Automatic hexahedral meshing using Dhondt's cut algorithm

François Protais

Automatic hexahedral meshing using Dhondt's cut algorithm

FRANÇOIS PROTAIS, Inria Sophia Antipolis, France

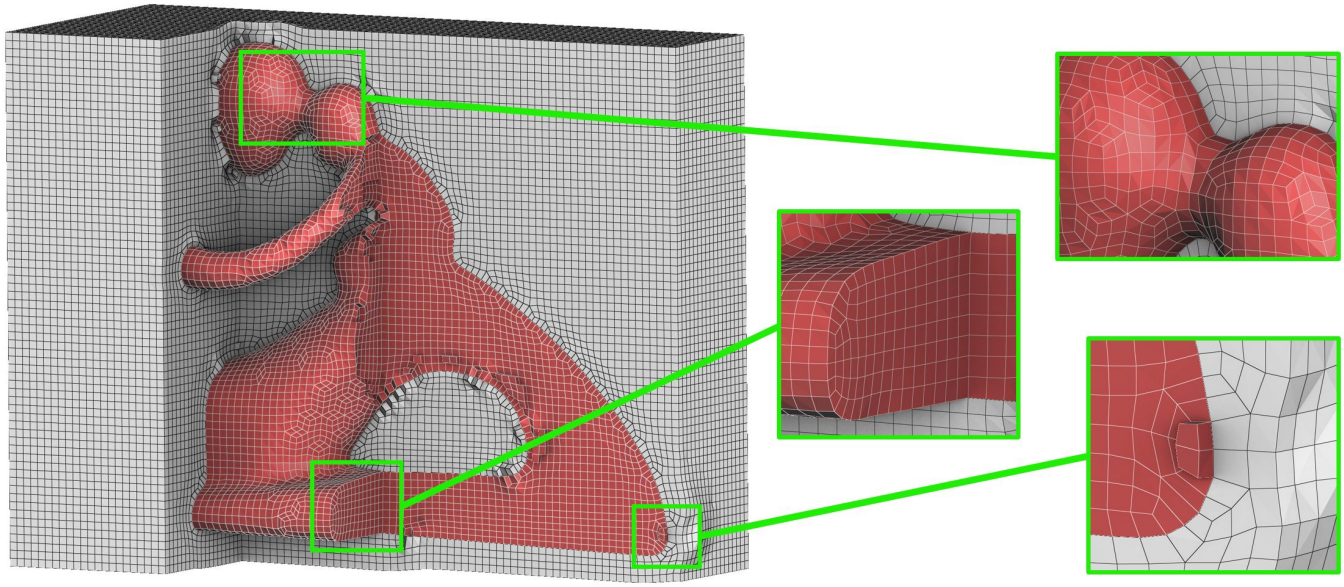


Fig. 1. Our method can construct bi-material meshes that are fully hexahedral, matching inside/outside and guaranteed free of inverted elements (scaled-Jacobian > 0).

Automatic hexahedral meshing is an active topic in research. The only current industrial approaches are octree-based methods. They provide a tool that works reliably, but gives unstructured results. Recent works tries to merge it with other approaches to bring more structure to the generated mesh. The major blocking point is often the ability to reproduce the state-of-the-art methods that rely on complex engineering. We propose an approach that gives a simple and reliable boundary management. We provide an open-source implementation that can generate hexahedral meshes without any inverted elements (guaranteed). Combined to other recent work [Livesu et al. 2022], it provides a complete octree-based hexahedral meshing pipeline.

1 INTRODUCTION

Automatic hexahedral meshing has been an active topic of research in the past two decades [Pietroni et al. 2022]. As of today, we can sort industrial hexahedral meshing algorithms into 2 categories:

(1) Not automatic
(2) Octree-based methods

The second category has a major flaw: the mesh is poorly structured. But to balance this, it has several great advantages: it start from a valid mesh and can only improve, has controlled sizing and is very efficient. Introduced in 1984 by Yerry and Shephard [Yerry and Shephard 1984], they extract an hexahedral mesh from a modified dual of an Octree grid. This mesh then needs modification such as padding and projection to best follow the boundary of the meshed domain, and its key sharp features. Industrially, the state of the art is Loïc Maréchal's Hexotic [Maréchal 2009, 2016].

