GUARANTEED QUALITY-DRIVEN HEXAHEDRAL OVERLAY GRID METHOD

Nicolas Le Goff[1] Franck Ledoux[1] Jean-Christophe Janodet[2] Steven J. Owen[3]

[1]*French Alternative Energies and Atomic Energy Commission (CEA), CEA, DAM, DIF, F-91297 Arpajon, France. nicolas.le-goff@cea.fr, franck.ledoux@cea.fr*
[2]*IBISC, Univ. Evry, Université Paris-Saclay, 91025 Evry, France. jeanchristophe.janodet@univ-evry.fr*
[3]*Sandia National Laboratories (SNL), 1515 Eubank SE Albuquerque, NM, U.S.A. sjowen@sandia.gov*

ABSTRACT

Hexahedral mesh generation using overlay grid methods has the benefit of being fully automatic, requiring minimal user input. These methods follow a mesh-first approach where an initial mesh, usually a grid, is used to overlay the reference geometry. Procedures to modify the initial mesh are then employed to best capture the geometry to get a conformal all-hex mesh [1]. One of the main drawbacks of those methods is the resulting mesh quality. While the interior of the mesh remains the same as the initial mesh, cells located at the material interfaces can end up quite deformed or even inverted, making the mesh totally useless for most numerical simulation codes. Considering an input mesh carrying volume fractions of the materials, the main purpose of the presented work is to ensure a minimal cell quality. Our method draws upon the overlay grid pipeline described in [2] where several steps (cell assignment correction, interface reconstruction, mesh adaptation) are altered to control cell quality.

Keywords: mesh generation, overlay grid, quadrilateral, hexahedral, guaranteed quality

1. INTRODUCTION

The finite element method (FEM) and the finite volume method (FVM) require to discretize the physical domain of interest with a mesh. Depending on the application field and the numerical scheme that is used, mesh properties can be totally different and can go from unstructured tetrahedral meshes to boundary-aligned block structured hexahedral meshes. In this work, we focus on hexahedral meshes and more specifically on the generation of unstructured hexahedral meshes starting from grid-like meshes.

This process is of interest particularly in an industrial context where a full simulation process is built by loosely coupling simulation codes acting on different mesh representations. In our case of study, we consider the "*Euler to Lagrange*" situation where (1) a first simulation code computes a solution to a physical problem onto a mesh (usually a structured grid) made of multi-materials cells, i.e a cell can contain different materials, then (2) a second simulation code acting on a pure unstructured hexahedral mesh is used (see Figure 1 for an illustration of the first stage), where every single cell contains a single material. The first code is said to be Eulerian, while the second one is said Lagrangian. In this context, the Eulerian code only needs a grid to work on, and the Lagrangian code requires a pure hexahedral unstructured mesh that can be created using an overlay grid method. Those methods [3, 4, 5, 2] developed in recent years have dramatically improved the ability to rapidly and automatically generate hexahedral meshes for complex geometries. They rely on a *mesh-first* approach to mesh generation where an initial base grid is used to overlay the reference geometry. Procedures to modify the base grid are also employed to best capture the geometry to define a conformal all-hex mesh.

As the generated mesh is used as an input to a nu-

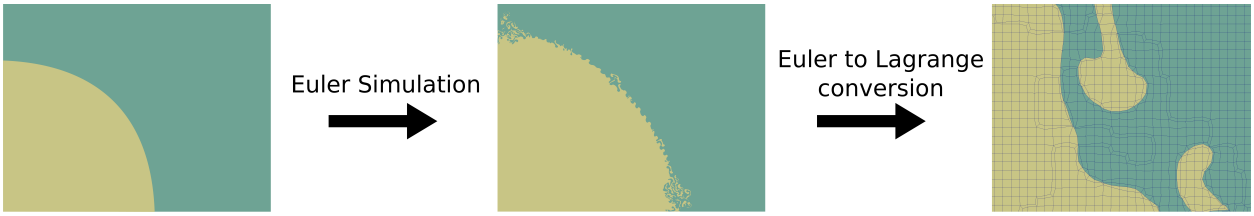


Figure 1: From left to middle, during an hydrodynamic simulation on a grid where two materials are defined (green and yellow) the interfaces between the two materials evolve. For running a Lagrangian code that requires a pure conformal mesh as an input, the mesh must be adapted along with material interfaces. Zoom is performed on the right on the final Lagrangian mesh.

merical simulation code, it must meet some quality requirements dictated by the numerical code. This quality can be defined by a quality metric and a minimum threshold that must be verified by every cell of the mesh. For instance, having non-inverted cells or cells whose scaled Jacobian measure is over 0.2 are possible criteria. The aim of this work is to control as much as possible the quality of the generated mesh. Considering the output of an Eulerian code, a selection of materials to consider, and a couple (quality metric, minimal threshold), we modify an existing overlay grid algorithm to control the mesh quality. For our purposes, the overlay grid algorithm we start from is based on Sandia's Sculpt [2] algorithm.

1.1 Related works

The overlay grid methods [3, 4, 5] are all based on the same principle. Starting from a box B, which overlays the reference geometry Ω to be meshed, they discretize B using a size-adapted grid $\mathcal{G}$, then adapt $\mathcal{G}$ both topologically and geometrically. Geometric adaptation mainly consists in projecting some nodes of $\mathcal{G}$ onto $\partial\Omega$, while topologic adaptation consists in applying refinement patterns in order to adapt the grid resolution to some local features of Ω (like a hole in a surface, a small detail compared to the overall geometry). The mesh to be deformed is then no longer a grid but an unstructured hexahedral mesh.

One of the main differences of [2] is to take as an input some volume fraction defined on $\mathcal{G}$. The reference geometry is then implicit and it can be used in our context to generate a pure Lagrangian mesh from a multi-material Eulerian grid. Some other works [6, 7] share this type of inputs. In [7], the authors extended their interface reconstruction method [6] by iteratively moving the obtained interfaces, represented by triangular surface meshes in 3D, combining a Laplacian smoothing and a volume control contribution. Their method is dedicated to visualization purposes, and so one of their concerns is to obtain "good quality" triangles; they can also adapt the surface meshes, depending for example on a triangle edge length criteria threshold.

As our reference geometry is implicitly defined, interface reconstruction methods are relevant for us. Volume of fluids method [8], may be considered as related to our work. Starting from volume fractions living on any unstructured meshes (like an Eulerian grid for instance), those methods try to build interfaces with a strict volume preservation per cell. They do precisely control for volume inside each cell of the input. However, rather than producing pure computational elements, they can yield mixed elements in the output mesh where local interfaces are defined by discrete planar geometry. Such interfaces are globally non-conformal and aside from their usefulness for the simulation codes can only be used for simple visualization process or for initializing other numerical codes. On the contrary, overlay grid algorithms, such as Sculpt algorithm, require to get a smooth interface that approximates the interface surface. This smooth interface is then used to define a geometric model that can be a support for hex meshing.

1.2 Main contributions

Considering the "Euler to Lagrange" context, we propose a complete pipeline of steps to apply, where we try and control the mesh quality as much as possible at each step. This quality control is one of the main differences in the global process since most of the time mesh quality is ensured by post-processing where some mesh smoothing and untangling methods [9] are applied at the end of the pipeline, but these sometimes fail to improve the mesh quality.

Other contributions of our work are technical improvements on three steps of the pipeline. An overview of this pipeline is given in Section 2. The three improved steps are described in details from Sections 3 to 5. In Section 3, we provide a new assignment correction process to avoid non-manifold topological configurations. In Section 4, we propose a new interface reconstruction algorithm. Compared to [2], it allows us to build an independent geometry model that can be used afterwards for performing node relocation and projection. This process mimics techniques described in [6, 7] but

with the capability to better preserve volume fractions when $\mathcal{G}$ is an unstructured mesh. Section 5 introduces an adaptation loop that interleaves topological and geometrical operations in order to perform the grid movement. The main originality of this step is to define and use local pillowing operations to best control mesh modifications.

2. OVERVIEW

The method we propose adapts the pipeline defined in [2], where the initial input is n-dimensional grid $\mathcal{G}$, with $n = 2$ or 3. Each cell of $\mathcal{G}$ is a quadrilateral cell in 2D and a hexahedron in 3D. We have the same context in this paper where every step of the algorithm can be applied to any unstructured quadrilateral or hexahedral mesh as an input without any algorithm and code modifications. As the input grid $\mathcal{G}$ is the output of an Eulerian simulation code, each cell $c \in \mathcal{G}$ can contain several materials. Let $\mathcal{M}$ be the set of materials defined on $\mathcal{G}$. Then, we also have as an input a family of *material assignment functions* $ma_{\mathcal{G},m}$, simply noted ma_m in the following, such that $ma_m : \mathcal{G} \to [0, 1]$ and

$$\sum_{m \in \mathcal{M}} ma_m(c) = 1, \forall c \in \mathcal{G}. \quad (1)$$

A cell is said to be *pure* if it contains only one material and *mixed* otherwise. The material assignment function gives the volume fraction of materials of $\mathcal{M}$ in each cell of $\mathcal{G}$. For the sake of simplicity, we note ma the family of material assignment functions, and we will call it the material assignment ma.

Let us now give a brief overview of the approach considering the 2D grid given in Figure 2-a as an input. We have $|\mathcal{M}| = 3$ and the volume fraction of each material is indicated in each cell.

