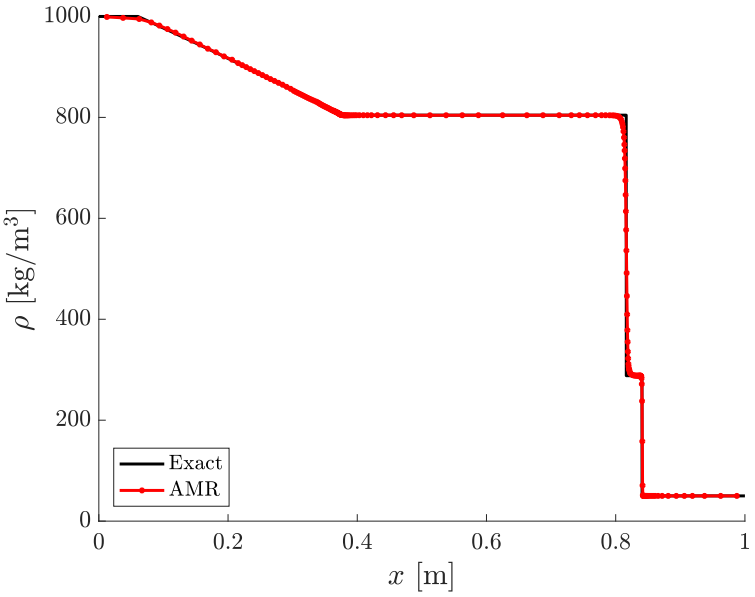


Figure 7: Density $\rho$ along the $x$-direction for the 1D shock-tube test case. Exact solution and AMR result are shown.

- Concerning the expansion waves: Even if the result is satisfactory, one can observe that the number of cells at the head of the expansion waves (left part of the expansion waves in the images) decreases. This point is clearly observable in Figure 8 where the distribution of the refinement levels is shown for $t = 0$ and $t = t_{\text{final}}$. The latter corresponds to the result shown in Figure 7. Initially, the mesh is only fully refined around the discontinuity (located at 0.7 m). At the end of the simulation, the mesh is fully refined around the shock, the contact discontinuity and the tail of the expansion waves (right part of the expansion waves in the images). In fact, due to the smooth variation of the thermodynamic variables, it is difficult to find a good criterion to refine the expansion waves without refining most of the domain.

Table 4 contains data to point out the ability of the AMR method to lead to a real gain in comparison to the non-AMR method (not shown) concerning the computational time (a CPU time ratio of 26 is reached) and the memory involved, even in 1D simulation. Note that the recorded memory involved is only the highest memory used during the simulation for the AMR method, this one evolves during the simulation in function of the mesh distribution.

| | | | Ratio between non-AMR and AMR | |
|---|---|---|---|---|
| Mesh | Computational time | Memory (highest involved for AMR) | Computational time factor | Memory factor |
| Non-AMR | 26s | 6.0Mo | 26.0 | 2.61 |
| AMR | 1s | 2.3Mo | 1 | 1 |

Table 4: Performances for the different test cases of the 1D shock-tube test case.

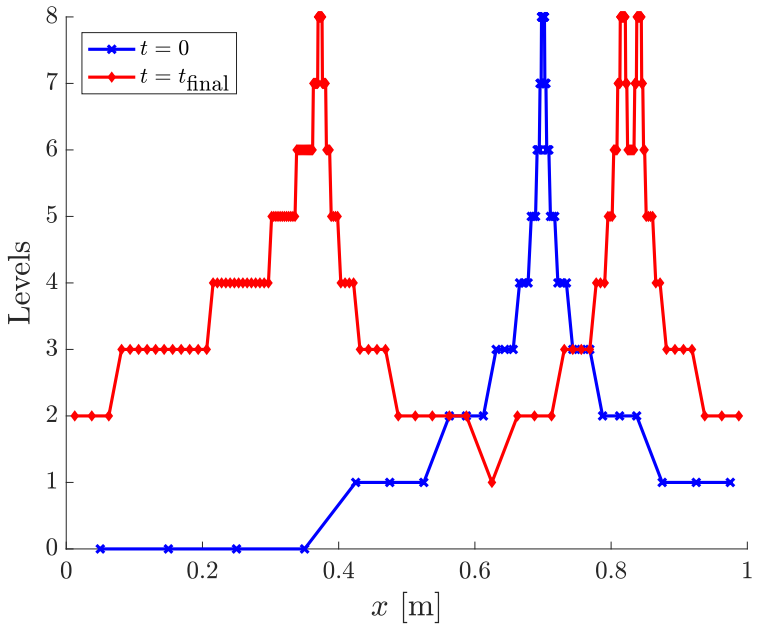


Figure 8: Distribution of the refinement levels along the $x$-direction for the 1D shock-tube test case using the AMR method. Initialization $t = 0$ and final result $t = t_{\text{final}}$ are shown. Each marker corresponds to a cell where the physical computations occur.

### *5.3. 3D surface-tension test: Laplace jump*

In the following 3D simulation test, a droplet of water is placed in an air environment. At the steady state, the Laplace pressure jump must be recovered thanks to the surface tension. The expression of the theoretical pressure jump for a sphere is:

$$[p] = \frac{2\sigma}{R},$$

where $[p]$ expresses the pressure jump between inside and outside the droplet, here $p_{\text{water}} - p_{\text{air}}$.

For the simulation, the droplet of water is initially located at the center of a 3D domain (0.75 m x 0.75 m x 0.75 m) filled with air (see Figure 9). To accentuate the impact of the surface-tension force, the used stiffened-gas EOS parameters for water are $\gamma_{\text{water}} = 2.1$ and $p_{\infty,\text{water}} = 1.10^6$ Pa and the size of the droplet is emphasized (radius of 0.15 m) as well as the surface-tension coefficient ($\sigma = 800$ N.m$^{-1}$). The pressure initially is the same in each fluid and is equal to $p = 1.10^5$ Pa, the velocity is null ($\mathbf{u} = \mathbf{0}$ m.s$^{-1}$), the densities are $\rho_{\text{air}} = 1$ kg.m$^{-3}$ in air and $\rho_{\text{water}} = 1,000$ kg.m$^{-3}$ in water, and an outgoing pressure waves boundary condition is used. The simulation time to obtain a converged result is $t = 0.59$ s.

