Intercode hexahedral meshing from Eulerian to Lagrangian simulations

Nicolas Le Goff, Franck Ledoux and Jean-Christophe Janodet

Abstract In this chapter, we deal with the problem of mesh conversion for coupling lagrangian and eulerian simulation codes. More specifically, we focus on hexahedral meshes, which are known as pretty difficult to generate and handle. Starting from an eulerian hexahedral mesh, i.e. a hexahedral mesh where each cell contains several materials, we provide a full-automatic process that generates a lagrangian hexahedral mesh, i.e. a hexahedral mesh where each cell contains a single material. This process is simulation-driven in the meaning that the we guarantee that the generated mesh can be used by a simulation code (minimal quality for individual cells), and we try and preserve the volume and location of each material as best as possible. In other words, the obtained lagrangian mesh fits the input eulerian mesh with high-fidelity. To do it, we interleave several advanced meshing treatments – mesh smoothing, mesh refinement, sheet insertion, discrete material reconstruction, discrepancy computation, in a fully integrated pipeline. Our solution is evaluated on 2D and 3D examples representative of CFD simulation (Computational Fluid Dynamics).

Nicolas Le Goff
Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, 91680, Bruyères-le-châtel, CEA, DAM, DIF, F-91297 Arpajon Cedex, France e-mail: nicolas.le-goff@cea.fr

Franck Ledoux
Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, 91680, Bruyères-le-châtel, CEA, DAM, DIF, F-91297 Arpajon Cedex, France e-mail: franck.ledoux@cea.fr

Jean-Christophe Janodet
IBISC, Univ. d'Evry, Université Paris-Saclay, 91025 Evry, France, e-mail: jeanchristophe.janodet@univ-evry.fr

1 Introduction

Many numerical simulation codes require to discretize a study domain Ω by a *mesh* that partitions Ω into a set of basic connected elements, called *cells*, that will carry physical data (pressure, temperature, material description, etc.). Cells define a support to apply traditional numerical methods, like Finite Element Methods (FEM) or Finite Volumes Methods (FVM), which rely on basic functions that are defined onto finite elements or finite volumes. Depending on the numerical methods, many types of meshes can be used and they can differ in the way they match simulation materials. Let us consider a study domain made of two materials A and B that are in movement, and depicted in various ways on Figure 1. The first line shows the "physical" evolution where A and B are respectively colored in blue and red. Materials are moving accordingly to the effect of a simulated physical phenomenon, leading to an expansion of A that tries to fill the whole domain, while B is contracted. Meshes are shown on the three remaining lines. At the initial time, the meshes are identical: a *pure* quadrilateral mesh, that is a mesh where each 2D cell is a quad and contains a single material (A or B). Three approaches are then possible:

- **Euler.** On the second line, the mesh remains fixed while the materials move through cell boundaries; when several materials are present inside one cell, the cell is called *mixed* and we denote such a mesh as being *eulerian*. In this case the interface between A and B is lost.
- **Lagrange.** On the third line, the mesh moves at the same speed as materials do. Cells remain pure during the whole simulation but their geometry can drastically change. We note such a mesh as *lagrangian*, and the interface between A and B is totally defined by a set of mesh nodes and edges.
- **ALE.** Eventually, on the fourth line, the mesh moves but not at the same speed as materials do. Cells can become mixed but their geometry remains controlled by the simulation code. We qualify such a mesh as being *ALE*, for *Arbitrary Lagrangian-Eulerian*.

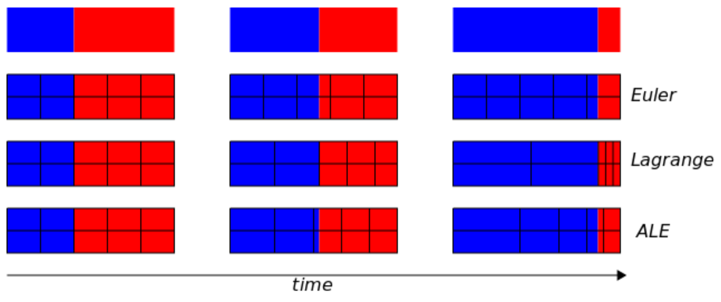


Fig. 1 A 2D domain made of two materials (top) discretized in eulerian, lagrangian and ALE ways.

Conversion of data between simulation codes are usual in real-case studies where different codes are used to solve complex multi-physics problems. It can be done in many manners going from loosely-coupled codes, where codes are assembled in a pipeline and communicate by reading and writing files, to tightly-coupled codes, that are interleaved in a simulation loop and share in-memory data. We focus in this work on loosely-coupled codes. As each code has its own input and output requirements, an intercode tool is required to convert the output of one code into the input of the next one. In the case of converting the output of an eulerian code into the input of a lagrangian code, this task is not trivial at all, especially when we aim to generate full hexahedral lagrangian meshes (see Figure 2). Eulerian meshes are in most cases easy to generate. They are typically a grid, where mixed cells are specified by the *volume fractions* of the materials they contain. Some of those eulerian codes have Adaptive Mesh Refinement (AMR) capabilities, meaning that, usually by use of an octree-like data structure, the mesh is locally refined or coarsened to respectively track a phenomenon of interest or to reduce the execution time and the memory footprint of the simulation[1]. On the opposite, in the case of lagrangian codes, quadrilateral and hexahedral meshes are quite more challenging to generate especially in 3D.

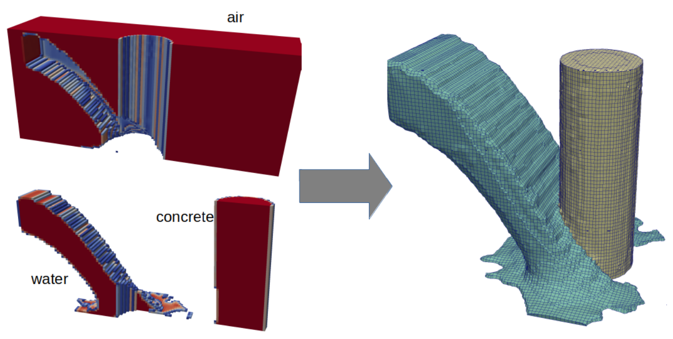


Fig. 2 A 3D Example of what we want to achieve after having run the simulation code Gridfluid [14] with water being poured against a concrete pillar. On the left, our input, a grid mesh carrying the volume fractions of respectively the water, the concrete pillar and the air. On the right the output lagrangian mesh where we only display the the water and the concrete. The air is also discretized by a pure hexahedral mesh but not shown here for a sake of clarity. Obtained mesh quality is controlled for the three materials (water, concrete, air).

This issue is not directly handled by many research works that try either to create volume-preserving material interfaces without satisfying some meshing constraints, or to generate a full hexahedral mesh without preserving material volumes. First are the interface reconstruction methods [22]. These methods take an eulerian mesh as an input and reconstruct the interfaces between materials inside each cell; they are

[1] We do not consider AMR meshes in this work.

classically used in ALE simulation codes and in associated scientific visualization software [9, 2]. While the volume fractions preservation is actually enforced by design, a limitation of those methods is that the obtained interfaces do not fit our purpose, as they are jagged, not continuous and potentially with small slivers of materials. It seems impossible to use such interfaces for projecting mesh nodes associated to those surfaces without any significant modification, which would in turn render the volume preservation property null and void. Secondly figure among the variety of hexahedral meshing techniques the overlay-grid methods [30], or octree-based isocontouring methods [34], where a shape – an explicit geometrical CAD model for instance – that needs to be meshed is embedded into a mesh that discretizes its bounding box. Said mesh will usually be a grid, easy to generate and possibly locally refined [31]; its cells are assigned to the components of the geometrical model and those outside discarded, and the mesh is then deformed or a padding layer of hexahedral cells is inserted in order to capture the geometrical features of the model. Based on this pioneer idea of embedding a shape into an existing mesh, several works were proposed during the two past decades to improve the process. For instance, [34, 37] consider single-material domains and use local refinement patterns to adapt the mesh around the material interface; sharp features are tried to be preserved in [18]; multi-material are considered with hybrid meshes mixing tetrahedral and hexahedral elements in [36]; mesh quality is improved in [29]. We can also cite geometric flow-based methods that preserve volume for each material region and that have been approved mathematically and employed in mesh quality improvement [35, 17]. Contrary to the interfaces reconstruction methods, extracting a geometrical model from that mesh will give a relatively smooth model with a clean topology, but as a drawback one does not guarantee to preserve the volume of materials. There are additional incompatibilities with our aim: first the expected inputs of those methods are expliclity-defined models, likd CAD ones, not meshes carrying volume fractions; secondly, most of those methods are designed to mesh only one component and cannot be used to mesh a CAD model that is the assembly of several pieces, while we have several materials. Thirdly, those methods heavily rely on the ability to generate an adequate initial mesh, possibly with local refinements to offer more robustness, to better capture the CAD or to provide a modicum of volume preservation [11]. This is again in direct conflict with what we need, as in our case the input mesh is fixed as part of our input.