1. **Cell material assignment.** The first step of the algorithm consists in assigning each cell of $\mathcal{G}$ to a single material of $\mathcal{M}$ (see Section 3.1). This is done by: First assigning a material to each cell c considering only the volume fractions of materials in c (see Figure 2-b); Secondly, correcting wrong topological configurations in the vicinity of every grid node (see Figure 2-c and Section 3.2.). Each cell of $\mathcal{G}$ is virtually assigned to a single material and we have a set of interfaces between cells virtually assigned to different materials. We call those *virtual grid interfaces*.

2. **Material interface reconstruction.** In every mixed cell of $\mathcal{G}$ containing materials $\{m_1, \ldots, m_p\}$, with $p > 1$, we locally rebuild geometric interfaces between all those materials (see Section 4). This set of geometric interfaces will help us modify the grid $\mathcal{G}$ in Section 5.3.

3. **Mesh adaptation.** The final stage of the approach consists in modifying $\mathcal{G}$ in such a manner that the virtual grid interfaces fit geometric interfaces as much as possible while controlling grid cell quality. It relies on three main operations that we interleave in our process (see Section 5):

   - First, we can move the nodes lying on a virtual grid interface towards the corresponding geometric interface (see Figure 2-d).
   - Secondly, we can change the grid topology by performing pillowing[1] [10] operations. It can be done in a traditional manner on each set of grid cells virtually assigned to the same material (see Figure 2-e) or by using local pillowing techniques (see Section 5).
   - Thirdly, we can smooth all the grid nodes: inside each material and along the virtual grid interfaces (see Figure 2-d).

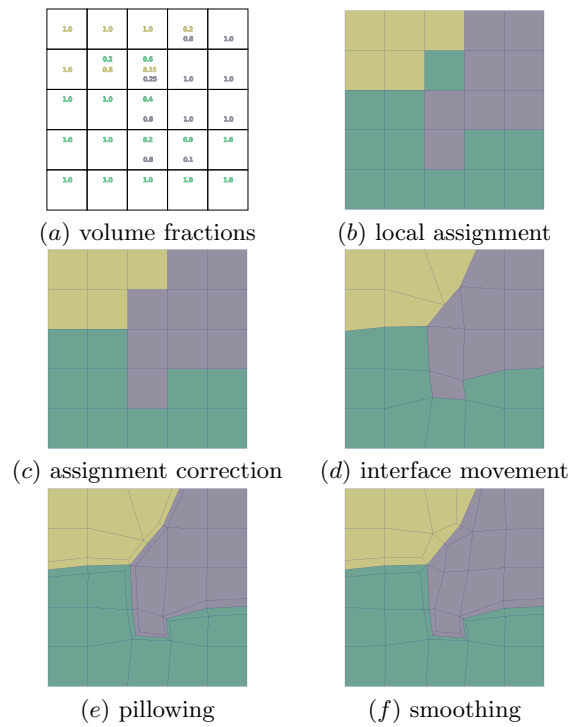


Figure 2: Overview of some operations involved in the proposed pipeline. Starting from the volume fractions given in (a), grid cells are virtually assigned to each material in two stages (b and c) before being modified geometrically (d and f) and topologically (e).

[1]The pillow operation consists in isolating a connex set of hexahedral cells S_H by inserting a layer of hexahedral cells around it. Each cell of this layer can be created by inflating each quadrilateral cell of ∂S_H into a single hexahedron.

3. CELL MATERIAL ASSIGNMENT

The first stage of the pipeline consists in defining a new assignment to each cell of $\mathcal{G}$, where each cell is pure. This assignment is said *virtual* and is denoted va. The term virtual is used in opposition to the material assignment fraction, which is the reference assignment given as an input of the pipeline. As previously said, the material assignment is initialized locally at each cell then upated to resolve wrong topological configurations.

3.1 Initialization via a local assignment

In order to initalize the virtual assignment in each cell $c \in \mathcal{G}$, we simply select the material having the highest volume fraction in c; in case of equality the first material examined is selected. More formally,

$$\forall c \in \mathcal{G},\ va(c) = m \ with\ ma_m(c) = \max_{p \in \mathcal{M}} ma_p(c). \quad (2)$$

It gives us the assignment as depicted in Figure 2-b from the material assignment shown on Figure 2-a.

3.2 Assignment correction

It can be mandatory for the simulation codes or for mesh generation algorithms that each material be an assembly of disconnected manifold (when they need for instance to compute the normal to a material at all nodes of its interface) or for practical purposes (ease of implementation).

Informally, in 2D, it means that a material is represented as a set of surface patches that do not overlap one another and do not touch along a single point. A single patch cannot also touch itself at a point (see Figure 3 on the left). Topologically, it corresponds to the fact that, in each point of a d-dimensional patch, one can define a single d-dimensional ball.

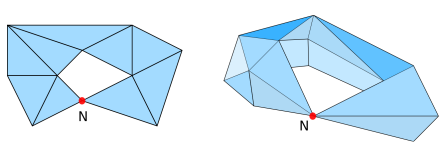


Figure 3: Illustration of non-manifold simplicial patches in 2D (left) and 3D (right). In both cases, the red node N induces that the patch is not a manifold.

In order to detect such situations for a material m at a node n, we build a polyhedron made of the cells adjacent to n assigned to m and we use the Euler-Poincaré relation for convex polyhedra. Considering a polyhedron having N nodes, E edges and F faces, we have the formula

$$N - E + F = 2. \quad (3)$$

Let us apply this formula on a single hexahedral element made of 6 faces, 12 edges and 8 nodes, we have $N - E + F = 8 - 12 + 6 = 2$ and relation 3 holds. Let us now consider the polyhedron build from the two cells shown on Figure 4-a. As they only share a node, we have $N = 15$, $E = 24$ and $F = 12$ and so $N - E + F = 3 \neq 2$. Relation 3 does not hold and the set of cells C_{n,m_i} forms a non-manifold around point n.

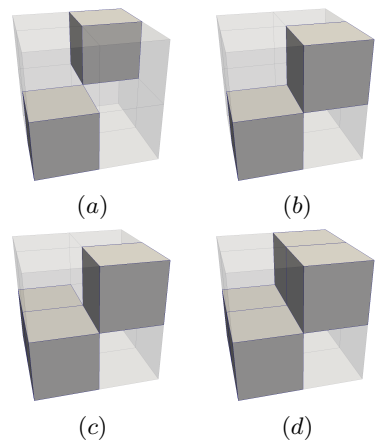


Figure 4: Example of non-manifold configurations for a material (non-transparent cells) around a node.

At the end of the mesh traversal, we have for each material a set of nodes indicating where non-manifold configurations occur. It is the input of the resolution procedure detailed by Algorithm 1. The resolution consists in modifying the virtual assignment initially done around the vicinity of each collected node. Note that solving the non-manifold issue locally to the vicinity of a node can lead to making non-manifold configurations appear at neighbouring nodes, which were originally not problematic (see for instance Figure 5-b). Hence, our resolution process is an iterative procedure where new nodes can be added to the set. As a consequence, we also must prevent a node to be treated several times leading to a livelock situation. We prevent such a situation from occuring by keeping track of local virtual assignments already performed around a node. If we treat a node again, we select a valid virtual assignment which was not already applied (see Figures 5-c and d). We iterate until there are no remaining non-manifolds.

Let us dive into the procedure we apply locally to every non-manifold node. This procedure is given in Algorithm 2. Let n be a node where a non-manifold configuration occurs for adjacent cells $\mathcal{B}_n$. We get all

Algorithm 1: Full-mesh non-manifold resolution.

Data: mesh $\mathcal{G}$, material assignment ma
Result: virtual assignment va with only manifold configurations

```
1  va ← ma
2  N_nm ← getNonManifoldNodes(G, va)
3  validConfigs ← ∅
4  while N_nm ≠ ∅ do
5      for n ∈ N_nm do
6          if validConfigs(n) = ∅ then
7              /* this node was never treated */
8              /* call to Algorithm 2 */
9              validConfigs.add(n,
                solveManifoldnessAtNode(n, va))
10         end
11         /* we try to get an unused stored
            combination */
12         c ← validConfigs.getConfig(n)
13         if c == NULL then
14             /* the number of solutions stored per
                node was insufficient */
15             return FAIL
16         else
17             validConfigs.remove(n, c)
18             va.reassign(c)
19         end
20     end
21     N_nm ← getNonManifoldNodes(G, va)
22 end
```

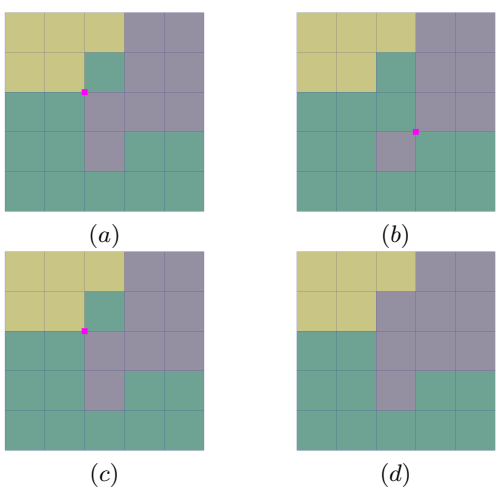


Figure 5: Example of non-manifold resolution and live-lock avoidance strategy. (a) the initial material assignment with the non-manifold node highlighted; (b) the new assignment (at Algo 1:7) solves the non-manifold at the first node but causes another to appear; (c) the new non-manifold node is solved but we are back to the initial assignment; (d) the non-manifold is solved again with another assignment (at Algo 1:10), thus avoiding an infinite loop between state (a) and (b).

the possible assignments for any cell of $\mathcal{B}_n$ using the material assignment (line 2). They are gathered in the set $\mathcal{M}_{possible}$. We then build the set that gathers all the possible configurations of virtual assignment where each cell of $\mathcal{B}_n$ can take any value in $\mathcal{M}_{possible}$ (line 3). In the loop (lines 4 to 16), we test those possible configurations and only keep the acceptable ones, i.e. configurations where we don't have non-manifold situations; we order them by *cost* and only keep a limited number of those (lines 14 and 15). The "best" configuration is the one that minimizes:

$$cost = \sum_{c \in \mathcal{B}_n} |1 - ma_{va(c)}(c)| Vol(c). \tag{4}$$

We have the following remarks about our procedure:

- The number of possible configurations is equal to $|\mathcal{M}_{possible}|^{|\mathcal{B}_n|}$. This number can far exceed the current hardware capabilities, so a limit to the potential materials $|\mathcal{M}_{possible}|$ is enforced, such as keeping only the four or five most preponderant materials that are in $\mathcal{B}_n$, based on material assignment;
- We keep the twenty (user's choice) best valid virtual assignments. Since we test every possible assignment and a node can be treated several times, we store the solutions when first treating the node and provide the next best not already used assignment (we allow the initial assignment to be tried twice) when asked again. This allows us to avoid infinite cycles, but if the upper limit is reached our code stops and the user has to rerun using a higher limit.