The information concerning the AMR data is given in Table 5. To correctly treat surface tension, only the thickness and the curvature of the interface are important. Thus, the gradient refinement criterion is only based on the volume-fraction variation. Moreover, the diffusion is accentuated when dealing with higher dimensions and it also directly impacts the smoothing of the $\xi$ variable. Thus, the $\xi$ criteria are taking lower than in the 1D case, *i.e.*, $\xi_{\text{split}} = 0.02$ and $\xi_{\text{join}} = 0.02$.

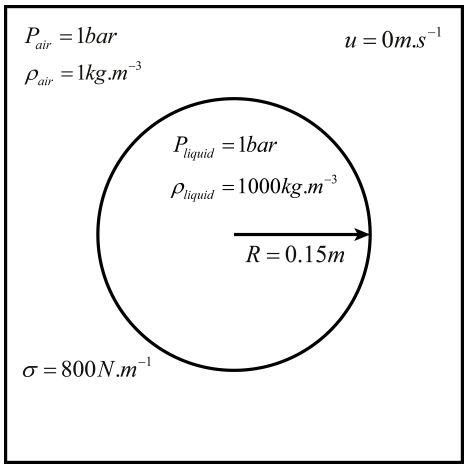


Figure 9: 2D sketch of the initial conditions for the 3D simulation of a liquid droplet placed in air.

For this test, only the AMR method was computed due to the too long computation of a non-AMR method in 3D. As shown in Table 5, the number of cells that involves a non-AMR computation is of $4,096,000$ cells while the AMR one only involves a maximum of $267,224$ cells, and this last one already took more than 16 days (on one core). Indeed, the time step in compressible flows is governed by wave speeds and, here, it is around $\Delta t = 1.5\text{x}10^{-6}$ s. To reach the steady state ($t = 0.59$ s), $3.93\text{x}10^{5}$ time steps are therefore necessary. Moreover, even if it is not shown here, waves induced by the computation of surface tension are coming from the interface locations at the beginning of the simulation. These waves slowly decrease and disappear while reaching the steady state. In the case of the AMR method, the waves are smoothed inside and outside the droplet because of the coarser mesh and thus, less oscillations occur than with the fully non-AMR mesh. It results that the AMR method accelerates the convergence of the simulation to a steady state in comparison to non-AMR method.

| Initial number of cells | $l_{\max}$ | Equivalent mesh | $\epsilon$ | $\xi_{\text{split}}$ | $\xi_{\text{join}}$ | Max number of cells involved | Computational time |
|---|---|---|---|---|---|---|---|
| 20x20x20 | 3 | 4,096,000 | 0.1 | 0.02 | 0.02 | 267,224 | 16d 21h 53m |

Table 5: AMR data for the 3D surface-tension test case. The simulation was performed on one core.

The theoretical pressure jump of Laplace for this test case is $[p] = 10720$ Pa and it is well recovered in the 3D simulation when using the AMR method as shown in Figures 10 and 11.

It is also interesting to compare the computational time spent to manipulate the tree structures of the AMR method to the time spent to update the solution. In this simulation, at the final time of computation, less than 5% of the computational time was devoted by the algorithm to manage AMR. This computational time includes smoothing of refinement indicator, refinement and unrefinement of the mesh.

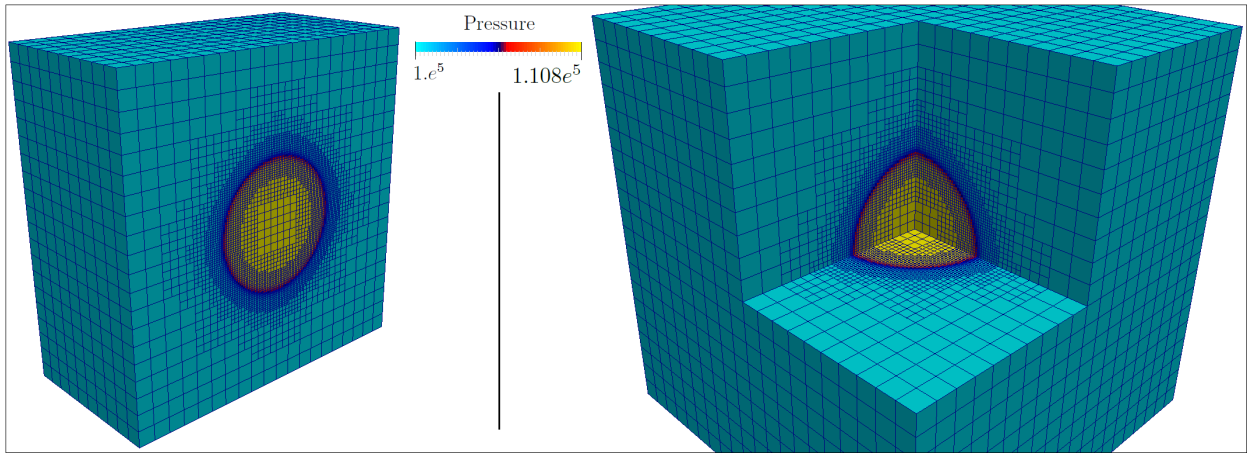


Figure 10: Two different cut views of the the 3D surface-tension test case. Pressure $p$ is shown for the converged state.