In order to get a valid mesh for numerical simulation while preserving materials, we propose to adopt an overlay-grid approach, that meets our concerns, and more specifically, we extend the Sculpt algorithm [27, 26], which implements an overlay-grid approach considering volume fractions data as an input. Starting from a 3D eulerian mesh M_E with a set of materials $\mathcal{M}$, we aim to generate both an interface geometrical model $\mathcal{G}$, and a full hexahedral mesh M_L such that :

1. $\mathcal{G}$ is usable for mesh generation, that is its surfaces are smooth and “cleanly”[2] connected along curves and vertices;

[2] When three material meet, we need to get a clear curve that is adjacent to the three of them and not a series of small curves and surfaces that would be adjacent to only a pair of them.

2. Cells of M_L fit minimum quality requirements to be used by a FEM or FVM simulation codes;
3. Every material $m \in \mathcal{M}$ is preserved as best as possible, in the meaning that the overall volume of m is similar in M_E and M_L and it is located at the same spatial location.

To do it, we designed a fully-integrated pipeline that interleaves several advanced meshing treatments – mesh smoothing, mesh refinement, sheet insertion, discrete material reconstruction, discrepancy computation. In this chapter, we focus on the generation of the geometrical model $\mathcal{G}$, which is a key component of our pipeline. It is described in Section 3. Our solution is evaluated on 2D and 3D examples representative of CFD simulations (Computational Fluid Dynamics) in Section 4. But beforehand, we give an overview of the full pipeline in Section 2.

2 Eulerian to lagrangian hexahedral remeshing pipeline

In order to generate a lagrangian hexahedral mesh M_L that fits the materials carried by the input eulerian mesh M_E, we follow the process depicted on Figure 3, where three main stages are identified:

1. **Geometry Extraction.** The first stage consists in extracting a valid geometrical model $\mathcal{G}$ while assigning each mixed cell of M_E to a single material. A refinement stage is interleaved into it when we encounter topologic inconsistency between $\mathcal{G}$ and the pure mesh M_L deduced from M_E. Figures 4.*a* to *d* show such a refinement.
2. **Quality-driven mesh projection.** Then, the mesh M_L is modified and its nodes moved in order to obtain both a minimum cell quality (which is required by simulation codes) and a mesh that fits the topology of $\mathcal{G}$. Even if what defines a "good" mesh varies between simulation codes and the cases run, some geometrical cell quality criteria are common among the simulation codes, which cannot operate when even one single cell becomes too distorted [20].
3. **Discrepancy-driven mesh deformation.** Eventually, we apply a smoothing stage to measure and improve the volume preservation compared to the eulerian mesh M_E they originated from.

In this chapter, we focus on the first stage of this process. But let us give a few words abouts the second and third stages. The quality-driven mesh projection strongly relies on three key ingredients: first the existence of the geometrical model built at stage 1; second, the idea to deform the mesh to fit the geometry but only if the cell quality meets lagrangian code quality requirements, i.e. material interface nodes are moved towards the geometrical model but they reach it only if the quality of adjacent cells is good enough; third, some topological local padding operations are performed to give more degree of freedoom to nodes that do not reach the expected location. A full description of this stage can be found in [23].

The discrepancy-driven mesh deformation process is a smoothing stage that aims to measure and improve the volume preservation compared to the eulerian mesh

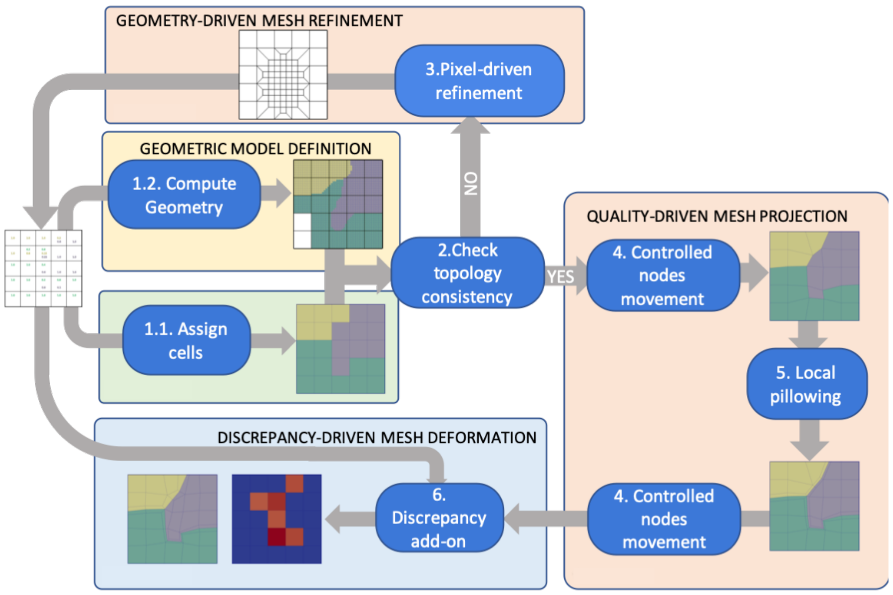

GEOMETRY-DRIVEN MESH REFINEMENT
3.Pixel-driven refinement
GEOMETRIC MODEL DEFINITION
1.2. Compute Geometry
1.1. Assign cells
NO
2.Check topology consistency
YES
QUALITY-DRIVEN MESH PROJECTION
4. Controlled nodes movement
5. Local pillowing
4. Controlled nodes movement
DISCREPANCY-DRIVEN MESH DEFORMATION
6. Discrepancy add-on

Fig. 3 Main stages of ELG, an Euler to Lagrangian hexahedral remeshing pipeline.

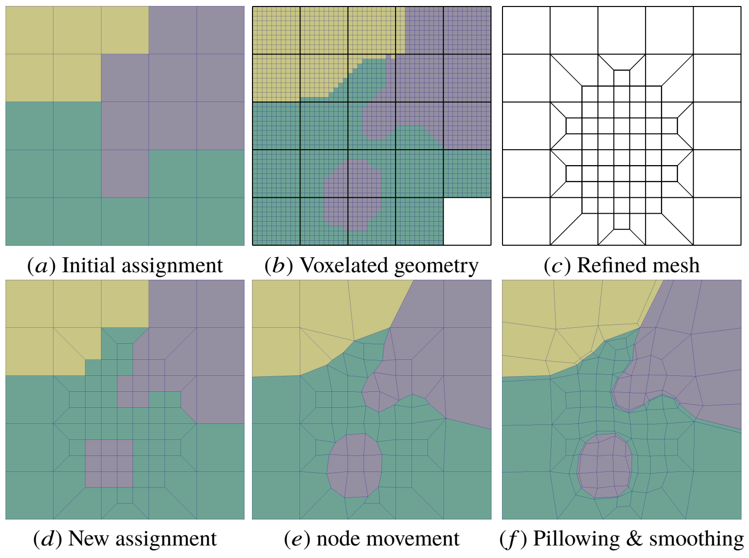


Fig. 4 Example of input mesh refinement in order to fit the built geometrical model.