Note that additional assignment modifications can be dictated by other criterias, such as the expected final mesh quality as seen in [11] where cells forming isolated or thin chunks of one material are reassociated because these configurations usually lead to having bad quality cells, or sometimes the simulation code cannot handle small slivers of materials.

4. INTERFACE RECONSTRUCTION

This section focuses on building interfaces between materials that could help in the computation of a prospective new position (currently done similarly to [2]) for the interface nodes (the nodes adjacent to cells assigned to different materials) . Additionally, extracting such interfaces will also bring the added benefit of producing a geometric model.

Material interface extraction poses two main issues: the desired interfaces should be smooth, typically when the goal is to visualize them, but at the same time they should also fit the input data as best as

Algorithm 2: Local non-manifold resolution.

Data: node n, mesh M, material assignment a, volume fraction VF, materials $\mathcal{M}$
Result: stored valid configurations in $\mathcal{B}_n$

```
B ← M.adjacentCells(n)
M_possible ← {m ∈ M / ∃c ∈ B with a_m(c) > 0 }
possibleConfigurations ←build(M_possible)
for C_i ∈ possibleConfigurations do
    isValid ← true
    for m ∈ C_i do
        if .not. isManifold(m) then
            /* cells of m form a non-manifold */
            isValid ← false
            break
        end
    end
    if isValid then
        cost ← computeCost(C_i, ...)
        store(n, cost, C_i)  /* we store only the 20
          best solutions */
    end
end
```

possible, namely the volume fractions; those two objectives can conflict with one another. Several methods exist, in different fields:

- ALE-simulation interface reconstruction.
  A domain where material interface reconstruction is extensively studied and applied is CFD simulations. While those reconstructed interfaces have a built-in volume fractions preservation, they are not smooth, as they are discontinuous across cells [8]. Most of these methods also have the additional drawback of being material order-dependent;

- Sculpt
  The position computed for the interface nodes in [2], and that we use in our implementation, depends on the volume fractions (and their gradient) and the virtual assignment. We have new positions for the nodes, but no geometric model;

- voxels
  These methods decompose the mixed cells into sub-elements – typically a hexahedron will be refined into a grid – on which a partitioning is called with respect to the volume fractions inside each cell. The interfaces at the sub-elements level are aliased, and since these methods originated from visualization purposes the interfaces are usually simplified into smooth triangular surfaces. We will further study voxel-like methods in this section.

4.1 problem under voxel terminology

The discrete interface reconstruction techniques stems from the need to visualize the location of materials in the case where some of the cells are mixed and where the number of materials is greater than two. In the case where the number of materials equals two, classic iso-contouring methods provide an adequate solution but with more materials small gaps or artifacts can appear that are non-desirable. In [12] the authors introduced the decomposition (or refinement) of mixed cells into subcells (or pixels) which are in turn assigned to the materials present in the mixed cells they were spawned from; the work in [6, 7] extends it to cases with more than three materials per cell.

The voxel problem is the following: all the mixed cells and their adjacent pure cells are subdivided into pixels, and we want a virtual assignment on those pixels. Pixels spawned from pure cells are already assigned to the material of their corresponding pure coarse cell, leaving those spawned from mixed coarse cells as "free". We favor a pixel virtual assignment that has a low

- *edgecut*, defined as the sum of pairs of adjacent pixels assigned to different materials;
- *discrepancy*, defined as the sum over each coarse cell cc of the absolute difference between the volume of each material present in cc (for material k it is $ma_k(cc)Vol(cc)$) and the volume of the pixels of cc assigned to k. It expresses whether the pixels assignment fit the volume fractions.

We compare four methods that solve the problem:

- mixed-integer linear programming
- simulated annealing
- graphcut
- greedy heuristic

4.1.1 MIP

We formulate the problem as a linear problem; since our variables are integers we in fact have a mixed-integer linear problem. In our implementation, we use the GLPK [13] library to solve the problem.

Problem formulation:

$$\left\{ \begin{array}{ll} min \sum\limits_{p \in P, k \in Mat} |c_{p,k} - \frac{1}{|N(p)|} \sum\limits_{q \in N(p)} c_{q,k}| & \\ \text{constrained to} & \\ c_{p,k} \in \{0,1\} & \forall p,k \\ \sum\limits_{k \in Mat} c_{p,k} = 1 & \forall p \\ \sum\limits_{p \in cc} c_{p,k} = nbsub * ma_k(cc) & \forall k, \forall cc \end{array} \right\}$$

where $c_{p,k}$ returns 1 if pixel p is assigned to material k, 0 otherwise (using the previous notations, it gives $va(p) = k$). The first two constraints indicate that every pixel has an assignment and only one. The third constraint expresses that we want to have a *discrepancy* equal to zero (*nbsub* being the number of pixels in a coarse cell). The objective function that we want to minimize reflects the desire for pixels assigned to the same material to be clustered together, i.e. having a low *edgecut*.

4.1.2 Simulated Annealing

This method introduced in [6] consists in randomly assigning the pixels to materials with respect to the volume fractions data; pair of pixels spawned from the same coarse cell will then swap their virtual assignment, as described in Algo 3. Since the initial assignment to the pixels fit as best as possible the volume fractions and that only swaps are performed, the resulting pixels virtual assignment fits just the same. Note that this property does not hold true when the mesh is unstructured; more precisely when the pixels are of different size (see Figure 8).

4.1.3 Graphcut

Our problem can be seen as an energy minimization problem; in [14, 15, 16] it was shown that expressing the energy function as the sum of a so-called data and smooth costs allows the problem to be solved using a graphcut.

Problem formulation, with f_p being the virtual assignment of the pixel p and E the pairs of adjacent pixels:

$$E(f) = \lambda \sum_{p \in P} D_p(f_p) + \beta \sum_{\{p,q\} \in E} V_{p,q}(f_p, f_q)$$

with basically the edgecut:

$$V_{p,q}(f_p, f_q) = \begin{cases} 0. & \text{if } f_p = f_q \\ 1. & \text{otherwise} \end{cases}$$

and the cost of assigning a material to a free pixel (nearest is the nearest pixel issued from a pure cell assigned to f_p):

$$D_p(f_p) = (1. - ma_{f_p}(p)) p.distance(nearest(f_p))$$

We should note that the first term does not enforce matching the volume fractions. In order to apply the graphcut this term has to be dependent on only p; the expression that we chose tries to emulate that property but we will see in the results (Figure 7-c) that it is far from being efficient. The second issue comes from the energy function itself that is the sum of two terms not related to one another, and the values chosen for (λ, β) might depend on the cases we run on.

Algorithm 3: SubPixels assignment via simulated annealing (as implemented in Visit 2.13.3).

```
Data: volume fraction VF, pixelated mesh
Result: subpixels assignment
/* temperature stays constant but it could decrease over time */
T ← 0.25
while time ≤ allotedtime do
    iter ← 0
    for iter ≤ 1000 do
        /* randomly select a mixed (coarse) cell */
        c ← getMixedCell()
        /* randomly select a pair of pixels of c assigned to different materials; give up after ten tries */
        p0, p1 ← c.getSwapCandidates()
        l0 ← label(p0)
        l1 ← label(p1)
        /* evaluate the energy with the current labeling and the prospective one where the labels are swapped */
        current_energy ← energy(p0, l0) + energy(p1, l1)
        future_energy ← energy(p0, l1) + energy(p1, l0)
        if future_energy < current_energy then
            swap(p0,p1,l0,l1)
        else
            ΔE ← |future_energy − current_energy|
            if rand(0, 1) < e^(−ΔE/T) then
                /* randomly swap anyway depending on the temperature */
                swap(p0,p1,l0,l1)
            else
                if future_energy = current_energy and rand(0, 1) < 0.5 then
                    /* when equal randomly swap */
                    swap(p0,p1,l0,l1)
                end
            end
        end
        iter++
    end
end
```

4.1.4 Greedy Heuristic

We have implemented a greedy heuristic (see Algo. 4) where at each iteration the free pixels are assigned volume fractions that depend on the values in their respective coarse cells adjusted to take into account the pixels that were already assigned (see Figure 6). 3D results are shown in Figure 9

Algorithm 4: SubPixels assignment greedy heuristic.