Octree-based methods are still subject to recent research [Gao et al. 2019; Livesu et al. 2022]. While on paper, they seem to have reached their full potential, their is hope to obtain better results by combining them with other methods, such as parametrization or prior deformation. The heavy engineering involved makes reproductibility and flexible use difficult. [Livesu et al. 2022] did a great job at specifying the octree part of the method, but the projection is still an unclear matter. In this document, we present a simple framework to handle boundaries and the projections, using a cut-based approach introduced by Dhondt [Dhondt 2001].

The input of our algorithm is a grid overlapping the domain Ω, which can be regular or obtained through an octree. On each voxel, we extract an hexahedral pattern with valid elements, with an inside and an outside. On a pattern, some vertices are clear candidates for projection on key features. We project them iteratively and improve the mesh locally using an elliptic energy. This guarantee the validity of the mesh at all step of the process.

Improving on the work of Dhondt, we propose a simple algorithm to obtain each pattern automatically, not relying on exhaustive case handling. Moreover, with our automatic matching of features through patterns, we obtain a straight forward smoothing

You can find the code corresponding to this report at: *https://github.com/fprotais/marchinghex*.

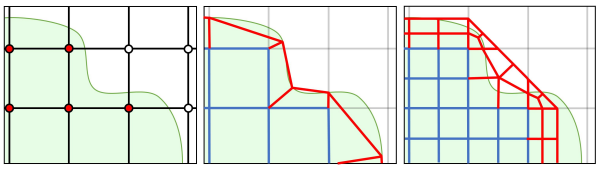


Fig. 2. From an given overlay grid **(a)**, [Maréchal 2009] projects vertices to obtain a padding layer matching the boundary **(b)**, while with Dhondt approach , we extract valid elements by construction through an additional splitting **(c)**. In both approach, a better quality and boundary matching is obtained through smoothing.

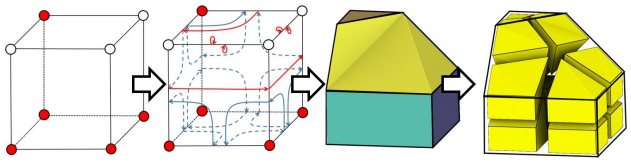


Fig. 3. Starting from a voxel configuration **(a)**, the dual of the voxel is simplified **(b)**, from which a polyhedron can be extracted **(c)**. Finally the hexahedral pattern is obtain through mid-point subdivision **(d)**.

procedure that is simple and robust. Finally, to facilitate reproductibility and future uses, the code is available on github: *https://github.com/fprotais/marchinghex*.

In a way, the proposed method can be seen as a volumetric equivalent of Dual Contouring [Ju et al. 2002], where we obtain a valid hexahedral mesh at the end. There are two main differences: first, to obtain a valid inner topology, an additional subdivision is required. Second, we don't use the same type of projection to recover sharp features, as we want to rely on actual data provided by the user. Finally, our progressive approach enables us to guarantee the validity of the mesh at all steps.

This document is a technical report, and outlines the current state of the code. While the results obtained are already worth displaying, there is still some work in progress, that are mentioned in the corresponding sections. Moreover, we only give a short discussion on related work, reader can refer to the recent hex-meshing survey [Pietroni et al. 2022] for further details.

Discussion on related works

The octree-based pipeline can be split into 3 steps:

Octree grid generation: We do not give any insight on this matter, as it is already well done in the recent [Livesu et al. 2022]. We will rely on a regular grid in this document, and leave the study of the effect of our approach on general octree grids to further work.

Hexahedral extraction: As shown in fig.2, [Maréchal 2009, 2016] directly works on the output of the octree-based grid generation, on which they need to add a layer of hexahedra to make sure that the mesh will be able to match the boundary. This step is challenging, as it may introduce poorly shaped elements. We prefer to uses Dhondt approach [Dhondt 2001], that is guaranteed to create only good elements on regular grid (and octree's?). The drawback is that it needs an additional splitting, introducing eight time the number of elements for a given grid. This implies that to achieve a desired mesh resolution, different settings on the octree generation are required.

Final mesh smoothing: [Maréchal 2009] rely on a simple procedure to move vertices locally. While they produce good element in practice, there is no published guarantees that this will be the case. We propose a smoothing procedure that slowly align with the boundary and key features. It relies on a elliptic smoothing energy [Garanzha et al. 2021] that is guaranteed to not flip elements. We sample this energy on a set of tetrahedra, similarly to the well known mesquite [Knupp 2012]. The chosen set is sufficient to ensures that the scaled-Jacobian remains positive.