M_E they originated from. This stage is done as a post-process to ensure the overall constraint, which is to avoid to deviate too much from the input data, namely the material volume fractions. This constraint is both strong and weak: strong because as this process is used in a physical simulation pipeline, it is mandatory to preserve physical quantities as best as possible to get "high-fidelity" results; weak since this problem is overconstrained and so approximations are unavoidable. Getting both high-fidelity material preservation and a smooth clean geometrical definition of the material interfaces is quite difficult in many cases. In order to control the material preservation, we use the notion of *discrepancy* as introduced in [24]. Let A and B be two meshes of a domain Ω and $\mathcal{M}$ be the set of materials that disjointly fills Ω, the global difference of material volumes between A and B is

$$\Delta V = \sum_{m \in \mathcal{M}} |\Delta V_m| = \sum_{m \in \mathcal{M}} |V_m^A - V_m^B| \tag{1}$$

where V_m^A and V_m^B are respectively the volume of a material m in meshes A and B. Minimizing ΔV only ensures a global volume preservation, which proves not to be sufficient as it can lead to unexpected results (see Fig. 5). To get a local volume control, it is mandatory to consider the notion of *local discrepancy*. Let us consider the meshes A and B again with A the input mesh and B the output mesh. Let c_j^A be a cell of M^A and m be a material of $\mathcal{M}$, we note $d_{j,m}$ the discrepancy of c_j^A relatively to material m and mesh B and we define it as

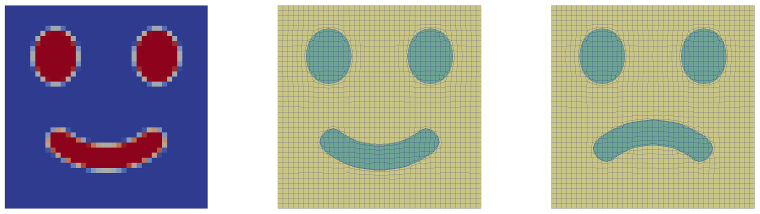
$$d_{j,m} = d(c_j^A, m) = V(c_j^A \cap B_{|m}) - f_{j,m} V(c_j^A) \tag{2}$$

where $V(X)$ is the volume of any geometric space X, $B_{|m}$ is the output mesh restricted to the pure cells of material m and $c_j^A \cap B_{|m}$ is the geometrical intersection of c_j^A with the cells of $B_{|m}$. Let us note that in practice, we compute geometrical intersections using[3] [16]. In order to compare the whole meshes A and B, we finally use the *global discrepancy* of a cell $c_j^A \in A$ defined as $d_j = d(c_j^A) = \sum_{m \in \mathcal{M}} |d_{j,m}|$, and the global discrepancy of A is defined as $d = \sum_{c_j^A \in A} d_j$. The global discrepancy gives us a way to compare material locations between two meshes both globally and locally to each cell and so each subpart of Ω. In [24], this quantity is used to move node interfaces in the lagrangian mesh B in order to improve the material preservation. We will use it in Section 4 to evaluate our solution.

3 Geometry Extraction

Geometry extraction and cell material assignment are performed pairwise in order to obtain a consistent correspondance between the geometrical model and the mesh. Cell assigment is performed as the SCULPT algorithm [32, 27] does. The mesh M_L is first created as a copy of M_E, then cells of M_E are assigned to materials on a

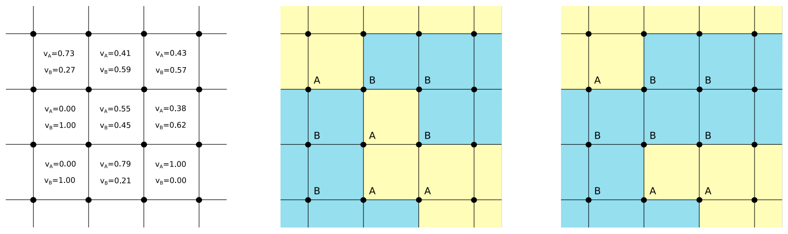
[3] The interested reader can directly use the open-source library *portage* [16] based on *R3D* [28].



(a) (b) (c)

Fig. 5 Given a grid mesh with the material volume fractions represented in (a) ranging from 0 in blue to 1 in red, we can see that despite being of the same total volume for each material, the material mesh (b) fits better the volume fraction grid than (c) does.

majority basis: a cell is assigned to the material of highest volume fraction in the corresponding input cell in M_E (see Fig. 6). A correction step can be optionnaly activated to avoid some topological issues for the incoming lagrangian simulation code (as the inability to handle non-manifold interfaces for instance). When changing materials, cells are reassigned around each problematic node to the second best (in the sense that it is closest in termes of volume fractions) correct assignment and as it could lead to non-manifold configurations appearing in the neighborhood of changed nodes this phase is executed again.



(a) volume fractions (b) majority assignment (c) assignment correction

Fig. 6 Starting from the volume fractions given in (a), grid cells are assigned to each material in two stages (b and c).

Extracting the geometrical model $\mathcal{G}$ raises two main issues: the desired interfaces have to be smooth, typically when the goal is to visualize them or to use them to generate a mesh, but at the same time they must also fit the input data as best as possible, namely the volume fractions; those two objectives can conflict with one another. To address these issues we consider that several methods are relevant. A scientific field where material interface reconstruction is extensively studied and applied is Arbitrary Lagrangian-Eulerian (ALE) CFD simulations. Obtained reconstructed interfaces have a built-in volume fractions preservation but are not smooth and discontinuous across cells [22]. Most of these methods also have the additional drawback of being material order-dependent. In order to visualize material interfaces for eulerian meshes, several approaches relies on a voxel decomposition of the mixed cells. Each mixed cells is split into sub-elements – typically an hexahedron will be refined into a grid of voxels – on which a partitioning strategy is applied 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. Finally, in the Sculpt algorithm [27], interface nodes' locations are computed using the volume fractions (and their gradient) and the material assignment but it fails to capture at all the materials that have no majority volume fractions in any of the mesh cells;

Considering that the Sculpt strategy – that we have extensively evaluated – can become limited in some cases, and that getting smooth surface is of first interest for our purpose, we propose to rely on a voxel-based approach[4].

3.1 Interface reconstruction via voxel assignment

The discrete voxel-based interface reconstruction technique 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 a "clean" solution but with more materials small gaps or artifacts can appear that are non-desirable. In [15], the authors introduced the decomposition of mixed cells into subcells (or voxels) which are in turn assigned to the materials present in the mixed cells they were spawned from; the work in [3, 4] extends it to cases with more than three materials per cell.

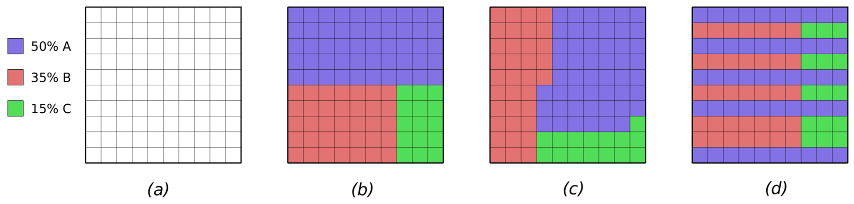


Fig. 7 Example of the voxel-assignment problem and some unexpected valid results.

The voxel-assignment problem can be stated as follows. Considering a coarse mixed cell c containing materials $m \in \mathcal{M}$, with volume fractions denoted $f_{c,m}$ such that $\sum_{m \in \mathcal{M}} f_{c,m} = 1$, and c discretized as a set $\mathcal{V}_c$ of n_v voxels, assign one single material m to each voxel of $\mathcal{V}_c$ while ensuring material volume preservation. Figure 7 illustrates

[4] We can note that as we handle both 2D and 3D cases in structured and unstructured cases, throughout this chapter we will use the term "voxel" as a misnomer in place of pixel (in 2D), sub-cell or sub-element.

such a situation where in (a), a coarse mixed cell, made of 50% of material A, 35% of material B and 15% of material C, is split into 100 voxels. Results given in (b), (c) and (d) are valid solutions for the voxel-assigment problem, as they all respect the volume fractions, but they are wildly different from one to the another; it probably means that our problem could receive some additional description in order to be appropriately formalized. The voxel-assignment should aim towards several objectives:

- First, to enforce the volume preservation of each material, we favor solutions having a low *discrepancy*, defined as the sum over each coarse cell c of the absolute difference between the volume of each material present in c (for material m it is $f_{c,m}V(c)$) and the sum of the volumes of the voxels of c assigned to m. It expresses whether the voxels material assignment fits the volume fractions;
- Secondly, as usual in partitioning algorithms, we favor connected components for each material. It translates into minimizing the *edgecut* function, defined as the sum of the number of pairs of adjacent voxels assigned to different materials;
- Thirdly, surrounding pure cells of c provides the initialization to our problem. If a mixed cell is bounded by pure cells, then we extend the voxelization process to the vicinity of c, i.e all the mixed cells and their adjacent pure cells are subdivided into voxels, and voxels spawned from pure cells are already assigned to the material of their corresponding pure coarse cell, leaving those spawned from mixed coarse cells as being "free" (see Fig. 8).