Data: volume fraction VF, pixelated mesh
Result: subpixels assignment

```
threshold ← 1.
freeSubpixels ← allsubpixels
fixedSubpixels ← ∅
vf ← (VF, freeSubpixels)
for freeSubpixels ≠ ∅ do
    /* get the free pixels with a vf higher than the threshold for one material */
    fixedPixelsToAdd ← extractPixelsBelow(freeSubpixels, threshold)
    fix(fixedPixelsToAdd)
    if fixedPixelsToAdd ≠ ∅ then
        reduce threshold
    end
    /* update the vf while substracting the pixels already assigned */
    vf ← update(VF, freeSubpixels)
    for iter ≤ maxNbIter||convergence do
        /* kind of a vf smoothing */
        vf ← average(vf) for pixels where ¡ threshold
        normalize(vf)
    end
end
```

4.2 results

The results of the four methods can be seen applied on the 5x5 example in Figure 7. The MIP implementation is impractical, as it does not return a solution in an acceptable time; it can sometimes return a valid (meaning that it fits the constraints) but not optimal solution, which is the case in Figure 7-*a*. The graphcut tends to return straight interfaces, resulting in a good edgecut, but as we have mentionned is quite bad when considering the *discrepancy*. That leaves us with the simulated annealing, which is better than our greedy heuristic in the case of a grid, but fares badly concerning the *discrepancy* in unstructured cases, as shown in Figure 8. All of those methods have the same memory limitation, as the submesh can be quite big. In practice, we will use our heuristic to build the pixelated interfaces, as it is a good compromise in structured and unstructured cases and does not rely on tuning parameters depending on the case.

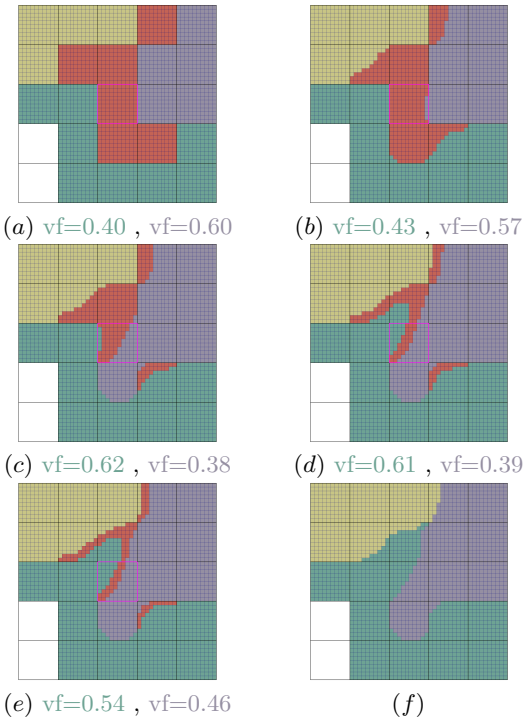


Figure 6: Greedy heuristic applied to the 5x5 example where we can see the evolution of the volume fractions (see Algo. 4:13) assigned to the free pixels of the central coarse cell below each figure. The wireframe black grid is the coarse mesh and the pixels colored in red are those not yet assigned.

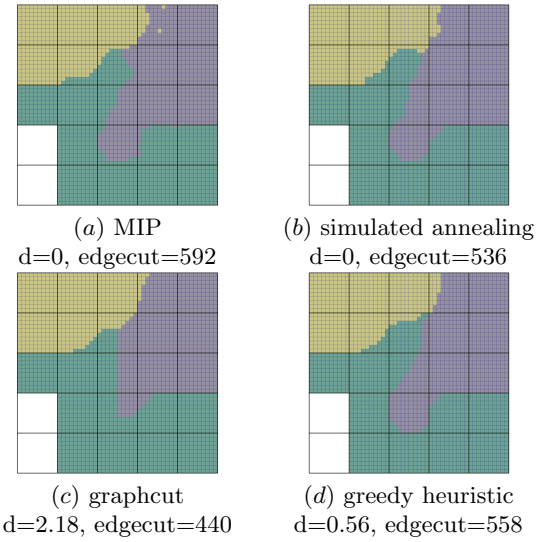
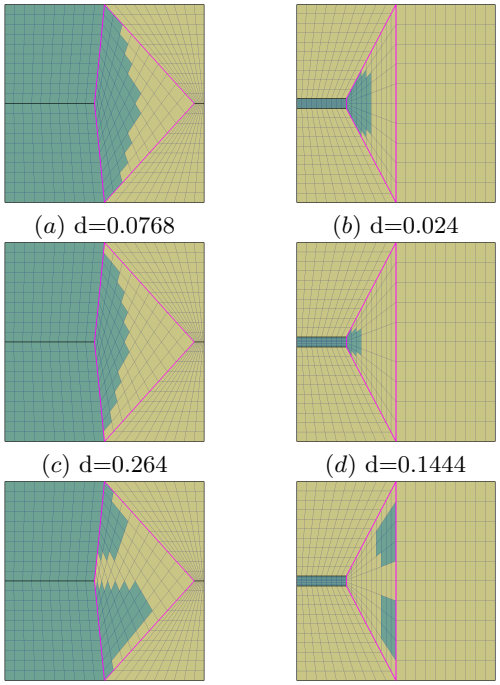


Figure 7: Comparison of the pixelated interfaces reconstruction methods. Note that we stopped the MIP implementation after 5 minutes.

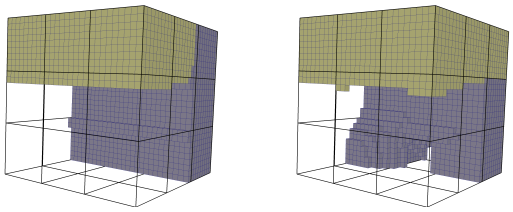


(a) d=0.0768 (b) d=0.024

(c) d=0.264 (d) d=0.1444

(e) d=0.1344 (f) d=0.1254

Figure 8: Greedy heuristic (first row) versus simulated annealing (second and third rows) applied to unstructured cases. Only the highlighted cell is mixed, and the respective volume fractions are (0.5,0.5) in the left, (0.2,0.8) in the right. Two different results are shown for the simulated annealing method because some cluster of pixels can appear due to the randomness of the initial pixel assignment and the swaps.

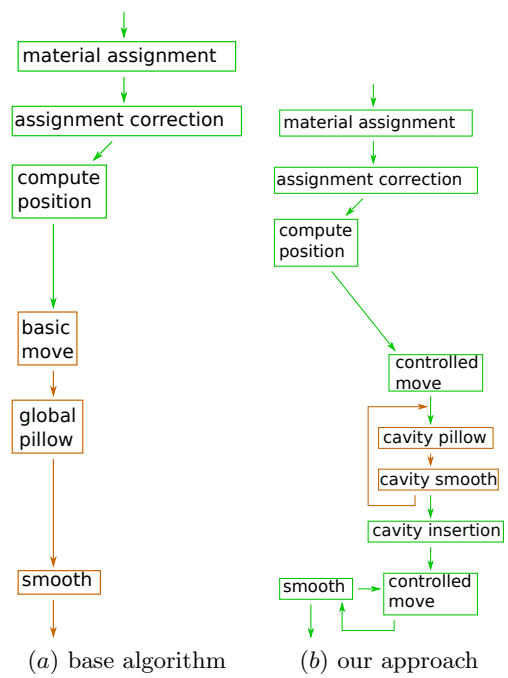


(a) (b)

Figure 9: Greedy heuristic applied on two 3D cases with three materials. The third material is hidden. (a) an extruded case; (b) a "real" 3D case.

5. MESH ADAPTATION

Up to now, the geometry and the topology of the grid $\mathcal{G}$ have not been modified. The aim of this section is to do so while offering some quality guarantees about the output mesh. From now on, we consider that the user provides a cell quality threshold $S_q > 0$[2] in addition to the input mesh $\mathcal{G}$ carrying volume fractions data materialized by the material assignment functions ma. Assuming that all cells of $\mathcal{G}$ are initially above the user-input threshold S_q, $\mathcal{G}$ keeps satisfying the quality requirements at the beginning of this stage of the process. It is the operations that we are going to perform now that have an impact on the mesh geometry and/or its topology and where there is no longer any guarantee on cell quality. Those operations are: node movement, pillowing and smoothing. As a reminder, our starting point is the pipeline of operations given in [2, 11] and illustrated on Figure 10-a.



(a) base algorithm (b) our approach

Figure 10: Base algorithm diagram versus our modifications. In green the part where we have a guaranteed cell quality, and in orange the part where there is no control over this.

In this initial pipeline, after computing ideal locations for the nodes of $\mathcal{G}$ that are on virtual grid interfaces, we move the virtual grid interface nodes to these new locations in order to better capture the material interfaces. Such a direct movement tends to decrease the cells quality. Then a pillowing phase is applied,

[2]Typically, it can be a minimum scaled Jacobian [17] value below which no cell must be.

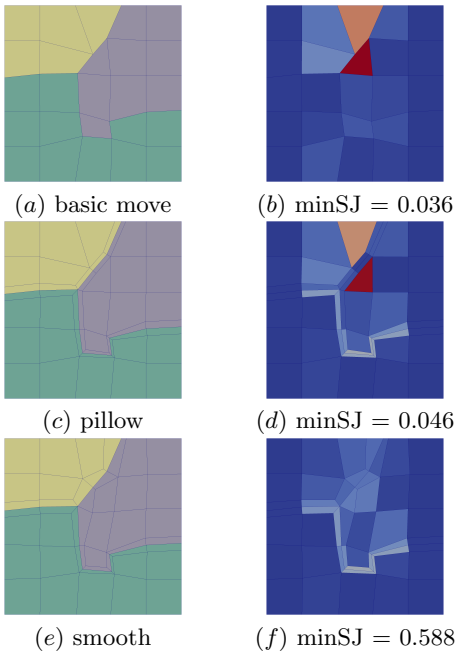


Figure 11: Evolution of the cell quality during the base algorithm. In (a) and (b) the node movement causes a sharp decrease of the cell quality; in (c) and (d) the pillowing does not really improve the cell quality; in (e) and (f) the smoothing is efficient in this example.