Paper layout

Our key idea is to extract a valid hexahedral mesh that does not match with the boundary, and then to improve the matching slowly while keeping validity. We present a local pattern extraction in sec. 2, that is then improved using an elliptic energy to better match the boundary in sec. 3. Finally, we display a set of results for different applications in sec. 4.

2 PATTERN EXTRACTION

We denote Ω the domain that we want to mesh. We are given an overlay grid and from this grid we will extract an hexahedral mesh. To avoid any confusion, as overlay grid can be itself be considered a hexahedral mesh, we use the following naming convention:

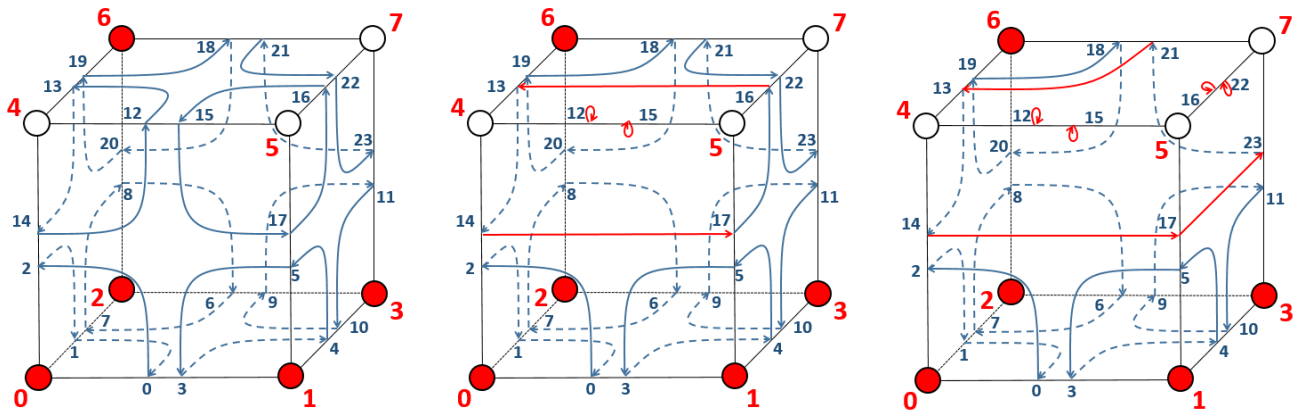
- An element of the overlay grid, later referred as *grid*, will be called a *voxel* and its vertices *nodes*,
- An element of the final hexahedral mesh, later referred as *hexmesh*, will be called a *hexahedron* and its vertices *vertices*,
- A *point* will refer to a point of the domain Ω.

The sole information needed to begin the pattern extraction process is whether each node of the grid is inside or outside Ω. Depending on the configuration of the voxel's nodes a pattern will be extracted on each voxel (subSec. 2.1). The algorithm used guarantees that neighboring voxels will match. Some patterns may require a pre-process of the grid to guarantee validness (see subSec. 2.2). SubSec. 2.3 provides an analysis of the quality of generated hexahedra on a regular grid.

2.1 Automatic pattern generation

The number of patterns needed to cover all configuration is simply $2^8 = 256$ (similarly to Marching Cubes [Lorensen and Cline 1987]). This means that hand crafting all configuration is possible, but is a tedious task. Instead we propose an alternative that rely on the computation of polyhedron matching the domain. As shown in Figure 3, we start by simplifying the dual of the voxel. The primal of the newly obtained dual give us polyhedron, and the hexahedral pattern is obtained through its mid-point subdivision.

Dual simplification. The key to the simplification is to use a simple dual array HE of 24 integers encoding the connectivity, as displayed



ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
(a) HE =	2	0	1	4	5	3	7	8	6	11	9	10	13	14	12	17	15	16	20	18	19	22	23	21
(b) HE =	2	0	1	4	5	3	7	8	6	11	9	10	-1	14	17	-1	13	16	20	18	19	22	23	21
(c) HE =	2	0	1	4	5	3	7	8	6	11	9	10	-1	14	17	-1	-1	23	20	18	19	13	-1	21

Fig. 4. Example of iteration of algorithm 1. **(a)** is initialization of dual array HE with all nodes of voxel inside. **(b)** and **(c)** show successive iterations as edges $\{4,5\}$ and $\{5,7\}$ are removed.

in Fig.4-**(a)**. Using algorithm 1, HE is iteratively modified to match the property of the voxel (see Fig.4-**(b)**-**(c)**).