Such a problem can be formally described by the following mixed-integer linear program (MILP):

$$\begin{cases} min \sum\limits_{v\in\mathcal{V},m\in\mathcal{M}} |ma_{v,m} - \frac{1}{|N(v)|}\sum\limits_{w\in N(v)} ma_{w,m}| & \\ \text{constrained to} & \\ ma_{v,m} \in \{0,1\} & \forall v,m \\ \sum\limits_{m\in\mathcal{M}} ma_{v,m} = 1 & \forall v \\ \sum\limits_{v\in\mathcal{V}_c} ma_{v,m} = nbsub * f_{c,m} & \forall m, \forall c \end{cases}$$

where $\mathcal{V}$ is the total set of voxels of the whole domain, $ma_{v,m} = 1$ if voxel v is assigned to material m and $ma_{v,m} = 0$ otherwise; $N(v)$ is the set of voxels adjacent to v. The first two constraints indicate that every voxel has one assignment and only one. The third constraint expresses that we want to have a *discrepancy* equal to zero (*nbsub* being the number of voxels in a coarse cell c). The objective function that we want to minimize reflects the aim for voxels assigned to the same material to be clustered together, i.e. having a low *edgecut*. Since our variables are integers, we in fact have a mixed-integer linear problem. This type of problem can be solved using various solving libraries [12, 6, 1, 13]. But it is in practice too computationaly expensive to be used in our pipeline. That is why we propose a greedy heuristic that is designed to fit our specific requirements.

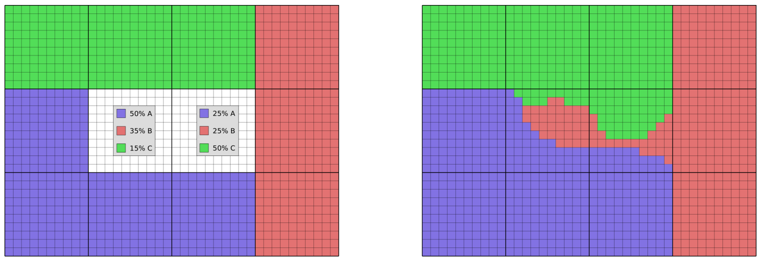


Fig. 8 Two mixed cells surrounded by pure cells where voxels must be partitioned into materials (left). Considering adjacent pure cells and the objective of minimizing the number of connex components for each material could lead to the result shown on the right.

3.1.1 A greedy heuristic to assign voxels

We follow the Algorithm 1 to assign voxels. It iteratively assigns a material value to free voxels that are surrounded by enough already material-assigned voxels (see the evolution at several iterations in Figure 9). Some 3D results are shown on Figure 12. The underlying idea of this algorithm is to assign a material to each voxel following an advancing-front strategy. We consider a set S of connected mixed cells as a starting point (orange cells on Figure 9-*a*). Each cell $c \in S$ is split into voxels that we have to assign to a specific material. The material each voxel will be assigned to depends on the volume fractions of materials that compose its parent cell in S. For example, on Figure 9, volume fractions of the central cell are given; the central cell should be filled by 40% of green and 60% of grey voxels at the end.

In order to assign a material to a voxel v, we consider the materials that are already assigned in its vicinity (the 8 surrounding pixels in 2D when the case is structured) and we diffuse those materials into the voxel v. Voxel v is assigned to a material m if its newly computed volume fraction is higher than a minimum threshold. The threshold value is iteratively decreased in order to avoid blocking situations where the algorithm is unable to assign a material to any voxel during an iteration. At the end of each iteration, the volume fractions to reach for each material in a cell are updated (see Figure 9-*a* to *e*). With this strategy voxels on the boundary of S tend to be assigned first and we get the expected advancing front assignment.

To evaluate our solution, it was compared on several cases with three other approaches: the mixed-integer linear program previously given, that provides us an optimal solution to the problem; simulated annealing as used in [3, 9]; as partitioning a graph via techniques like graphcut [8, 7, 21]. Comparisons between those four approaches were fully described in in [23]. Examples of results are given in 2D on Figure 10. The MILP implementation is impractical, as it does not return a solution in an acceptable time. For this small example, we stopped the optimization process after 5 minutes[5]. Let us note that the result is valid, in the meaning meaning that it fits the constraints, but not optimal, which is the case in Figure 10-*a*. The graphcut

[5] We use the open-source GLPK software [12].

Algorithm 1: Voxels assignment greedy heuristic.

```
Data: volume fraction $VF$, voxelated mesh
Result: Voxels assignment
1  $threshold \leftarrow 1.$
2  freeVoxels $\leftarrow$ allVoxels
3  fixedVoxels $\leftarrow \emptyset$
4  $vf \leftarrow (VF,$ freeVoxels)
5  for $freeVoxels \neq \emptyset$ do
6      /* get the free voxels with a vf higher than the threshold for one material */
7      fixedVoxelsToAdd $\leftarrow$ extractVoxelsAbove(freeVoxels, threshold)
8      fix(fixedVoxelsToAdd)
9      if $fixedVoxelsToAdd \neq \emptyset$ then
10         reduce $threshold$
11     end
12     /* update the vf while substracting the voxels already assigned */
13     $vf \leftarrow$ update($VF$, freeVoxels)
14     for $iter \leq maxNbIter || convergence$ do
15         /* kind of a vf smoothing */
16         $vf \leftarrow$ average($vf$) for voxels where < $threshold$
17         normalize($vf$)
18     end
19 end
```

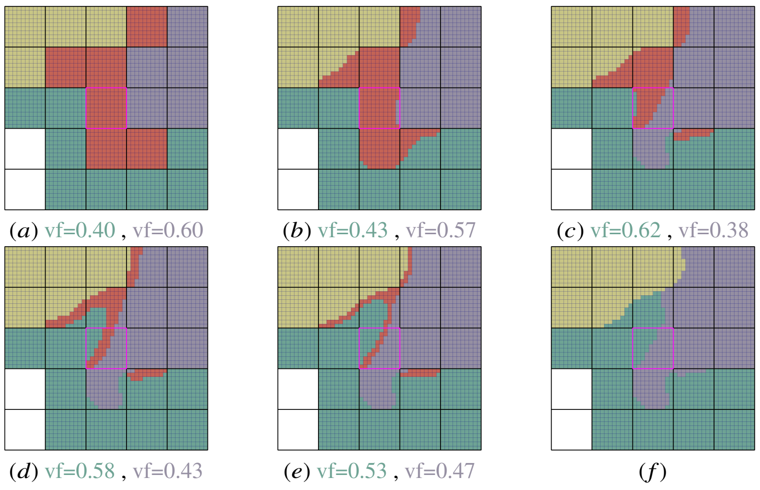


Fig. 9 A 5x5 example where the evolution of the volume fractions (see Algorithm 1:13) assigned to the free voxels of the central coarse cell are given below each figure. The wireframe black grid is the coarse mesh and the voxels colored in orange are those not yet assigned.

approach tends to return straight interfaces, resulting in a good edgecut, but is quite bad when considering the *discrepancy*. That leaves us with the simulated annealing, which is a little better than our greedy heuristic regarding the edgecut in the case of a grid, but fares badly concerning the *discrepancy* in unstructured cases, as shown in Figure 11. All of those methods have the same memory limitation, as the submesh, i.e. the set of voxels, can be quite large. In practice, we use our heuristic to build the voxelated interfaces as it is a good compromise between structured and unstructured cases and does not rely on tuning parameters depending on the case.