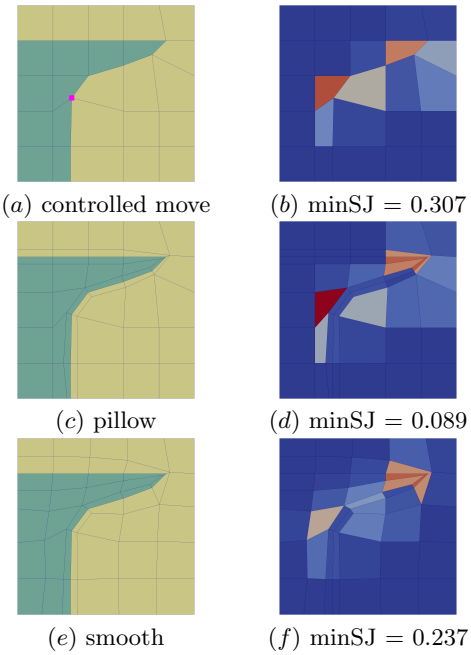


Figure 12: Global pillowing quality problem on a 5x5 two materials case. (a) and (b) mesh after the controlled move with the marked node stopped at a distance of 0.04 from its desired position; (c) and (d) global pillowing; (e) and (f) the smoothing makes good use of the additional edge added to the marked node in the green material area to improve quality but cannot improve it at the tip.

where each material is wholly pillowed without taking cell quality into account, and eventually a final global smoothing stage is executed so as to improve the overall cell quality. The intermediate pillowing phase does not in itself improves quality, but it provides more degrees of freedom for the smoothing algorithm to work with. The example of Figure 11 illustrates that such a pipeline can lead to good results. While the cell quality is not strongly controlled, the mesh quality is improved during the process.

The example of Figure 12 shows the exact opposite. The mesh quality worsens because of a global pillowing technique that does not take into account some local geometric features. In this case, performing a pillowing around the right tip of the green area leads to strongly decreasing the inner angle of each quadrilateral cell around this node. Note that the worst cell after proceeding with the pillowing (in red in Figure 12-d) was the one hampering the displacement of the marked node. The subsequent smoothing applied does not improve the situation and the quality of the worst cell is eventually lower than at the beginning (0.237 agains 0.307).

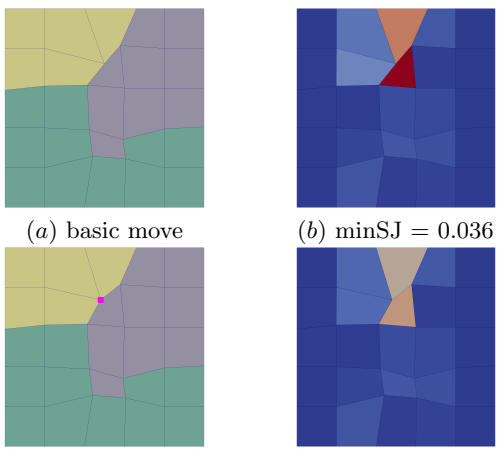
We consider that this process has two main drawbacks. First, node relocation is done without considering the quality of surrounding cells. We propose to control this node movement and to forbid it if the cells quality get lower than S_q. Secondly, we propose to adopt a local pillowing strategy instead of a global one. Eventually we interleave those operations with a traditional smoothing step in our process (see Figure 10-b). It allows us to preserve quality above the threshold at all times in the new pipeline, which includes:

- a controlled node movement strategy;
- a localized cavity pillowing in place of a global indiscriminate one;
- a node position computation based on a geometric model extraction - the one given by the interface reconstruction in Section 4 - in order to smooth the target material interfaces, particularly necessary in 3D.

In the subsequent examples used for illustration purposes, the threshold chosen for the minimum scaled Jacobian will always be $S_q = 0.3$.

5.1 Controlled node movement

As seen in Figure 10 and 11-b, the first phase that may decrease quality is the basic node movement. Considering that the input mesh meets the quality requirements, our strategy is to avoid moving the nodes when the quality is degraded below the threshold. For that purpose, we progressively move each node n of the virtual grid interfaces towards their expected location $\mathbf{p}_{ideal}$ on the geometric interface. At each small movement of n, we check the quality of cells surrounding n. In our implementation, a small movement corresponds to $1/16$ of the distance between the location of n and $\mathbf{p}_{ideal}$. The impact of this controlled movement can be seen in Figure 13, and the guarantee over the quality, represented in green in Figure 10-b, extends and reaches just before the pillowing stage.



(a) basic move (b) minSJ = 0.036

(c) controlled move (d) minSJ = 0.387

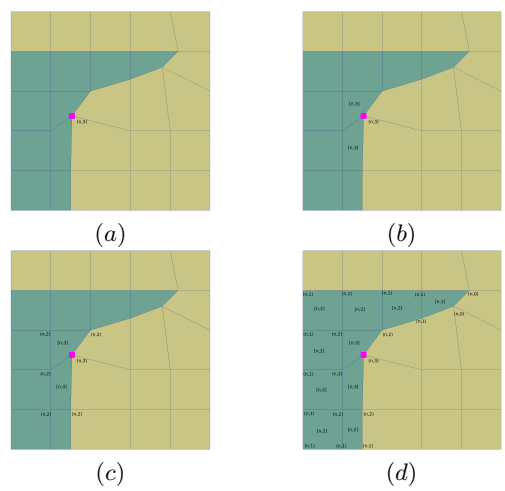
Figure 13: Example of controlled movement. (a) and (b) basic move and the mesh quality; (c) and (d) controlled move, the marked node could not move all the way and was stopped at a distance of 0.118 from its expected location and the cell quality remains above 0.3.

A direct side effect of this controlled movement is that some of the nodes did not reach their expected position. Let $\mathcal{N}$ be those nodes, we introduce an additional modification to the base algorithm to allow for those nodes to move further. After applying the topological modifications (the pillowing), the algorithm enters into a move-smooth loop (see Figure 10-b) so that the nodes of $\mathcal{N}$ can progressively keep moving towards the geometrical interfaces, eventually reaching it in the best cases. In this move-smooth loop, the smoothing stage has two prerequisites: it must not decrease the quality and nodes that have reached the geometric interfaces are fixed. At the end of this stage, the set $\mathcal{N}$ is not necessarily empty.

5.2 Cavity pillowing

As shown in Figure 12 , performing a global pillowing can lead to drastically decrease the mesh quality without any guarantee of successfully improving it during the final smoothing stage. In our pipeline, the pillowing is performed in a totally different manner. It aims to help moving nodes of $\mathcal{N}$ by providing more leeway for the smoothing algorithm to work with. The idea is then to apply pillowing operations in the vicinity of $\mathcal{N}$ while avoiding to change the mesh topology where the quality is already good.

The process we follow can be summarized as follows: for each node n of $\mathcal{N}$ and each material m adjacent to n, we extract cell groups that we are going to pillow. Each cell group is called a cavity. In theory, the idea would be to build many sets of cavities $\{c_i\}_{i>0}$ for (n, m), pillow and smooth each cavity c_i independently and keep the one that gives the best quality. If this quality is higher than the quality threshold S_q then the pillowing of this "best" quality is published in the mesh.



(a) (b)

(c) (d)

Figure 14: Cavity definition for the green material. (a) the node n that could not reach its destination is marked and assigned a range (here it is 3); (b) the adjacent cells are also selected and a reference to n is kept; (c) the selection is extended to the adjacent nodes, a reference to n is kept and the range is decreased; (d) we continue until the range reaches zero or all the cells assigned to the material are selected.

As there is a huge number of potential cavities, we consider in practice a maximal distance to the node n to build cavities. This distance corresponds to a vertex-based traversal of the mesh starting from n. For instance, for distance 1, the cavity of (n, m) contains all the cells of material m that are adjacent to n. In all our examples, a maximal distance of 3 is chosen. As an illustration, Figure 14 shows our process

to create cavities when $\mathcal{N}$ is reduced to a single node. Another implementation choice is to have a greedy approach where we do not perform the pilllowing for all the cavities. For a couple (n, m), we publish the first one that provides a quality above S_q. This greedy algorithm is done by progressively decreasing the cavity distance. We made this choice because the cavities of several nodes of $\mathcal{N}$ are more likely to merge when they are big enough. Such merging reduces the number of topological changes. Figure 15 shows how we can end up with potentially several large-size cavities.

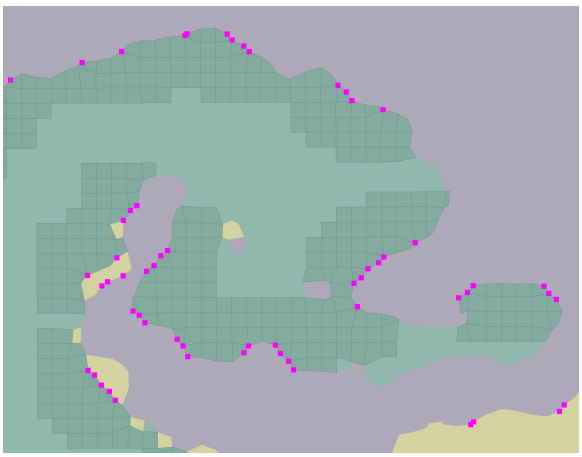


Figure 15: Cavity extracted for the green material in the triple point example (see Figure 19) where we can see the cavities spawned from different nodes merged to form bigger cavities. Note that marked nodes at the interface between the yellow and grey materials are ignored and do not spawn cavities, as we currently consider one material at a time.