Algorithm 1: Dual extraction of a voxel V

input: dual array HE of V with all its nodes *in*
Result: dual array HE of V with desired property

```
for {v1, v2} edge of V do
    if isOut(v1) and isOut(v2) then
        HE[prec(v1)] = HE[v1];
        HE[prec(v2)] = HE[v2];
        HE[v1] = −1;
        HE[v2] = −1;
    end
end
```

Return to primal. The aim is to compute a polyhedron such that if a node is inside Ω, then it is inside the polyhedron, and vice versa. With this information, we can introduce facets with new vertices where nodes are missing, and connect the polyhedron using HE.

Mid-point subdivision. The Polyhedron constructed as a key property: each of its corner has a valence 3. This means that a standard mid-point subdivisions, which introduces vertices in the middle of edges, faces and volume, will results in a fully hexahedral setup.

2.2 Invalid pattern handling

[Dhondt 2001] describes a set of nodes configurations which leads to poor hexahedra. Using our method, we observed that issues arise when facets exhibit an in-out-in-out configuration. Intuitively, we

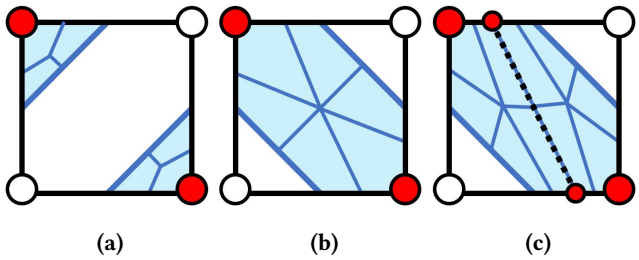


Fig. 5. **(a)** and **(b)** show 2 different meshes for one facet setup. This ambiguity leads to non matching facet and poorly shaped elements. **(c)** [Dhondt 2001] proposed to introduce cuts in the grid that remove those ambiguities all together.

can see on figure 5-**(a)**-**(b)** that this introduce ambiguity as which two couples should be linked inside the domain. On these setups, our method introduces topologically invalid hexahedra.

Thankfully, Dhondt also noticed that a clever introduction of cut in the grid could avoid these cases completely. On the previous example, this is equivalent to cutting the problematic facet into 2 valid facets. The drawback of the approach is that the cut must be propagated along a whole layer of the grid, introducing a significant number of voxels. The bright side is that those configurations can be easily avoided through a smart choice of grid, and a simple 3 cut is enough to guarantee the overall quality of the hexahedral mesh (see next subSec).

2.3 Quality

Starting from a regular grid , we can easily check the quality of all possible configuration (see Fig. 6). All configuration contain only

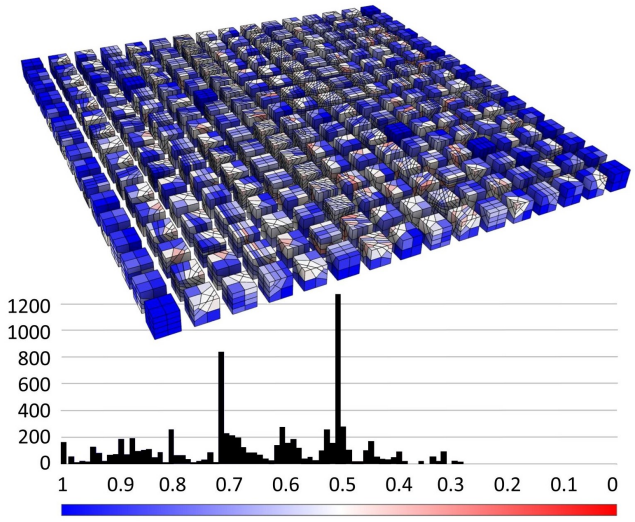


Fig. 6. All 256 possible configurations generated on a standard grid, with both inside and outside meshes. Minimum/average scaled-Jacobian is 0.277/0.643

valid elements, with a worst scaled-Jacobian of 0.277. Both inside and outside meshes are perfectly compatible. Worse configurations can need up to 3 cuts to ensure that they are valid. To generate all the configurations, there an available binary *make_example* in the code.

A work in progress is to test the quality of the configuration through different transformations, mainly the one generated by an octree grid generation.

3 BOUNDARY PROJECTION AND SMOOTHING

Pattern extraction rely solely on the grid, and doesn't match the boundary in anyway. To obtain the final mesh, we slowly move vertices to the boundary and feature edges (see Fig.7).