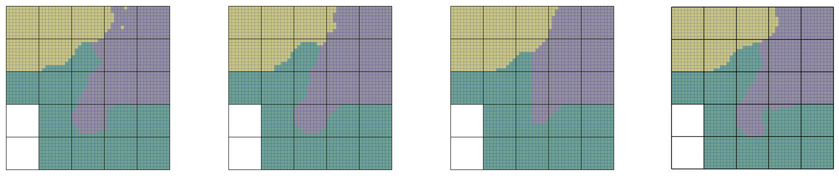
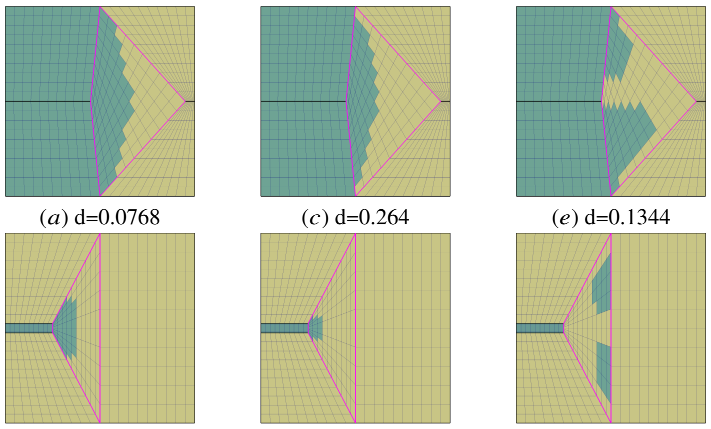

(*a*) MIP d=0, edgecut=592 (*b*) simulated annealing d=0, edgecut=536 (*c*) graphcut d=2.18, edgecut=440 (*d*) greedy heuristic d=0, edgecut=568

Fig. 10 Comparison of the interfaces reconstruction methods.


(*a*) d=0.0768 (*c*) d=0.264 (*e*) d=0.1344

(*b*) d=0.024 (*d*) d=0.1444 (*f*) d=0.1254

Fig. 11 Greedy heuristic (first column) vs. simulated annealing (second and third columns) on unstructured cases. Only the highlighted cell is mixed, and respective volume fractions are (0.5,0.5) on the top case, (0.2,0.8) on the bottom. Two different results (*c* and *e*), (*d* and *f*) are shown for the simulated annealing method since clusters of voxels can appear due to the randomness of the initial voxel assignment and the swaps.

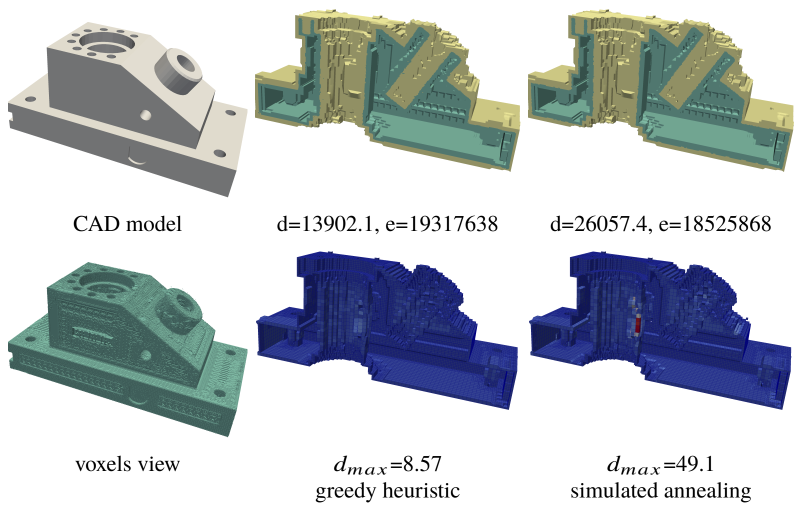


Fig. 12 Example in a real-life unstructured case. The results show that while our greedy heuristic (middle) is a little worse edgecut-wise than the simulated annealing (right), it fares better by a factor of 2 and 5 in terms of discrepancy, the total sum and its cell maximum respectively.

3.1.2 Voxel assignment improvement

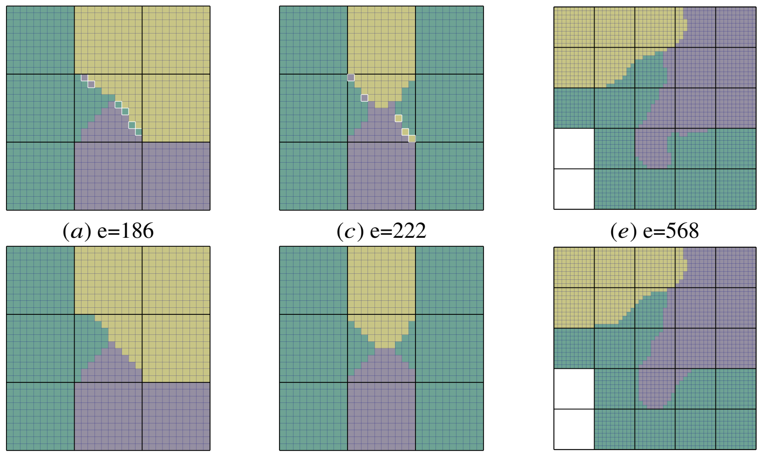
Our voxel assignment procedure can produce a few isolated voxels, which leads us to believe that the edgecut could be improved. Indeed our greedy approach tends to clump together voxels assigned to the same material but we do not make it mandatory for a free voxel to be assigned a material one of its neighbors is already assigned to. We have so devised a correction procedure that spawns from a simple consideration: as the voxels assignment can be seen as a graph partitioning problem, adjusting the obtained partitions can be considered a repartitioning problem. We have thus experimented with two well-known repartitioning algorithms: the Kernighan-Lin and the Fiduccia-Mattheyes algorithm. Interested readers can find some more up-to-date references on the subject of graph partitioning in [25, 5].

The Kernighan-Lin [19] graph bi-repartitioning algorithm takes as an input a graph with its vertices split into two sets and proceeds to improve upon the initial partitioning by exchanging vertices between the sets, two by two. The algorithm drives the swaps by determining the sequence of swaps that maximizes the gain[6]; incidentally it allows for "bad" moves, or negative gain moves as long as they are compensated. We modified the traditional implementation to fit some of our concerns. First we handle more than two partitions and secondly we restrict the possible exchanges to the swaps between voxels spawned from the same source cell.

Results in Figure 13 show that it is quite effective in reducing the *edgecut* when there are isolated assignment artifacts. As the Kernighan-Lin method atomic oper-

[6] "gain" in terms of improving the *edgecut*.

ation is the swap, it has the exact same issue as the simulated annealing when the coarse mesh is unstructured and the voxels do not all have the same volume; it will preserve the number of voxels assigned to each material, and while it may improve on the edgecut it may lead to an increase of the discrepancy (see Figure 14). In order to address this issue we applied to our problem the Fiduccia-Mattheyes repartitioning algorithm. Unlike the Kernighan-Lin algorithm, the Fiduccia-Mattheyes [10] method only changes the part to which a vertex is assigned instead of swapping two vertices at a time. In particular, it means that in the case where the initial partitions are perfectly balanced the algorithm will need to be allowed some wiggle room, i.e. the possibility to increase the imbalance between partitions, to be able to operate. In our problem it translates into our adaptation seen in Algorithm 2 line 6 where a material change for a voxel will only be considered if it does not degrade too much the discrepancy of the coarse cell this voxel is issued from. And again, negative gain moves can be performed, as long as they are compensated afterwards.


(*a*) e=186 (*c*) e=222 (*e*) e=568

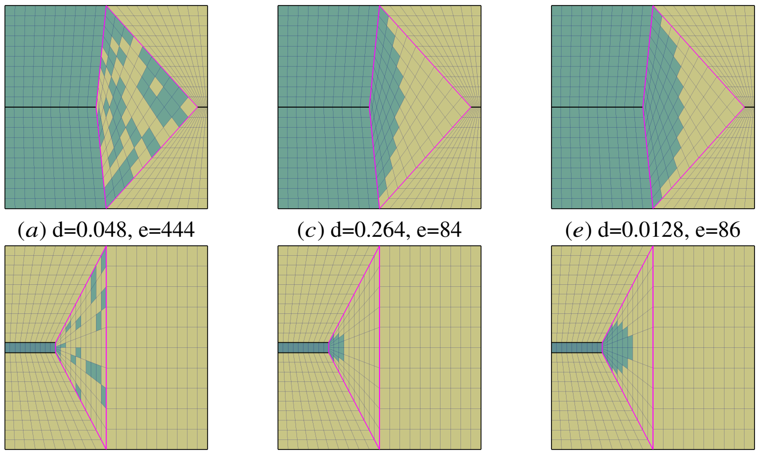
(*b*) e=144 (*d*) e=178 (*f*) e=532

Fig. 13 KL algorithm (bottom) applied after the greedy heuristic (top)

In Figure 14 is shown the benefits of being able to change the material assignment of the voxels – as is done in the Fiduccia-Mattheyes algorithm – instead of only proceeding by swapping in unstructured cases. The Kernighan-Lin implementation greatly reduces the edgecut at the cost of increasing the discrepancy, while the Fiduccia-Mattheyes manages to reduce both, albeit a little less concerning the edgecut.