As previously said, our greedy algorithm stops when we encounter a situation with a cavity quality that is above S_q. This is illustrated by Figures 16 and 17. In Figure 16, we first extract a cavity of range 3 for the marked node (in a) and then we pillow and smooth it (in b and c). As the quality is below the $0,3$ threshold, we decrease the cavity range to 2 and start a second iteration (in e, f, g and h) where we get the expected quality. In Figure 17, we see the local pillowing made on both materials adjacent to the marked node. Our overall algorithm is summarized by Algorithm 5).

5.3 Geometric model extraction, projection and smoothing

So far we have tried to move the nodes towards a computed location that depends solely on the input volume fractions and the cells virtual assignment (as computed in [2]). No care was taken for the expected interfaces quality. It becomes relevant in 3D where the mesh entities forming the interfaces are no longer

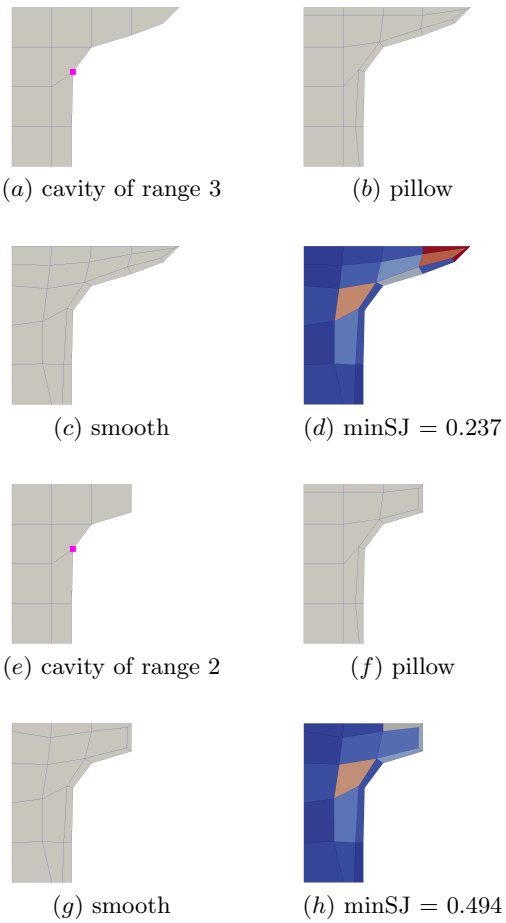


Figure 16: Cavity pillowing and smoothing loop. (a, b, c and d) A first iteration with a cavity of size 3 does not give the expected quality; (e, f, g) and h) a second iteration with size 2 meets the requirements.

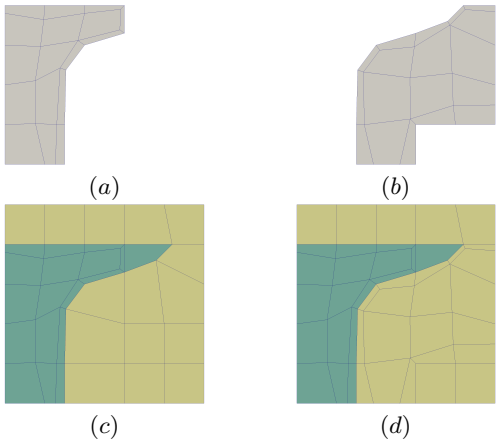


Figure 17: Cavity pillowing insertion back into the mesh. (a) and (c) the pillowed cavity for the green material and its insertion inside the mesh; (a) and (d) the same for the yellow material.

Algorithm 5: Cavity loop.

Data: marked Nodes $\mathcal{N}$
Result: mesh

```
for m ∈ Materials do
    /* we initially assign the same user-input range to all the marked nodes */
    R ← initRange()
    /* cavity extraction (see Figure 14) */
    cavity ← extractCavity(N, R, m)
    while cavity ≠ ∅ do
        /* we will apply modifications to the cavity and reduce it size until it meets the quality threshold (see Figure 16) */
        cavity.pillow()
        cavity.smooth()
        qual ← cavity.getQuality()
        if qual > threshold then
            /* the mesh modifications inside the cavity meet the quality requirements, we can insert it back into the mesh (see Figure 17) */
            mesh.insert(cavity)
            break
        end
        /* cells of the cavity keep a reference to the node that marked them, we decrease the range of such nodes for cells of quality below threshold */
        R ← updateRange()
        cavity ← extractCavity(N, R, m)
    end
end
```

1-cells (or edges) but are now 2-cells (or faces). Moving the nodes to their computed ideal location can by-design leads to bad quality faces, hence severely limiting our nodes controlled movement. Figure 18-a illustrates it in the asteroid case, the surface mesh we would obtain by moving the interface nodes at their expected locations has very bad-quality quadrilateral elements.

In order to counteract this we modify the computed nodes locations by applying a surface smoothing (see Figure 18-d). So as not to stray too far from the input volume fractions data and preserve the volume of the materials, the positions are constrained on a geometric model, which in our case is the model that can be extracted from interfaces reconstructed (see Figure 18-b) as seen in section 4. The resulting expected surface is shown in Figure 18-c. The benefits of this corrected new position computation appear in the asteroid 3D case where not applying this adjustment results in our algorithm being stuck after the first controlled move. Such a resulting mesh could be considered satisfactory, quality-wise, still we want the interface nodes to be located as much as possible close to where the material interfaces were determined to be.

6. RESULTS

In this section, we demonstrate and analyse the results of the proposed method on several cases, both 2D and 3D; the metrics on the results are shown in Table 1 where $dist_{init}$ and $dist_{final}$ are defined as the sum of the distances between the interface nodes and their expected position, respectively after the first controlled move and at the end of our algoritm. Admittedly those values heavily depend on the size of the grid, so comparisons between cases might not be relevant, but the ratio $\frac{dist_{final}}{dist_{init}}$ represents the improvements (the lower the better) our modified pipeline brings to counteract the drawback of the controlled node movement. We also compare our method with Sculpt on some examples regarding execution time and a *discrepancy* metric.

6.1 Results of our method

2D cases. We applied our approach in an intercode context, where our inputs are grid meshes carrying volume fractions data taken from a CFD simulation code at several time steps (see Figure 19) for two cases, the *triple point* and the *double bar* problems. They came from simulations run on grids of two different resolutions. The results in Table 1 highlight the motivations behind our approach: taking the triple point case at 1 second, we can see that for one grid resolution the base algorithm returns with a mesh containing no inverted cells (but still lower than the 0.3 minimum scaled ja-

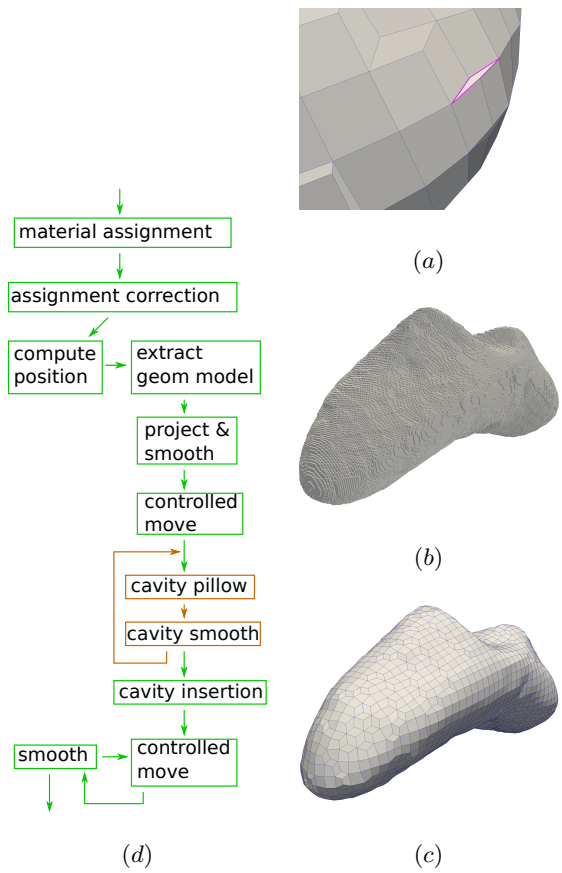


Figure 18: Motivation behind the adjustment to the computed node position, using the geometric model extraction and projection smoothing illustrated in the asteroid case in 3D. (a) close-up of the expected surface interface mesh where the marked quad has a quality of 0.068; (b) the pixelated interface between the asteroid and the exterior as obtained in section 4 and on which we will smooth the expected interfaces to remove problems such as those seen in (a); (c) the smooth and projected expected node position; (d) new pipeline that now includes the modifications to the computed positions.

cobian threshold chosen by the user). That is not the case for the other resolution, making it unreliable. It is unrealistic to ask users to rerun their simulations with different resolutions at random, assuming it is even feasible, hence the need for our algorithm that consistently works.

Our method was applied on two additional hydrodynamics simulations issued from [18] (see Figure 20); in all those cases it improves the distance by at least an order of magnitude.

3D extruded cases. The same cases from Figure 19 were extruded[3] and run in 3D.

While our method does indeed result in meshes meeting the quality requirements the ratio of $dist_{final}$ over $dist_{init}$ remains much higher than in the 2D cases.