3.1 Boundary matching

The patterns have a clear advantage: each vertex introduced has a clear preference on the boundary. As shown in Fig.8, if a vertex is on a edge on of a voxel, it has clear candidate for projection. A standard rule can be applied for feature points in voxel, feature edges crossing voxel' facets, and finally triangles cutting the voxel. In the code, we intersect the domain, the boundary and the features edges with the grid, and this give use wishes on each voxel. Those wishes are matching for neighbors voxels.

Those wishes are marked depending of the type: boundary, feature edge, and feature point. For edges and faces, we grow a small region (of lines for the edge case) around the wish. This way, we can adapt the projection through each smoothing iteration by querying the projection as the closest point in that small region.

Wishes near grid nodes will introduce a difficult configuration to smooth: two vertices of the mesh will be close to each other, and maintaining good quality while moving the boundary point to its wish will require a high number of iterations. Currently, the chosen grid is regular, which leads to a lot of local issues where convergence is slow. A good choice of grid would move nodes to make sure that they are not to close to the domain's boundary.

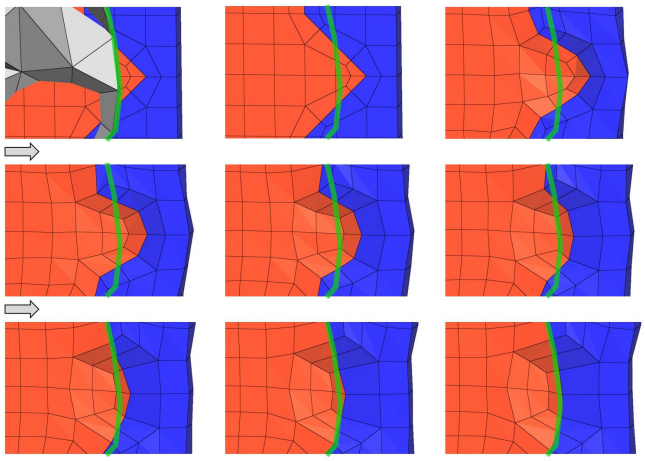


Fig. 7. Vertices of the mesh are slowly moved so that the boundary matches the domain.

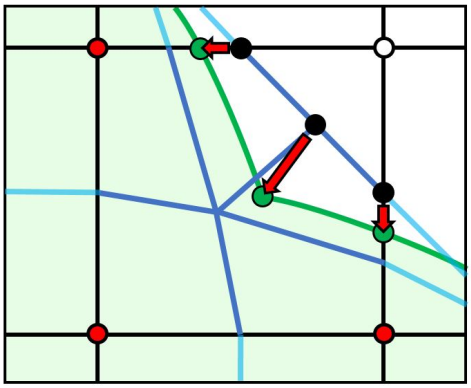


Fig. 8. Vertices (in black) of the hexahedral mesh have clear candidates (in green) for projection.

3.2 Local smoothing

To improve the boundary matching and keep the quality at the same time, we define an energy that give us a guess on the mesh quality. Each corner of an hexahedra defined a basis (the one used to compute the scale Jacobian). This basis provide a tetrahedron that must not be inverted. On this set of elements, we use the energy of [Garanzha et al. 2021]. This energy as several great properties:

- If no elements are inverted, inverting an elements gives an infinite energy.
- If there are inverted elements, it defines a sequence which will untangle them.
- It is well behaved (Positive definite Hessian).