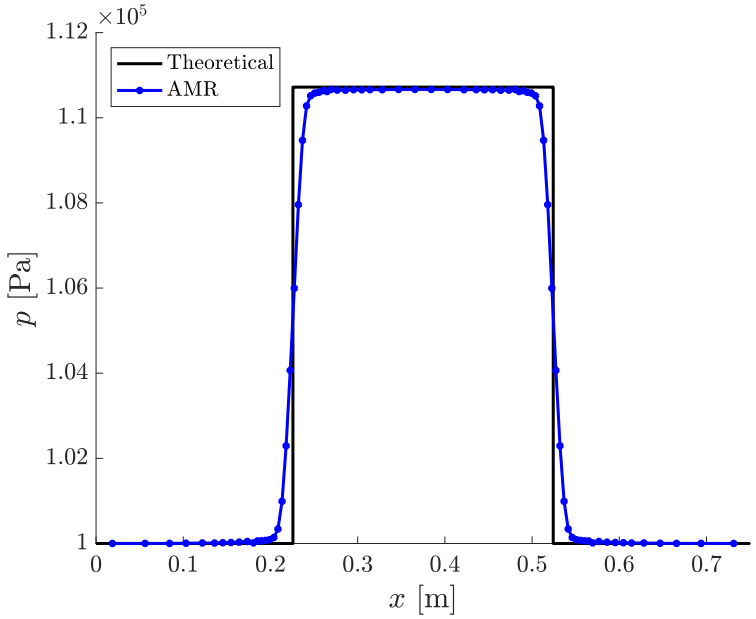


Figure 11: Pressure $p$ at the converged state for the 3D surface-tension test case. Theoretical and simulation results are shown. The converged time is $t = 0.59$ s.

### *5.4. Parallel scaling on a 3D cavitation test case*

The ability of the method on parallel scaling is presented in this section. Note that adaptive parallel load balancing has not been implemented yet in the current version of the code.

Collapses of three consecutive air bubbles induced by a shock wave in water is considered. The spherical air bubbles are initially at equilibrium in a water environment around atmospheric state. The initial diameter of the bubbles is $D = 1$ mm and the bubbles are separated by $h = 1.5$ mm. The collapses are initiated by a shock wave travelling from the

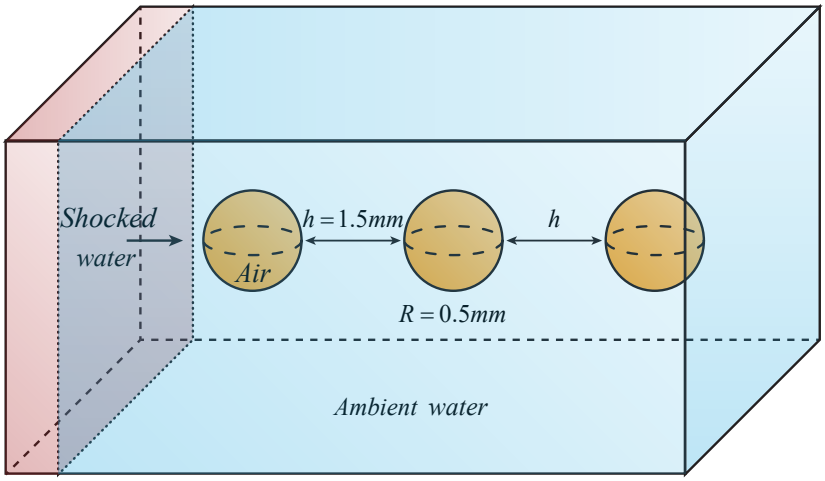


Figure 12: Sketch of the initialization of the cavitation test.

left to the right (see Figure 12 for initialization sketch). The simulation time is $t_{\text{final}} = 10$ $\mu$s and the surface-tension effects are neglected. The initial densities, pressures and velocities in the $x$-direction are:

- Shocked water: $p = 500.10^5$ Pa, $\rho = 1,018.3$ kg.m$^{-3}$ and $u = 29.95$ m.s$^{-1}$,
- Ambient: $p = 10^5$ Pa, $\rho_{\text{water}} = 1,000$ kg.m$^{-3}$, $\rho_{\text{air}} = 1.2$ kg.m$^{-3}$, and $u = 0$ m.s$^{-1}$.

The 3D computations are performed on a quarter of the whole domain with two symmetrical boundary conditions. The initial AMR mesh contains 75x25x25 computational cells in a physical domain of 12 mm x 4 mm x 4 mm. Parallel-scaling test cases are computed for a maximum number of refinement levels of $l_{\max} = 2$ and 3, and from 6 to 192 cores. Furthermore, an additional test case with $l_{\max} = 4$ has been computed on 96 cores to show the capabilities of the algorithm. The complete information concerning the AMR data is given in Table 6. The gradient refinement criterion is based on volume-fraction, mixture-pressure and mixture-density variations in order to catch the dynamics of interfaces as well as waves propagating into water and air.

| Initial number of cells | $l_{\max}$ | Equivalent mesh | $\epsilon$ | $\xi_{\text{split}}$ | $\xi_{\text{join}}$ |
|---|---|---|---|---|---|
| 75x25x25 | 2 | 3.00x10$^6$ | 0.08 | 0.02 | 0.02 |
| 75x25x25 | 3 | 2.40x10$^7$ | 0.08 | 0.02 | 0.02 |
| 75x25x25 | 4 | 1.92x10$^8$ | 0.08 | 0.02 | 0.02 |

Table 6: AMR data for the cavitation test.

Figure 13 shows the computational wall time of the total simulation $T$ and the corresponding speedup function of the number of cores. The speedup is expressed as:

$$\text{Speedup} = \frac{T_1}{T_k} C_1 \ ,$$

where subscripts 1 and $k$ refer to the base simulation (the one done with 6 cores) and the current simulation, respectively, while $C$ corresponds to the number of cores (here $C_1 = 6$). One can note that even if adaptive parallel load balancing is not present in the code yet, the algorithm is performing reasonably well and the trend seems to stay approximately constant. The speedup gives slightly better results for the cases with $l_{\max} = 2$ than the ones with $l_{\max} = 3$. On average, the speedup factor when increasing the number of cores by two is 1.75 and 1.63 for $l_{\max} = 2$ and 3, respectively, instead of an ideal factor of 2. This last is obviously explained by the lack of the previously mentioned adaptive parallel load balancing. An almost linear speedup is thus expecting in future versions of the code or for readers who wish to implement this AMR algorithm on an adaptive-parallel-load-balancing structure.