"Real" 3D cases. Fully 3D cases were studied, one which input is a grid where the volume fractions data were computed by imprinting an asteroid model into the grid (see Figure 18) and the other one is an hydrodynamics simulation of a ball of fluid falling into a box computed using [19] and taken at several time steps (see Figure 21).

These cases make use of the improvements on the computed node position as described in Section 5.3 and shown in Figure 18.d. We should also note that in these cases, the first controlled move was executed with a threshold $S_q = 0.3$ which was decreased to 0.2 for the remainder of the algorithm. Without this intermediate threshold the cavity adaptation never manages to produce a good enough one to insert back into the mesh, which means our output mesh would have been the mesh obtained after the first controlled move. The distance $dist_{init}$ and $dist_{fina}$ are illustrated in Figure 22, and a success of the decreasing cavity-size strategy is shown in Figure 23.

6.2 Discrepancy and comparison

Neither Sculpt nor our method preserve the volume fractions; in [20] the authors show that it is not an issue, depending on the application. Nevertheless, we are interested in measuring by how much our output deviates from the input; in [21] we define a *per-cell discrepancy* criteria for c a cell of $\mathcal{G}$ as

$$d_c = \sum_{m \in \mathcal{M}} |V(c \cap M^O_{|m}) - ma_m(c)V(c)|. \qquad (5)$$

where $V(X)$ is the volume of any geometric space X and $M^O_{|m}$ the output mesh restricted to the pure cells

[3]The 3D mesh is created from a 2D quad mesh, lying in the XY plane, by creating successive layers of hexahedral cells along the Z direction. Volume fractions are simply derived for each hexahedral cell from their origin quadrilateral cell.

Table 1: Quality and distance metrics for the examples.

case name	$minJS$ *base algo*	$minJS$ *our algo*	$dist_{init}$	$dist_{final}$	$\frac{dist_{final}}{dist_{init}}$
2D					
triplepoint 1s 420x180	0.215	0.322	0.0676	0.0071	0.105
triplepoint 1s 518x222	-0.071	0.310	0.0856	0.0061	0.071
triplepoint 2s 420x180	-0.031	0.311	0.165	0.0138	0.084
triplepoint 2s 518x222	0.097	0.308	0.186	0.0163	0.088
doublebar 0.5s 200x100	0.074	0.306	0.5915	0.0163	0.027
doublebar 0.5s 214x107	0.091	0.301	0.5950	0.0121	0.020
doublebar 1s 200x100	-0.177	0.300	0.5768	0.0319	0.055
doublebar 1s 214x107	-0.109	0.301	0.6146	0.0411	0.067
hydro_toro_a	-0.104	0.300	116.70	6.0177	0.051
hydro_toro_b	-0.994	0.300	1902.3	104.34	0.055
3D					
triplepoint 1s 420x180x3	0.067	0.300	34.048	21.587	0.634
triplepoint 2s 420x180x3	-0.157	0.300	74.5847	44.388	0.595
doublebar 0.5s 200x100x3	0.043	0.300	134.47	26.025	0.193
doublebar 1s 200x100x3	-0.159	0.300	122.24	44.576	0.365
With our algorithm (Figure 18-d)					
asteroid	-0.13	0.200	319.874	31.148	0.097
balldrop_10		0.274	41.426	5.5029	0.133
balldrop_15		0.209	35.243	6.3432	0.18
balldrop_20		0.221	35.824	18.149	0.506
balldrop_25		0.200	75.346	39.089	0.519

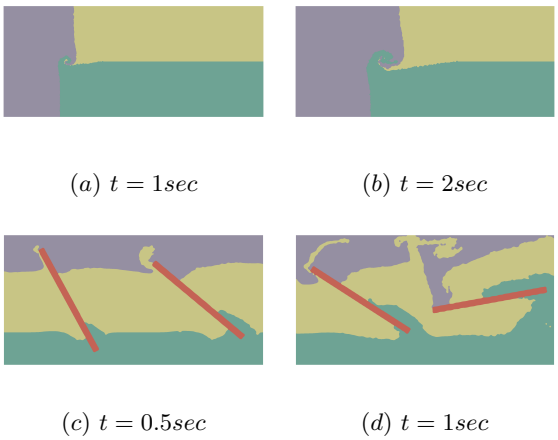


Figure 19: Examples of CFD simulations in 2D. Our algorithm was also applied to 3D cases extruded from the 2D. (a and b) triple point problem where three fluids of different densities lead to the formation of a vertex; (c and d) double bar problem where three fluids of different densities are stirred by two rotating blades.

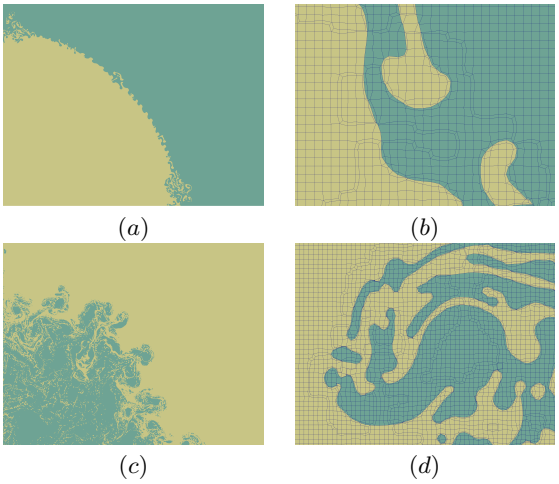
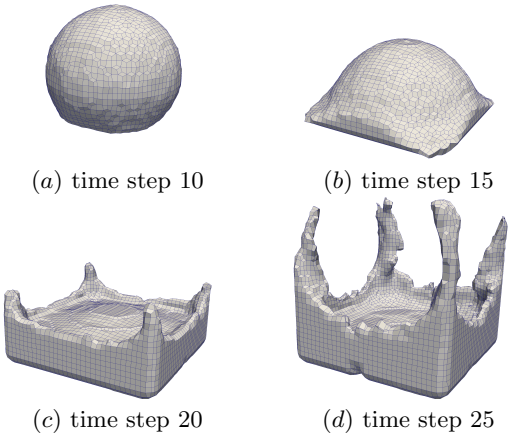


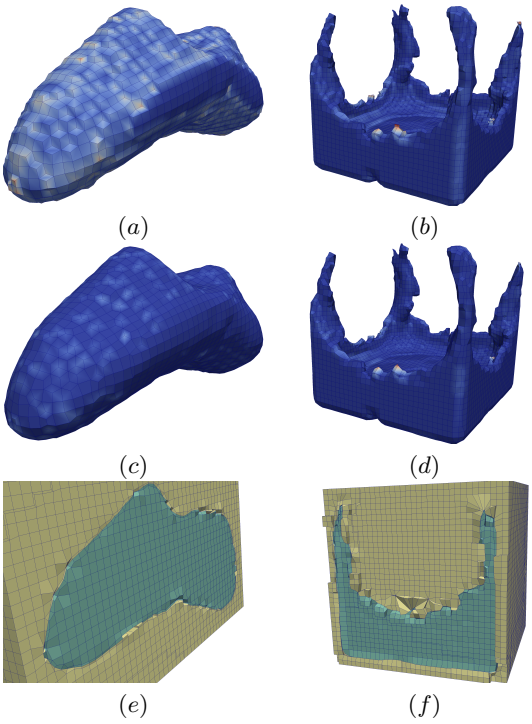
Figure 20: Other examples of hydrodynamics simulations in 2D [18]. (a) and (c) the two cases; (b) and (d) close-ups on our resulting meshes shown respectively.



(a) time step 10 (b) time step 15

(c) time step 20 (d) time step 25

Figure 21: Resulting mesh from our algorithm applied to the balldrop case.

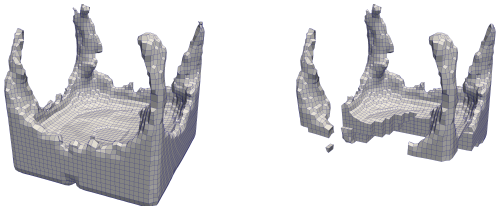


(a) (b)

(c) (d)

(e) (f)

Figure 22: 3D cases measuring the impact of our algorithm. (a and c) the distance in the asteroid case between the interface nodes after the first controlled move and their computed position, and the same after applying our whole algorithm; (b and d) the same for the balldrop case at step 25; (e and f) a clipped view of both cases in order to exhibit the exterior.



(a) $nbCells = 9548$ (b) $nbCells = 4115$

Figure 23: Cavity downsizing for the pillow on the balldrop case at step 25. (a) first cavity on which the pillow was tried for the fluid material, it is practically the whole fluid; (b) cavity where the pillow and smoothing phases managed to produce a submesh of acceptable quality that was inserted back into the mesh.

of material m. It basically expresses the difference between the input volume fractions and the ones computed by imprinting our output mesh onto the initial grid $\mathcal{G}$. The *discrepancy* is simply the sum of the term computed in eq 5 for all the cells of $\mathcal{G}$.

We compared our approach with Sculpt (using Cubit $v15.4b$) and Sculpt with an *active contouring* add-on developed in [21] aiming to reduce the *discrepancy*; the results can be seen in Table 2. We do not claim to be exhaustive in testing Sculpt, which has many options to drive the mesh generation.

We can see that our method fits better the volume fractions, even after applying the add-on. It could be explained by the fact that in order for Sculpt to return an output mesh with no negative scaled Jacobian elements in those cases the $defeaturing$ [11] option was activated, favoring mesh quality by sacrificing the volume fractions preservation (see the impact in Figure 24 where some small clusters of material were wiped out). Even with this option on, we were not able to obtain a mesh without inverted elements in the *double bar 1.0s* example. In Figure 25 we can see that an agressive smoothing makes the rotating bar loose its shape.