Algorithm 2: Fiduccia-Mattheyes.

```
1  while $gain\_cumul > 0$ do
2      $matAssign\_tmp \leftarrow matAssign$
3      compute costs for all vertices /* one cost per material */
4      free all vertices
5      while ∃ possible material change do
6          v, m ← find best material change /* change has to be allowed under volume
               fractions constraints */
7          store best material change in sequence
8          lock(v)
9          $matAssign\_tmp(v) \leftarrow m$
10         update costs for $v$ and neighbors
11     end
12     $gain\_cumul \leftarrow$ find sequence of material changes with maximal cumulative gain
13     if $gain\_cumul > 0$ then
14         $matAssign \leftarrow$ execute material changes
15     end
16 end
```


(*a*) d=0.048, e=444 (*c*) d=0.264, e=84 (*e*) d=0.0128, e=86

(*b*) d=0.0038, e=276 (*d*) d=0.1444, e=76 (*f*) d=0.001, e=96

Fig. 14 Example of re-partitioning applied on two unstructured cases that shows the limits of only swapping the assignments (Kernighan-Lin) instead of changing the assignment (Fiduccia-Mattheyes). (*a* and *b*) an initial random assignment; (*c* and *d*) after applying the Kernighan-Lin algorithm, where we can see that while the edgecut is reduced, the discrepancy increases; (*e* and *f*) after applying the Fiduccia-Mattheyes, which reduces both.

3.2 Geometrical model definition

Starting from our input mesh carrying volume fractions, we have built a finer submesh of "voxels" and assigned those to materials; we now get the opportunity to extract the interfaces between materials from which we can build an explicit geometrical model. This model can be built at two different levels of discretization (see Figure 15): either on the fine mesh made of voxels (top row) or on the coarse mesh (bottom row). By construction, the first one provides high-fidelity to the reconstructed interfaces and preserves material volumes, while the second one is coarser, making it easier to handle (smaller memory footprint, easier to visualize and to use for mesh-to-geometry projection and smoothing). Considering our goal of getting a pure full-hexahedral mesh starting from an Eulerian mesh, we deciced to build the coarser model. Moreover, it is a straighforward manner of ensuring to get a clean topology for the geometrical model $\mathcal{G}$.

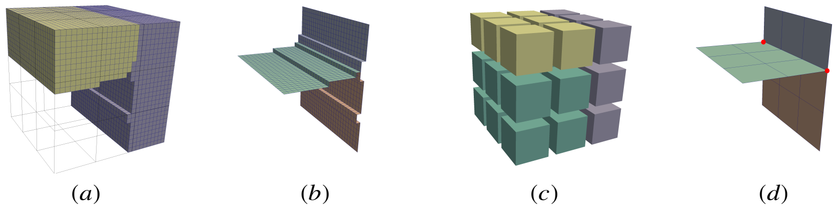


Fig. 15 Example of explicit geometrical models built from a 3 materials case in a $3 \times 3 \times 3$ grid. (a and b) the voxels assignment and its corresponding geometrical model; (c and d) the same after the assignment of the coarse cells.

Both models (the finest and the coarsest) can be built by first extracting the faces – the edges in 2D – between cells assigned to different materials (see Figure 15-b and d), then building a geometrical model $\mathcal{G} = (S, C, V)$. Starting from an hexahedral mesh $M = (H, Q, E, N)$, where H are hexahedral cells, Q are quadrilateral faces, E are edges and N are nodes, $\mathcal{G}$ can be extracted using the following rules:

- A **multi-surface** of S is a set of faces of Q that are adjacent to the same 2 materials. We get 3 such distinct multi-surfaces in the example on Figure 15;
- A **multi-curve** of C is defined as a set of edges bounding the quad of a surface $s \in S$. Considering all the faces forming s, we get the set of edges $E_s \subseteq E$ that bounds those faces. This set of edges is then partitioned into multi-curves as follows: two edges of E_s are assigned to the same multi-curve if they are adjacent to the same set of materials assigned to the cells. For instance, let us consider the green surface on Figure 15-b and d; this surface is bounded by 2 curves: the first one corresponds to the intersection between the three surfaces – all the edges are then adjacent to exactly the 3 same materials; the second one is made of the remaining edges, which are adjacent to only 2 materials and located on the boundary of the domain – here the bounding box of the input grid;

- A **multi-vertex** of V corresponds to all the nodes of N that are adjacent to the same multi-curves in C, or in other words, to the same set of materials assigned to the cells of H. Considering the example of Figure 15, the two nodes highlighted in (d) define a single multi-vertex.

Characterizing entities of the geometrical models using the material assignment of adjacent cells leads to form **multi**-entities that are potentially non-connected – in particular a vertex can have several spatial locations. For our purpose, this definition is sufficient and we do not further differentiate by splitting those entities into connected parts. In the example of Figure 15, the geometrical models extracted are both made of 3 multi-surfaces, 4 multi-curves and 1 multi-vertex with two positions.

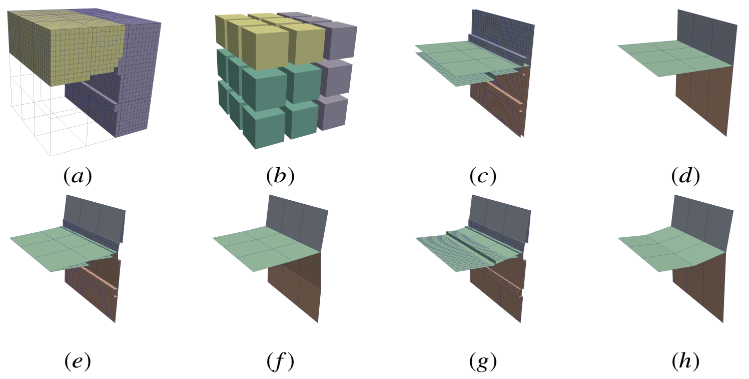


Fig. 16 Illustration of how the finer geometrical model is used as a support to project and smooth the coarser geometrical model. (c and d) the initial models; (e and f) the coarser model is projected onto the finer one; (g and h) the coarser model is smoothed while being constrained.

Now that we have those two models, we draw the correspondence between them and adapt the coarser representation by constraining it onto the finer model as seen in Figure 16. Considering a fine model G_f and a coarse model G_c extracted from compatible data (see Figure 16-a and b), multi-entities of G_f and G_c are defined from the same set of materials and so can be identified and associated through this material correspondence. In Figure 16, both models are superposed on (c), (e) and (g). To adapt G_c to fit G_f as best as possible, we first project each node of the coarse mesh that corresponds to a multi-entity of G_c onto the corresponding multi-entity of G_f (see Figure 16-e and f), then we smooth the node positions while keeping them projected onto the corresponding multi-entity of G_f (see Figure 16-g and h). The proposed solution has been widely used on several examples including realistic data, such as the result of a CFD simulation case that is shown in Figure 17 where our input is a grid mesh carrying the volume fractions at $t = 1sec$ and $t = 2sec$ of the simulation.

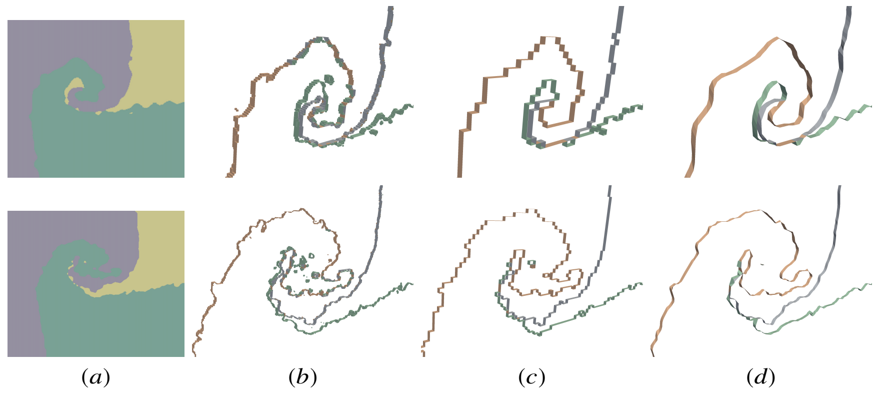


Fig. 17 Coarse geometrical model (c) projected and smoothed (d) onto the voxelated one (b) in the triple point problem at $t = 1sec$ (top) and $t = 2sec$ (bottom).