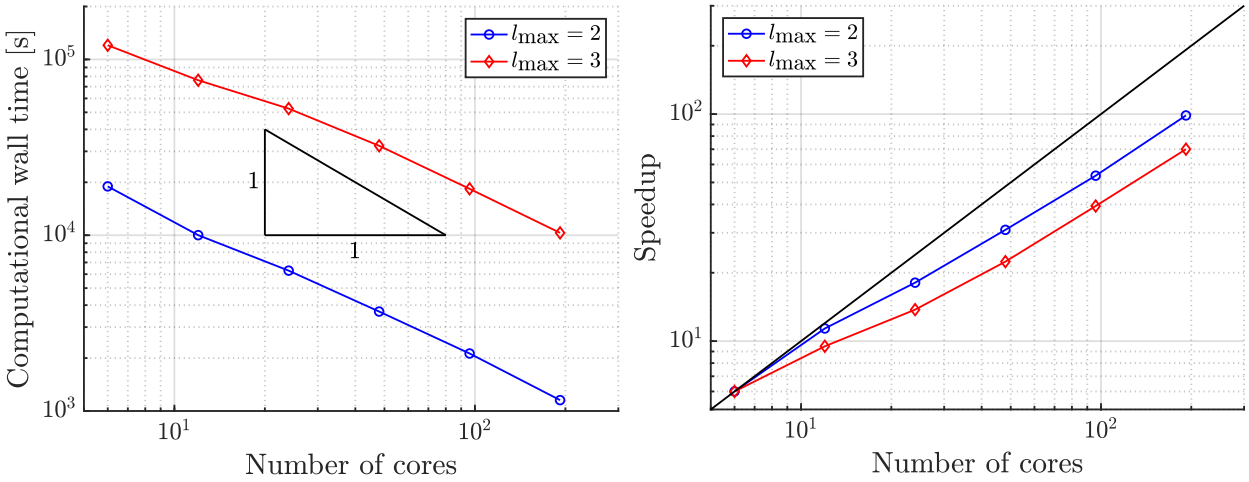


Figure 13: Computational wall time (left) and speedup (right) function of the number of cores for the cavitation test and for simulations with $l_{\max} = 2$ and 3.

Few typical flow results are also presented in Figure 14 for the simulation with 4 levels of refinement. The pressure (colors) at the boundaries of the computed 3D domain and air-water-interface isosurfaces (cyan, $\alpha = 0.5$) are shown for different times. An adaptive pressure scale is used due to the significant pressure variation during the collapses of bubbles. Indeed, when a bubble collapses under such gradients of pressure between the pressure behind the initial shock wave in the water and the pressure inside the bubble, the implosion of the bubble produces a strong induced shock wave when the bubble reaches its minimum radius. This last shock wave can lead to a significantly higher pressure pick than the pressure produced by the initial shock. Here, the emitted shock from the collapse of the first bubble is then going to enhance the collapses of the successive bubbles. Multiple bubble collapses and rebounds are thus observed along with the emitted shocks.

The computational wall time of this last and significant 3D test case has been reasonably short (44 hours and 51 minutes on 96 cores). Furthermore, over these different cavitation simulations, a maximum of 8.65% of total computational time has been recorded to be devoted to AMR management.