We can also see that our robustness comes at a price, our method being approximately twenty times slower (Sculpt was run on a single MPI process, and our algorithm on a single thread). In Table 2 is shown that for the *doublebar* cases, which have the same grid size, the execution time varies significantly: our method is not only sensitive to the grid size, but also to the carried data.

7. CONCLUSION AND FUTURE WORKS

In this work, we have improved on a straightforward overlay grid algorithm to make it compatible with the requirements of the "Euler to Lagrange" context

Table 2: Discrepancy and execution time comparison with Sculpt, Sculpt + add-on and our approach.

case name	metric	*Sculpt*	*Sculpt*+ add-on	*our algo*
triplepoint 1s 420x180x3	discrepancy	583.983	248.594	205.699
	time (s)	24.57		505.2
triplepoint 2s 420x180x3	discrepancy	1204.86	513.38	433.458
	time (s)	26.5		540.6
doublebar 0.5s 200x100x3	discrepancy	852.687	351.924	270.238
	time (s)	9.14		187.9
doublebar 1.0s 200x100x3	discrepancy	fail	fail	546.159
	time (s)	84		334.8

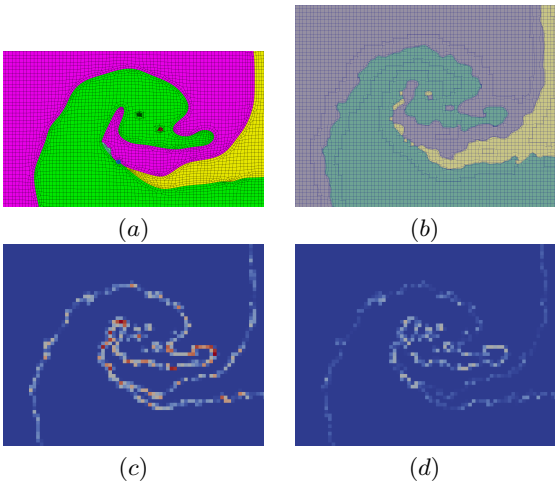


Figure 24: Comparison for the *triplepoint 2sec* case. (a) and (b) output meshes from Sculpt and our method; (c) and (d) their respective per-cell discrepancy.

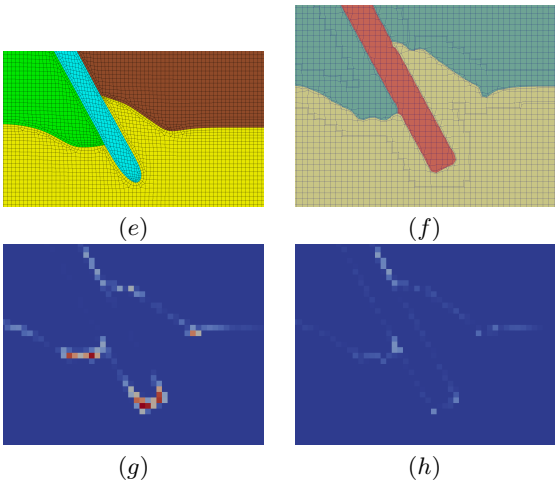


Figure 25: Comparison for the *doublebar 0.5sec* case. (a) and (b) output meshes from Sculpt and our method; (c) and (d) their respective per-cell discrepancy.

by means of being able to control cell mesh quality to make the generated mesh a valid input for a Lagrangian mesh. While the method can still be improved, it reaches the aim of keeping the cell quality above an acceptable threshold. It was made possible by controlling node movement and introducing locality during pillowing.

Our pipeline is robust, but in order to improve the results further, the mesh adaptation pipeline will be simplified in the short term in order to be a more traditional loop adaptation process as it is done for triangular and tetrahedral mesh adaptation. The key point to be added in this process will be a new node movement based on successive simple operations put in an iterative process: node position computation, controlled movement towards the pixelated model, surface and volume smoothing. It will improve upon the "projection and smooth" step done at the beginning of the mesh adaptation for some 3D cases.

As seen on several examples throughout the paper, the pillowing operation can drastically decrease the mesh quality. Detecting configurations where no good pillowing can be applied could help offer a diagnostic on the pillow resulting quality and inform the user that his expected quality threshold will not be reachable, or at the cost of capturing the interfaces very poorly. More investigation of local pillowing will be done.

Eventually, our output mesh is a transformation of the input mesh, keeping the number of cells and their size quite similar. As it is usually expected of a Lagrangian simulation to require fewer cells than an Eulerian simulation for the same accuracy, we might need to coarsen and/or refine the output mesh. This process will use the material assignment and the interface reconstruction results to coarsen and refine the mesh locally before the mesh adaptation steps. Analyzing the inconsistencies between the mesh and the pixelated geometric model (see Figure 26) should provide hints to know where the grid could be refined.

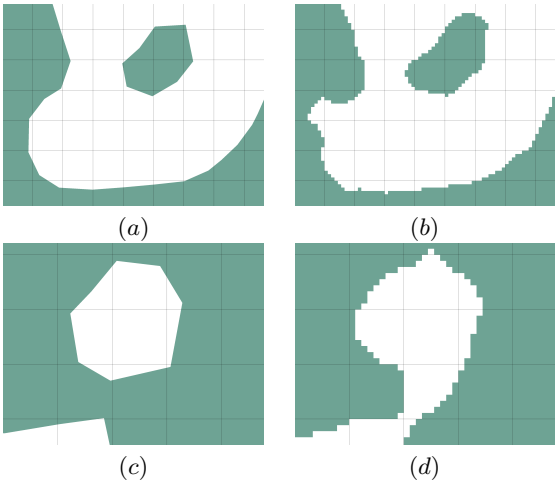


Figure 26: Topology similarities and differences between our final mesh and the pixelated interfaces on close-up of the hydro_toro_a case; the black wireframe is the original grid mesh carrying volume fractions data. (a) and (b) both return the same topology for the green material; (c) and (d) the topology is not the same.

References

[1] Schneiders R. "A grid-based algorithm for the generation of hexahedral element meshes." vol. 12, no. 3, 168–177, 1996

[2] Owen S.J., Staten M.L., Sorensen M.C. "Parallel Hex Meshing from Volume Fractions." *Engineering with Computers*, vol. 30, no. 3, July 2014

[3] Zhang Y.J. *Geometric Modeling and Mesh Generation from Scanned Images*. CRC Press, 2016

[4] "OpenFoam User Guide, SnappyHexMesh." `https://cfd.direct/openfoam/`, 2017

[5] Distene. "Volume Meshing: MeshGems-Hexa." `http://www.meshgems.com`, 2017

[6] Anderson J.C., Garth C., Duchaineau M.A., Joy K.I. "Discrete Multi-Material Interface Reconstruction for Volume Fraction Data." vol. 27, no. 3, 1015–1022, 2008

[7] Anderson J.C., Garth C., Duchaineau M.A., Joy K.I. "Smooth, Volume-Accurate Material Interface Reconstruction." vol. 16, no. 5, 802–814, 2010

[8] Kucharik M., Garimella R.V., Schofield S.P., Shashkov M.J. "A comparative study of interface reconstruction methods for multi-material ALE simulations." *Journal of Computational Physics*, vol. 229, no. 7, 2432–2452, 2010

[9] Freitag L.A. "On Combining Laplacian And Optimization-Based Mesh Smoothing Techniques." *Trends in Unstructured Mesh Generation*, pp. 37–43. 1997

[10] Mitchell S.A., Tautges T.J. "Pillowing Doublets: Refining A Mesh To Ensure That Faces Share At Most One Edge." *proceedings of the 4th International Meshing Roundtable*, pp. 231–240. Sandia National Laboratories, October 1995

[11] Owen S.J., Brown J.A., Ernst C.D., Lim H., Long K.N. "Hexahedral Mesh Generation for Computational Materials Modeling." vol. 203, 167–179, 2017

[12] Hege H.C., Seebass M., Stalling D., Zöckler M. "A Generalized Marching Cubes Algorithm Based on Non-Binary Classifications." Tech. Rep. SC-97-05, ZIB, Takustr. 7, 14195 Berlin, 1997

[13] GLPK. "GLPK (GNU Linear Programming Kit)." `https://www.gnu.org/software/glpk/`, 2018

[14] Boykov Y., Veksler O., Zabih R. "Fast approximate energy minimization via graph cuts." vol. 23, no. 11, 1222–1239, 2001

[15] Boykov Y., Kolmogorov V. "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision." vol. 26, no. 9, 1124–1137, 2004

[16] Kolmogorov V., Zabin R. "What energy functions can be minimized via graph cuts?" vol. 26, no. 2, 147–159, 2004

[17] Knupp P. "Algebraic mesh quality metrics." *SIAM J. Sci. Comput.*, vol. 23, no. 1, 193–218, 2001

[18] Toro E. "Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction." *Riemann Solvers and Numerical Methods for Fluid Dynamics*. 2009

[19] Guy R. "A PIC/FLIP fluid simulation based on the methods found in Robert Bridson's "Fluid Simulation for Computer Graphics": rlguy/GridFluidSim3D." URL `https://github.com/rlguy/GridFluidSim3D`

[20] Owen S.J., Shelton T.R. "Evaluation of grid-based hex meshes for solid mechanics." *Engineering with Computers*, vol. 31, 5–29, 2015

[21] Le Goff N., Ledoux F., Owen S. "Hexahedral mesh modification to preserve volume." vol. 105, 42–54