4 Results

In the previous section, we built an explicit geometrical model $\mathcal{G}$ from a finer but dirty voxel-based geometry representation. We make use of $\mathcal{G}$ in our pipeline so as to avoid situations that can be encountered when working with implicit geometry representation only. For instance, if one tries to move the nodes towards a location computed using only the input volume fractions and the cells material assignment in the eulerian mesh, and considering each node independently and not the interfaces as a whole; in particular with no care taken for the expected interfaces quality (see Figure 18-a). It becomes especially relevant in 3D where the mesh entities forming the interfaces are no longer edges but faces. Moving the nodes to their computed ideal location can by-design lead to bad quality faces, hence severely limiting nodes movement during the quality-driven mesh projection. This causes our algorithm to be stuck with a mesh still fairly stair-shaped (see Figure 18-b). Such a resulting mesh could be considered satisfactory, quality-wise, still we want the interface nodes to be located as close as possible to where the material interfaces were determined to be. Such issues are avoided when working with the coarser model $\mathcal{G}$ (see Figures 18-c and d).

In the remainder of this section, we give results obtained with the implementation of our pipeline, named *ELG*, in comparison with our own implementation of the Sculpt algorithm [27, 26], named *base algorithm*. Experiments were done on 2D and 3D CFD cases and results are gathered in Table 1. Cell quality is measured by the minimum scaled jacobian and a minimum threshold of 0.3 is chosen. Let us first begin with the triplepoint and doublebar problems for which input grids of different resolutions were available. Taking the triple point case at $1s$, we can see that for one grid resolution the base algorithm (where the mesh projection step does not strictly enforce a minimum quality) returns with a mesh containing no inverted cells (but still lower than the 0.3 minimum scaled jacobian threshold). That is not the case for

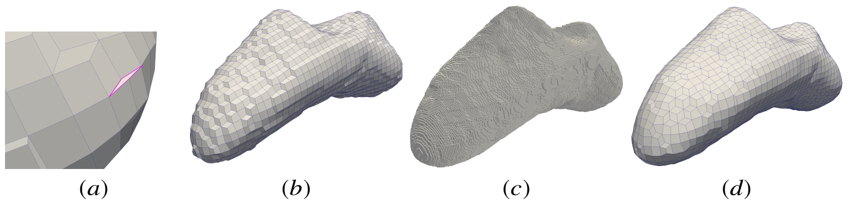


Fig. 18 Motivation of the geometrical model extraction and projection smoothing. (a) close-up of the expected interface mesh where the marked quad has a low quality of 0.068; (b) the mesh after our quality-driven mesh projection step; (c) the voxelated interface G_f between the asteroid and the exterior; (d) the coarse geometrical model G_c paired to G_f ; it replaces (a) as the expected positions of the interface nodes and no longer has low quality quads.

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 quality-driven mesh projection method that consistently works. Our method was applied in 2D on two additional hydrodynamics simulations issued from [33] (see Figure 19); in all those cases it improves the distance by at least an order of magnitude.

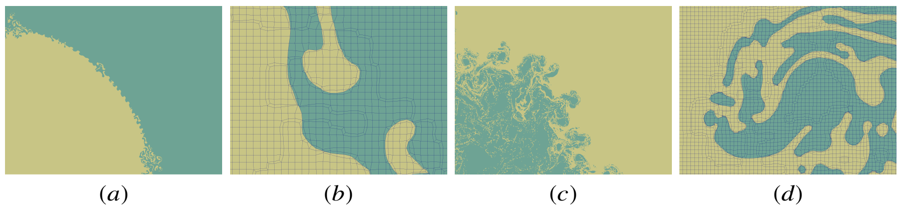
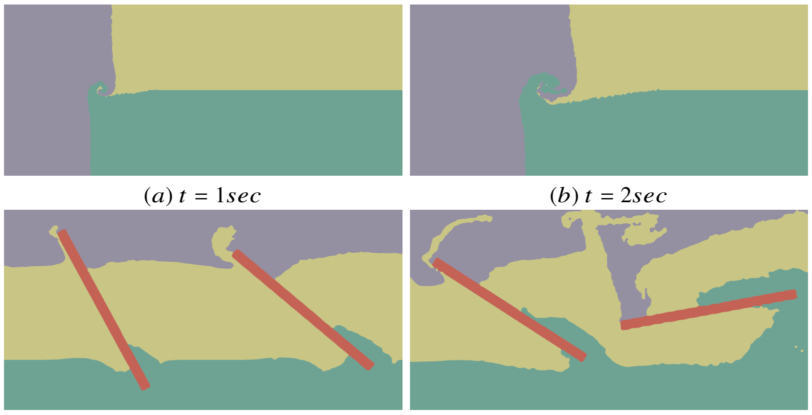


Fig. 19 Other examples of hydrodynamics simulations in 2D [33]. (a) and (c) the two cases; (b) and (d) close-ups on our resulting meshes shown respectively.

The cases from Figure 20 were extruded[7] 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 purely 2D cases. Fully 3D cases were also studied, one of which input is a grid where the volume fractions data were computed by imprinting an asteroid model onto the grid (see the example in Figure 18). Other examples are eulerian meshes from hydrodynamic simulations run using [14], a ball of liquid that drops in a box taken at several time steps, a dam that breaks in Figure 21 and a three material case where a liquid is poured onto a concrete pillar in Figure 22.

While the measured distance illustrates the efficiency of our quality-driven mesh projection step, and the use it makes of the geometric model extraction, it remains a

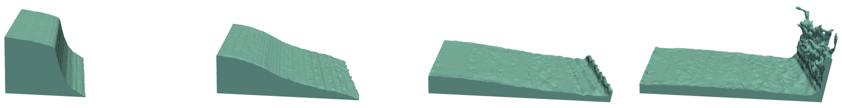
[7] The 3D mesh is created from a 2D quad mesh, lying in the XY plane, by creating successive layers of hexahedral cells along the Z direction. Volume fractions are simply derived for each hexahedral cell from their origin quadrilateral cell.



$(a)\ t = 1sec$ $(b)\ t = 2sec$

$(c)\ t = 0.5sec$ $(d)\ t = 1sec$

Fig. 20 Examples of CFD simulations in 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.



(a) time step 10 (b) time step 20 (c) time step 30 (d) time step 40

Fig. 21 Resulting meshes from our algorithm applied to the dambreak case.

metric that relies on data that we extrapolated from the input mesh M_E and its material volume fractions. In Table 2, we measure the proximity of M_L to M_E at several stages of our pipeline, using the discrepancy criterion computed by imprinting M_L onto M_E. In those examples minimal values for the scaled jacobian of 0.2 and 0.15 were chosen for respectively the quality-driven mesh projection and the discrepancy-driven mesh deformation steps. Figure 22 shows those results for the in_out_flow case.