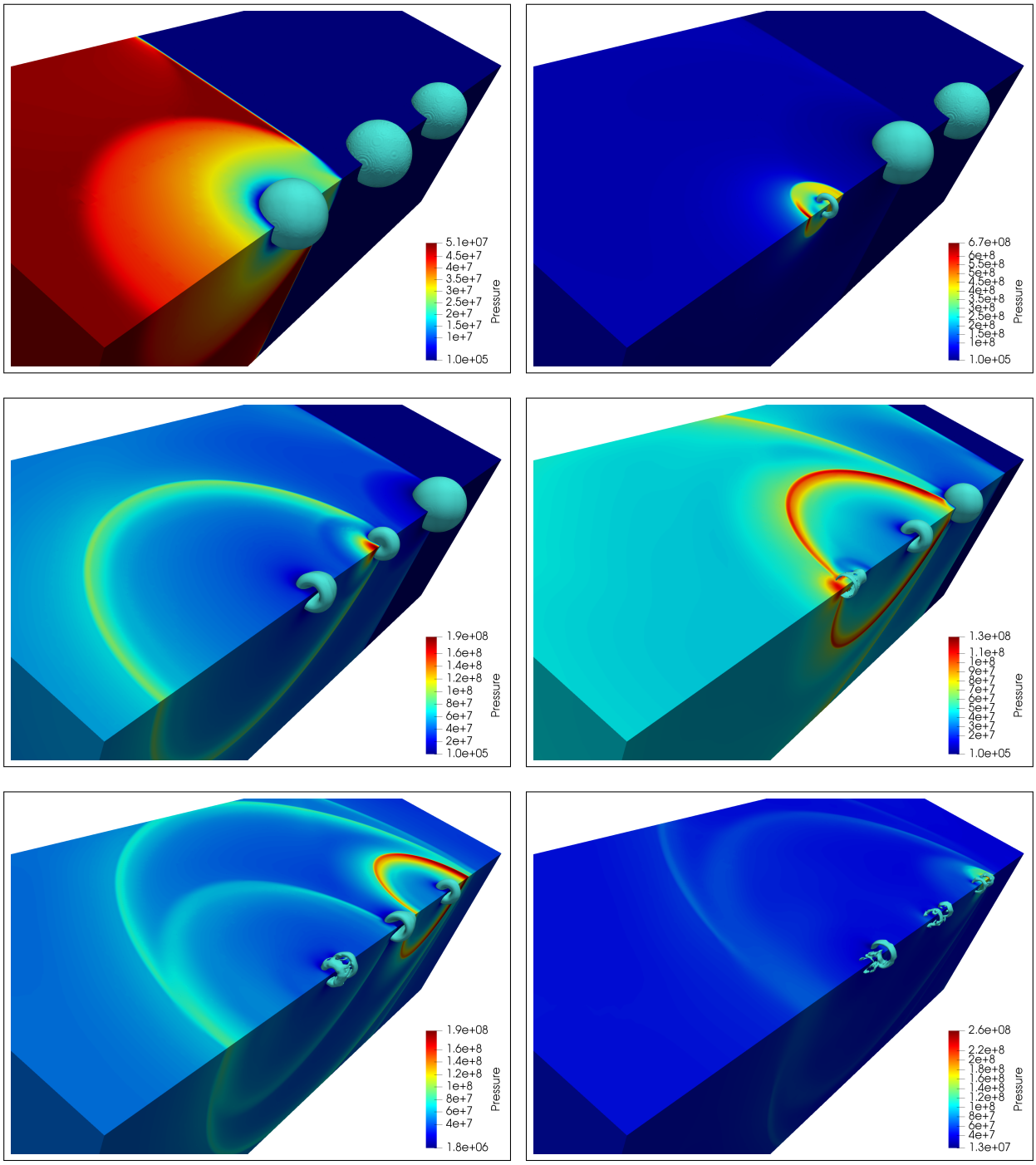


Figure 14: Pressure (colors) at the boundaries of the computed 3D domain and air-water-interface isosurfaces (cyan, $\alpha = 0.5$) for the cavitation test case with $l_{\max} = 4$. From left to right and from top to bottom, results at times: $t = 1.25$ $\mu$s, 2.5 $\mu$s, 3.75 $\mu$s, 5 $\mu$s, 6.25 $\mu$s and 8.75 $\mu$s. Adaptive pressure scale is used due to the significant pressure variation during the collapses of bubbles.

*5.5. 3D water-droplet atomization*

The ability of the method to solve complex flows is presented in this section. Atomization of a 3D water droplet induced by a high-speed flow is considered. A spherical water droplet is placed in an air environment already at a shocked state. The initial diameter of the droplet is $D = 6.4$ mm and the shocked state is the corresponding one behind a shock wave of Mach number 1.3 in atmospheric air (see Figure 15 for initialization sketch). The initial densities,

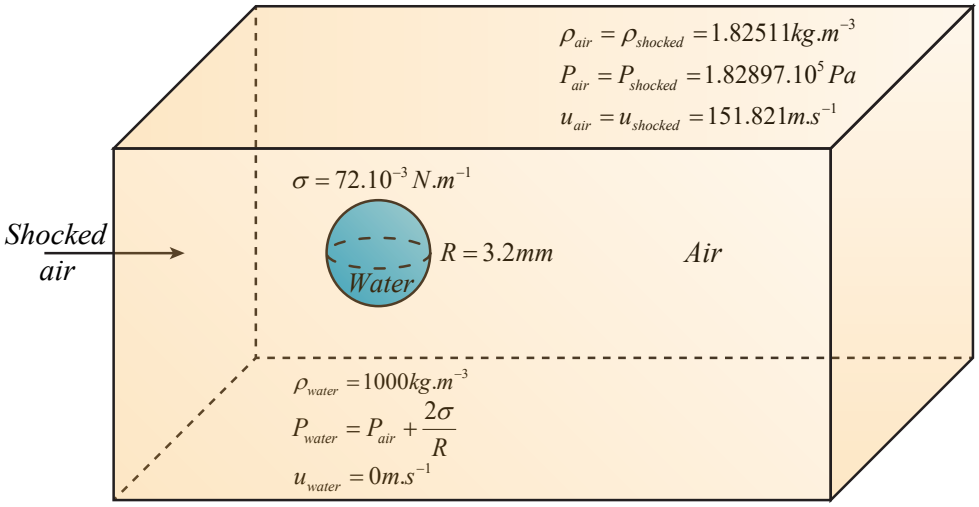


Figure 15: Sketch of the initialization of the droplet-atomization test.

pressures and velocities in the $x$-direction are:

- Air (shocked): $\rho = 1.82511$ kg.m$^{-3}$, $p = 1.82897.10^5$ Pa and $u = 151.821$ m.s$^{-1}$,
- Water: $\rho = 1,000$ kg.m$^{-3}$, $p = p_{\text{air}} + 2\sigma/R$, and $u = 0$ m.s$^{-1}$.

The 3D computation is performed on a quarter of the whole domain with two symmetrical boundary conditions. Shocked air is entering at the left boundary and the Newmann boundary conditions is used elsewhere. The initial AMR mesh contains 250x50x50 computational cells in a physical domain of 250 mm x 50 mm x 50 mm. The maximum number of refinement levels is $l_{\max} = 4$ and the maximum number of cells recorded is 14.68219x10$^6$ which is only 0.57% of the cells that would have been involved with an equivalent non-AMR mesh (2.56x10$^9$ cells). The complete information concerning the AMR data is given in Table 7. The gradient refinement criterion is based on volume-fraction, mixture-pressure and mixture-velocity variations.

| Initial number of cells | $l_{\max}$ | Equivalent mesh | $\epsilon$ | $\xi_{\text{split}}$ | $\xi_{\text{join}}$ | Max number of cells involved |
|---|---|---|---|---|---|---|
| 250x50x50 | 4 | 2.56x10$^9$ | 0.02 | 0.02 | 0.02 | 14.68219x10$^6$ |

Table 7: AMR data for the droplet-atomization test.

An example of the mesh distribution is given in Figure 16 at time $t = 1,000$ $\mu$s. In the background, the wireframe is represented for the computational domain with the colors of the mixture-density gradients (strong gradients in dark red and zero gradients in dark blue). The front image represents three quarters of the complete domain with apparent surfaces. One of the two observable surfaces is shown with the mesh and the other without. The refinement at the highest level occurs around the droplet and at its rear due to the vortices that are involved.

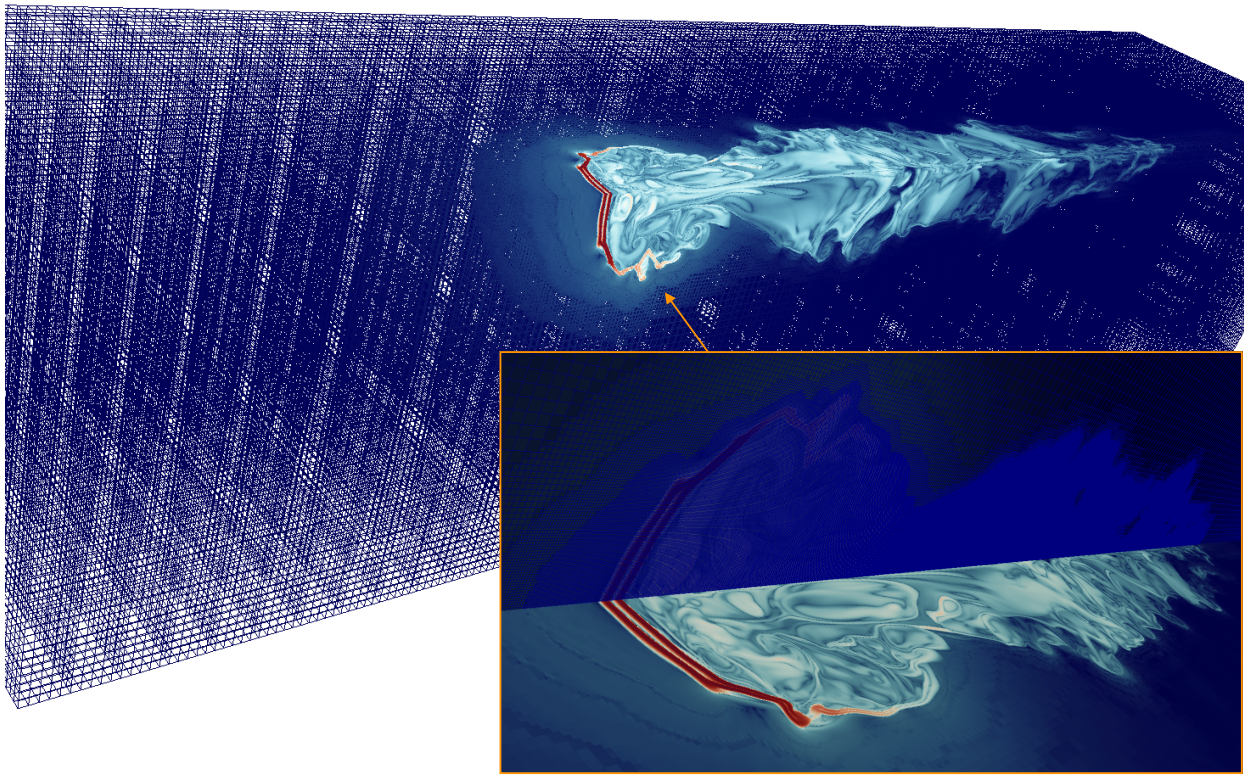

Figure 16: Two images with the mesh and the mixture-density gradients apparent (strong gradients in dark red and zero gradients in dark blue) for the droplet-atomization test at time $t = 1,000\mu$s. Refinement of the mesh is observable around the droplet and at its rear (vortices).

Time evolution of the droplet shape is shown in Figure 17. In this figure, water droplet density isosurfaces are represented ($\alpha_{\text{water}} = 0.5$). The wave propagating on the sides of the droplet due to its compression and stretching is observable. This wave is slowly making filaments all around the droplet and holes in these filaments are appearing at $1,000$ $\mu$s, which later will produce the reduced-size droplets. Figure 18 shows different views of the droplet isosurface at time $1,000$ $\mu$s. $\alpha_{\text{water}} = 0.001$ is used for the isosurfaces to show all the filaments and potential reduced-size droplets. Those last ones can also be interpreted as the mist of micrometer water droplets around and at the rear of the initial droplet. The mixture velocity norm is also shown to observe the speed of the smaller waves which occur on the surface of the droplet, and the velocity of the filaments and of the smaller droplets that are thrown away from the initial droplet (high mixture-velocity norms in dark red and low ones

in dark blue). One may also notice grid imprint at very long times. The latter has been discussed in Meng [26] and is not related to AMR. None concrete solutions are yet known even if one should expect less grid imprint for methods with higher order of accuracy.

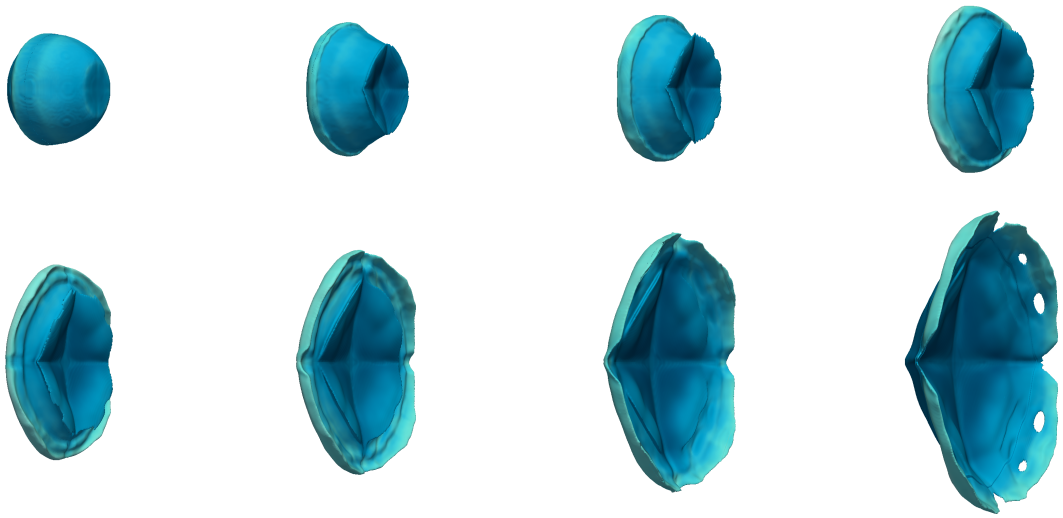

Figure 17: Side views of the droplet isosurface for the atomization test. Results are shown using $\alpha_{\text{water}} = 0.5$ for the isosurfaces and at times 200 $\mu$s, 400 $\mu$s, 500 $\mu$s, 600 $\mu$s, 700 $\mu$s, 800 $\mu$s, 900 $\mu$s and $1,000$ $\mu$s from left to right and from top to bottom.

## 6. Conclusion

A new Adaptive Mesh Refinement (AMR) method using dual trees for cells and cell faces has been presented. The addition of a second face-tree structure presents the advantages to reduce the number of operations during the time-stepping integration (neighboring-cell searching procedure is suppressed) and to simplify the general algorithm in comparison to a fully-threaded-tree method. It has been proven that the memory overhead involved in the method was reasonably controlled. Quantitative data regarding computational time as well as memory have been provided and have demonstrated that the method is efficient on single-phase compressible flows even if it is particularly devoted to multiphase, diffuse-interface, compressible models. Its abilities have been particularly presented through test cases that involve flows governed by the model of Schmidmayer et al. [38] for simulating compressible multiphase flows including surface tension.