5 Conclusion

With the pipeline described in this chapter, we have got the ability to transform the output of an eulerian simulation code into an acceptable input for a lagrangian simulation code. This input consists in a mesh, which is full-quad in 2D and full-hex in 3D. Materials are preserved as best as possible in terms of locality and global volume using the notion of global discrepancy. We also ensure the strong constraint

Table 1 Quality and distance metrics for the examples. $dist_{init}$ and $dist_{final}$ are the sums of the distance between the interface nodes and their computed destination at respectively the beginning and the end of the mesh projection phase.

case name	*minJS* *base algo*	*minJS* *ELG*	$dist_{init}$	$dist_{final}$	$\frac{dist_{final}}{dist_{init}}$
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
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
dambreak_10		0.200	34.444	17.638	0.512
dambreak_20		0.200	51.669	27.745	0.537
dambreak_30		0.200	46.866	24.097	0.514
dambreak_40		0.200	112.73	67.012	0.594
in_out_flow		0.200	132.75	75.079	0.565

of providing lagrangian cells whose minimal quality reaches a user-parameter threeshold. Ensuring it can be incompatible with the volume preservation. In this case,

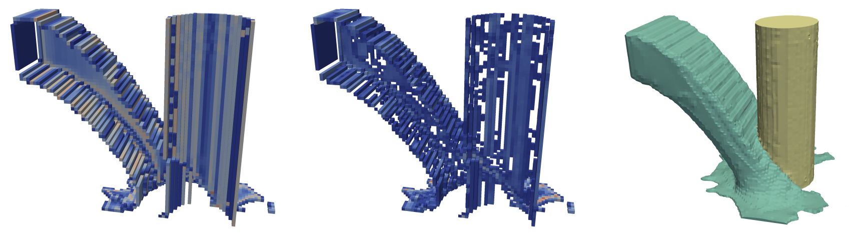

(a) d_{assign} (b) d_{proj} (c) obtained mesh M_L

Fig. 22 Discrepancy displayed after the assignment step and the mesh projection step in the in_out_flow case (a and b). Red indicates a higher discrepancy per cell while blue is lower.

Table 2 Discrepancy across the full pipeline. d_{assign} is the discrepancy measured after the assignment step (see Figure 6), d_{proj} is measured after our quality-driven mesh projection method and d_{deform} at the end after our discrepancy-driven final step.

case name	d_{assign}	d_{proj}	d_{deform}
dambreak_10	4.65907	1.3016	0.713
dambreak_20	6.2756	2.35647	1.370
dambreak_30	7.63199	3.48279	1.653
dambreak_40	12.3486	7.14286	4.486
in_out_flow	12.7552	4.70129	3.415

the priority is given to the cell quality, which is mandatory to run the lagrangian simulation code.

Let us note that the techniques depicted in this work are not limited to eulerian to lagrangian intercode problems: they are relevant as long as one is able to provide a mesh carrying volume fractions, which is typically the case for the examples issued from CAD models that we have used. Similarly, while the majority of the inputs that we have shown are grid meshes, we are not limited to those meshes and can handle any unstructured conformal hexahedral meshes as an input of the proposed pipeline. We do not handle non-conformal meshes, because as our method consists in using the input mesh as a base for our overlay-grid algorithm this base mesh should meet the requirements, first of all being conformal. Finally, most of the steps are not restricted to hexahedra and can directly accommodate other types of cells, such as tetrahedra and prisms, with the caveat of course that our output would not be an hexahedral mesh, but it was not the focus of this process.

References

1. CPLEX Optimizer, May 2020.
2. J. Ahrens, Berk Geveci, and Charles Law. ParaView: An End-User Tool for Large Data Visualization. In *Visualization Handbook*, January 2005.
3. J. C. Anderson, C. Garth, M. A. Duchaineau, and K. I. Joy. Discrete Multi-Material Interface Reconstruction for Volume Fraction Data. *Computer Graphics Forum*, 27(3):1015–1022, May 2008.
4. J. C. Anderson, C. Garth, M. A. Duchaineau, and K. I. Joy. Smooth, Volume-Accurate Material Interface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):802–814, September 2010.
5. Remi Barat. *Load Balancing of Multi-physics Simulation by Multi-criteria Graph Partitioning*. PhD Thesis, Bordeaux, December 2017.
6. Michel Berkelaar, Kjell Eikland, and Peter Notebaert. lp_solve, May 2004.
7. Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, September 2004.
8. Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

9. Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Návratil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pages 357–372, October 2012.
10. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181. IEEE Press, January 1982.
11. Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. Feature Preserving Octree-Based Hexahedral Meshing. *Computer Graphics Forum*, 38(5):135–149, 2019.
12. GLPK. GLPK - GNU Project - Free Software Foundation (FSF).
13. LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2020.
14. Ryan Guy. A PIC/FLIP fluid simulation based on the methods found in Robert Bridson's "Fluid Simulation for Computer Graphics": rlguy/GridFluidSim3D, June 2019.
15. Hans-Christian Hege, Martin Seebass, Detlev Stalling, and Malte Zöckler. A Generalized Marching Cubes Algorithm Based on Non-Binary Classifications. January 1997.
16. Angela Herring, Ondrej Certik, Charles Ferenbaugh, Rao Garimella, Brian Jean, Chris Malone, and Chris Sewell. (U) Introduction to Portage. Technical report, Los Alamos National Laboratory (LANL), 2017.
17. Leng J., Xu G., Zhang Y., and Qian J. Quality improvement of segmented hexahedral meshes using geometric flows. In *Image-Based Geometric Modeling and Mesh Generation. Lecture Notes in Computational Vision and Biomechanics*, 2013.
18. Qian J. and Zhang Y. Automatic unstructured all-hexahedral mesh generation from b-reps for non-manifold cad assemblies. *Engineering with Computers*, 28:345–359, 2012.
19. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970. Conference Name: The Bell System Technical Journal.
20. Patrick M. Knupp. Algebraic Mesh Quality Metrics. *SIAM J. Sci. Comput.*, 23(1):193–218, January 2001.
21. V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, February 2004.
22. Milan Kucharik, Rao V. Garimella, Samuel P. Schofield, and Mikhail J. Shashkov. A comparative study of interface reconstruction methods for multi-material ALE simulations. *Journal of Computational Physics*, 229(7):2432–2452, April 2010.
23. Nicolas Le Goff, Franck Ledoux, Jean-Christophe Janodet, and Steven J. Owen. Guaranteed quality-driven hexahedral overlay grid method. In *Proceedings of the 28th International Meshing Roundtable*, 2019.
24. Nicolas Le Goff, Franck Ledoux, and Steven J. Owen. Hexahedral mesh modification to preserve volume. *Computer-Aided Design*, 105:42–54, December 2018.
25. Sébastien Morais. *Study and obtention of exact, and approximation, algorithms and heuristics for a mesh partitioning problem under memory constraints*. PhD Thesis, University of Paris-Saclay, France, 2016.
26. Steven J. Owen, Judith A. Brown, Corey D. Ernst, Hojun Lim, and Kevin N. Long. Hexahedral Mesh Generation for Computational Materials Modeling. *Procedia Engineering*, 203:167–179, September 2017.
27. Steven J. Owen, Matthew L. Staten, and Marguerite C. Sorensen. Parallel Hex Meshing from Volume Fractions. In William Roshan Quadros, editor, *Proceedings of the 20th International Meshing Roundtable*, pages 161–178. Springer Berlin Heidelberg, 2012.
28. Devon Powell and Tom Abel. An exact general remeshing scheme applied to physically conservative voxelization. *Journal of Computational Physics*, 297:340–356, September 2015.
29. Jin Qian, Yongjie Zhang, Wenyan Wang, Alexis C. Lewis, M. A. Siddiq Qidwai, and Andrew B. Geltmacher. Quality improvement of non-manifold hexahedral meshes for critical feature determination of microstructure materials. *International Journal for Numerical Methods in Engineering*, 82(11):1406–1423, 2010.

30. R. Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers*, 12(3):168–177, September 1996.
31. Robert Schneiders, R. Schindler, and F. Weiler. Octree-based Generation of Hexahedral Element Meshes. *Proceedings of the 5th International Meshing Roundtable*, December 1999.
32. Matthew L. Staten and Steven J. Owen. Parallel octree-based hexahedral mesh generation for eulerian to lagrangian conversion. Report, September 2010. Library Catalog: digital.library.unt.edu Number: SAND2010-6400 Publisher: Sandia National Laboratories.
33. Eleuterio Toro. Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction. In *Riemann Solvers and Numerical Methods for Fluid Dynamics*. January 2009.
34. Zhang Y. and Bajaj C. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering*, 195(6):942–960, 2006.
35. Zhang Y., Bajaj C., and Xu G. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Communications in Numerical Methods in Engineering*, 25(1):1–18, 2009.
36. Zhang Y., Hughes T.J.R., and Bajaj C. An automatic 3d mesh generation method for domains with multiple materials. *Computer Methods in Applied Mechanics and Engineering*, 199(5):405–415, 2010. Computational Geometry and Analysis.
37. Zhang Y, Liang X, and Xu G. A robust 2-refinement algorithm in octree and rhombic dodecahedral tree based all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 256:88–100, 2013.