## References


[1] M. Aftosmis, J. Melton, and M.J. Berger. Adaptation and surface modeling for Cartesian mesh methods. In *AIAA Paper, 12th Computational Fluid Dynamics Conference*, page 1725, 1995.

[2] M. Anderson, E.W. Hirschmann, S.L. Liebling, and D. Neilsen. Relativistic MHD with adaptive mesh refinement. *Classical and Quantum Gravity*, 23(22):6503, 2006.

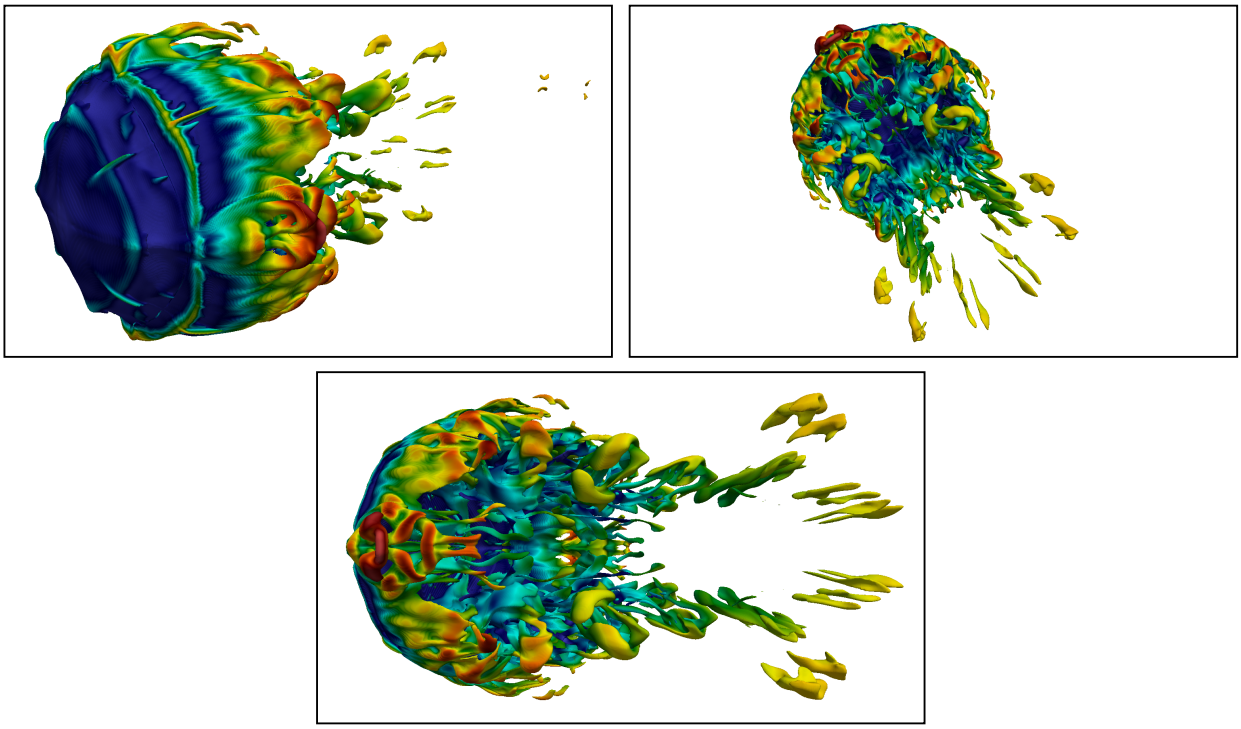

Figure 18: Different views of droplet isosurface for the atomization test. Results are shown using $\alpha_{\text{water}} = 0.001$ for the isosurfaces and at time $1,000$ $\mu$s. Colors represents mixture-velocity norm (dark red shows strong mixture-velocity norms and dark blue shows small ones).


[3] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[4] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[5] C. Burstedde, D. Calhoun, K. Mandli, and A. R. Terrel. ForestClaw: Hybrid forest-of-octrees AMR for hyperbolic conservation laws. *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, 25:253–262, 2014.

[6] C. Burstedde, L. C. Wilcox, and O. Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[7] X. Chen and V. Yang. Thickness-based adaptive mesh refinement methods for multi-phase flow simulations with thin regions. *Journal of Computational Physics*, 269:22–39, 2014.

[8] W.J. Coirier. An adaptively-refined, Cartesian, cell-based scheme for the Euler and Navier-Stokes equations. *Ph.D. thesis, University of Michigan*, 1994.

[9] P. Colella, D. T. Graves, T. J. Ligocki, D. F. Martin, D. Modiano, D. B. Serafini, and B. Van Straalen. Chombo software package for AMR applications design document. *Available at the Chombo website: http://seesar. lbl. gov/ANAG/chombo/(September 2008)*, 2009.

[10] M. Dumbser, O. Zanotti, A. Hidalgo, and D.S. Balsara. ADER-WENO finite volume schemes with space-time adaptive mesh refinement. *Journal of Computational Physics*, 248:257–286, 2013.

[11] N. Favrie and S.L. Gavrilyuk. Diffuse interface model for compressible fluid–compressible elastic–plastic solid interaction. *Journal of Computational Physics*, 231(7):2695–2723, 2012.

[12] N. Favrie and S.L. Gavrilyuk. Dynamic compaction of granular materials. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 469(2160):20130214, 2013.

[13] N. Favrie, S.L. Gavrilyuk, and S. Ndanou. A thermodynamically compatible splitting procedure in hyperelasticity. *Journal of Computational Physics*, 270:300–324, 2014.

[14] B. Fryxell, K. Olson, P. Ricker, F.X. Timmes, M. Zingale, D.Q. Lamb, P. MacNeice, R. Rosner, J.W. Truran, and H. Tufo. FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1):273, 2000.

[15] E. Han, M. Hantke, and S. Müller. Efficient and robust relaxation procedures for multi-component mixtures including phase transition. *Journal of Computational Physics*, 338:217–239, 2017.

[16] S. Hank, N. Favrie, and J. Massoni. Modeling hyperelasticity in non-equilibrium multiphase flows. *Journal of Computational Physics*, 330:65–91, 2017.

[17] A. Harten, P.D. Lax, and B. van Leer. On upstream differencing and Godunov type schemes for hyperbolic conservation laws. *SIAM Rev.*, 25:33–61, 1983.

[18] R. D. Hornung and S. R. Kohn. Managing application complexity in the SAMRAI object-oriented framework. *Concurrency and computation: practice and experience*, 14(5):347–368, 2002.

[19] A. Hosangadi, V. Ahuja, and S. Arunajatesan. Simulations of cavitating flows using hybrid unstructured meshes. *ASME J. Fluids Eng*, 123:331–340, 2001.

[20] A.M. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, 143(2):519–543, 1998.

[21] O. Le Metayer, J. Massoni, and R. Saurel. Elaborating equations of state of a liquid and its vapor for two-phase flow models. *Int. J. of Thermal Sciences*, 43:265–276, 2004.

[22] O. Le Métayer, J. Massoni, and R. Saurel. Dynamic relaxation processes in compressible multiphase flows. application to evaporation phenomena. In *Esaim: Proceedings*, volume 40, pages 103–123. EDP Sciences, 2013.

[23] P. MacNeice, K. M. Olson, C. Mobarry, R. De Fainchtein, and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer physics communications*, 126(3):330–354, 2000.

[24] J. Massoni, R. Saurel, B. Nkonga, and R. Abgrall. Proposition de methodes et modeles Euleriens pour les problemes a interfaces entre fluides compressibles en presence de transfert de chaleur. *Int. J. Heat and Mass Transfer*, 45:1287–1307, 2002.

[25] J. Melton, M.J. Berger, M. Aftosmis, and M. Wong. 3D applications of a Cartesian grid Euler method. In *AIAA Paper, 33rd Aerospace Sciences Meeting and Exhibit*, page 853, 1995.

[26] J.C. Meng. *Numerical Simulations of Droplet Aerobreakup*. PhD thesis, California Institute of Technology, 2016.

[27] A. Murrone and H. Guillard. Behavior of upwind scheme in the low mach number limit: Iii. preconditioned dissipation for a five equation two phase model. *Computers & Fluids*, 37(10):1209–1224, 2008.

[28] G.S.H. Pau, J.B. Bell, A.S. Almgren, K.M. Fagnan, and M.J. Lijewski. An adaptive mesh refinement algorithm for compressible two-phase flow in porous media. *Computational Geosciences*, 16(3):577–592, 2012.

[29] F. Petitpas, J. Massoni, R. Saurel, E. Lapebie, and L. Munier. Diffuse interface models for high speed cavitating underwater systems. *International Journal of Multiphase Flows*, 35(8):747–759, 2009.

[30] F. Petitpas, R. Saurel, E. Franquet, and A. Chinnayya. Modelling detonation waves in condensed energetic materials: Multiphase CJ conditions and multidimensional computations. *Shock waves*, 19(5):377–401, 2009.

[31] S. Popinet. Gerris: A tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572–600, 2003.

[32] S. Popinet and G. Rickard. A tree-based solver for adaptive ocean modelling. *Ocean Modelling*, 16(3):224–249, 2007.

[33] R. Saurel, S.L. Gavrilyuk, and F. Renaud. A multiphase model with internal degrees of freedom: Application to shock-bubble interaction. *Journal of Fluid Mechanics*, 495:283–321, 2003.

[34] R. Saurel and F. Petitpas. Introduction to diffuse interfaces and transformation fronts modelling in compressible media. In *ESAIM: Proceedings*, volume 40, pages 124–143. EDP Sciences, 2013.

[35] R. Saurel, F. Petitpas, and R. Abgrall. Modelling phase transition in metastable liquids: application to cavitating and flashing flows. *Journal of Fluid Mechanics*, 607:313–350, 2008.

[36] R. Saurel, F. Petitpas, and R.A. Berry. Simple and efficient relaxation methods for interfaces separating compressible fluids, cavitating flows and shocks in multiphase mixtures. *Journal of Computational Physics*, 228(5):1678–1712, 2009.

[37] K. Schmidmayer, A. Marty, F. Petitpas, and E. Daniel. ECOGEN, an open-source tool dedicated to multiphase compressible multiphysics flows. In *53rd 3AF International Conference on Applied Aerodynamics*, 2018.

[38] K. Schmidmayer, F. Petitpas, E. Daniel, N. Favrie, and S.L. Gavrilyuk. A model and numerical method for compressible flows with capillary effects. *Journal of Computational Physics*, 334:468–496, 2017.

[39] R. Teyssier. Cosmological hydrodynamics with adaptive mesh refinement - A new high resolution code called RAMSES. *Astronomy & Astrophysics*, 385(1):337–364, 2002.

[40] A. Tiwari, J.B. Freund, and C. Pantano. A diffuse interface model with immiscibility preservation. *Journal of Computational Physics*, 252:290–309, 2013.

[41] D.P. Young, R.G. Melvin, M.B. Bieterman, F.T. Johnson, S.S. Samant, and J.E. Bussoletti. A locally refined rectangular grid finite element method: application to computational fluid dynamics and computational physics. *Journal of Computational Physics*, 92(1):1–66, 1991.

[42] U. Ziegler. The NIRVANA code: Parallel computational MHD with adaptive mesh refinement. *Computer Physics Communications*, 179(4):227–244, 2008.

[43] M. Zingale, A.S. Almgren, M.G. Barrios Sazo, V.E. Beckner, J.B. Bell, B. Friesen, A.M. Jacobs, M.P. Katz, C.M. Malone, A.J. Nonaka, D.E. Willcox, and A. Zhang. Meeting the Challenges of Modeling Astrophysical Thermonuclear Explosions: Castro, Maestro, and the AMReX Astrophysics Suite. In *Journal of Physics: Conference Series*, volume 1031, page 012024. IOP Publishing, 2018.