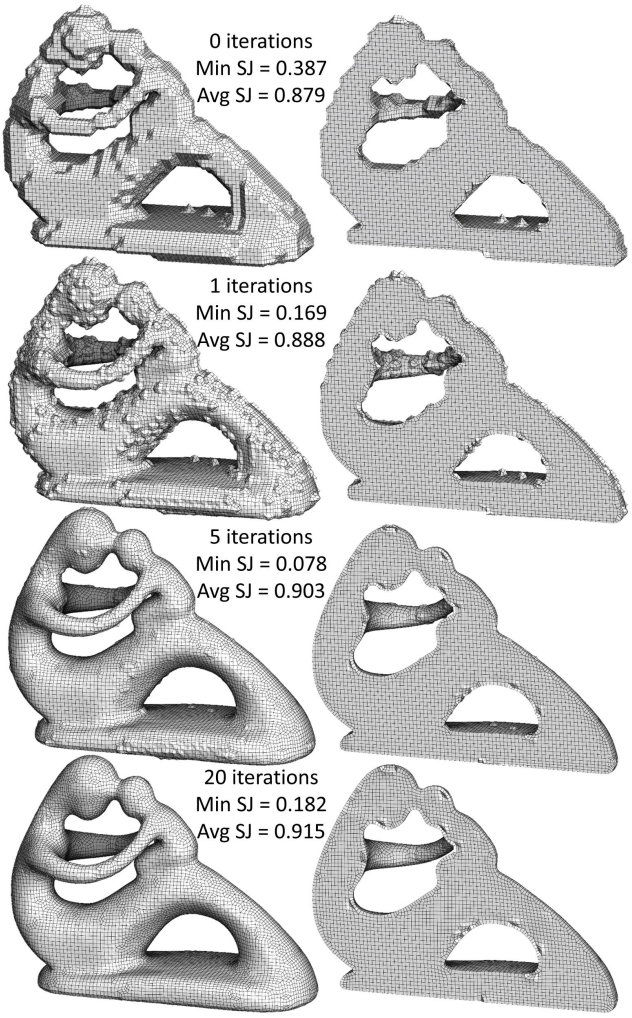


Fig. 9. Hexahedral meshing of Fertility model using a regular grid.

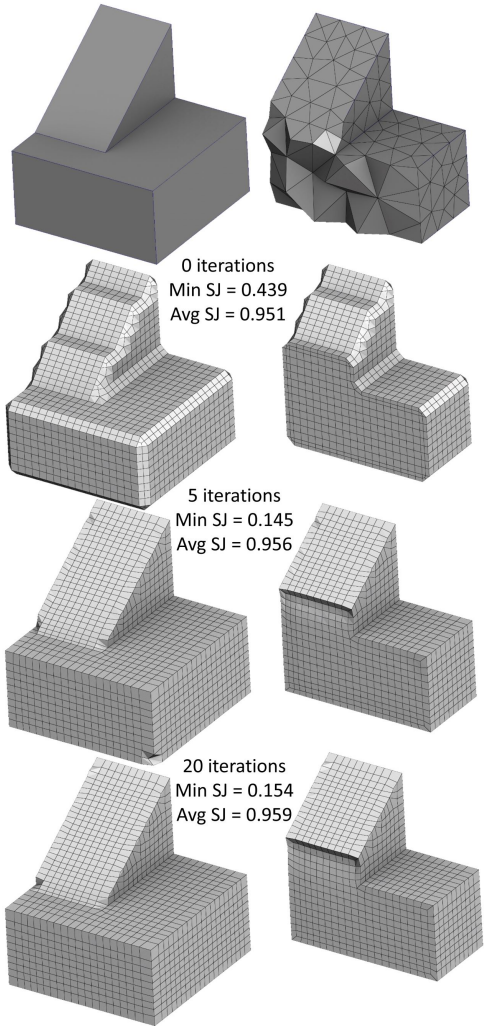


Fig. 10. Hexahedral meshing of a CAD model with feature edges using a regular grid.

Our process goes as follow: we move each points independently, with a newton iteration, with the constraint that we move only if we improve the local quality. Then, if the point needs a projection, we find the closest candidate, and move the furthest we can while not flipping elements.

If we start from a valid mesh, this process is guaranteed not to invert any elements. It will still improve the quality of the mesh if it start inverted, as the energy uses a good regularization. To guarantee any inversion, a complex set of tetrahedra must be chosen, which is a trade-off between guarantees and performance. We found that a set of 32 can prevent any inversion in practice, but is slow, see Subsec.4.1.

Using a local approach enable a robust boundary projection. Moreover, we currently avoid parallelization to make sure that two close points will not be moved together, introducing inverted elements. A lot of improvements could be done in this matter, but the code as the advantage of being simple.

4 RESULTS

4.1 Timings

All timings are done using *https://github.com/fprotais/marchinghex* on a 6 cores i7-8850 using MSVC. The matching between the domain and the grid is fully parallelizable, and our implementation can generate several millions hexahedra per minute. The clear bottleneck is smoothing. Currently, the code averages 5000 vertices per second per iteration, and to obtain good results, at least 10 iterations are required.

4.2 Quality

Fig.9 displays a hexahedral mesh obtained on a smooth domain with our algorithm on a regular grid. We can see that 20 iterations of smoothing will be sufficient to obtain a good looking mesh. Fig.10 shows that our algorithm does a good job at preserving features

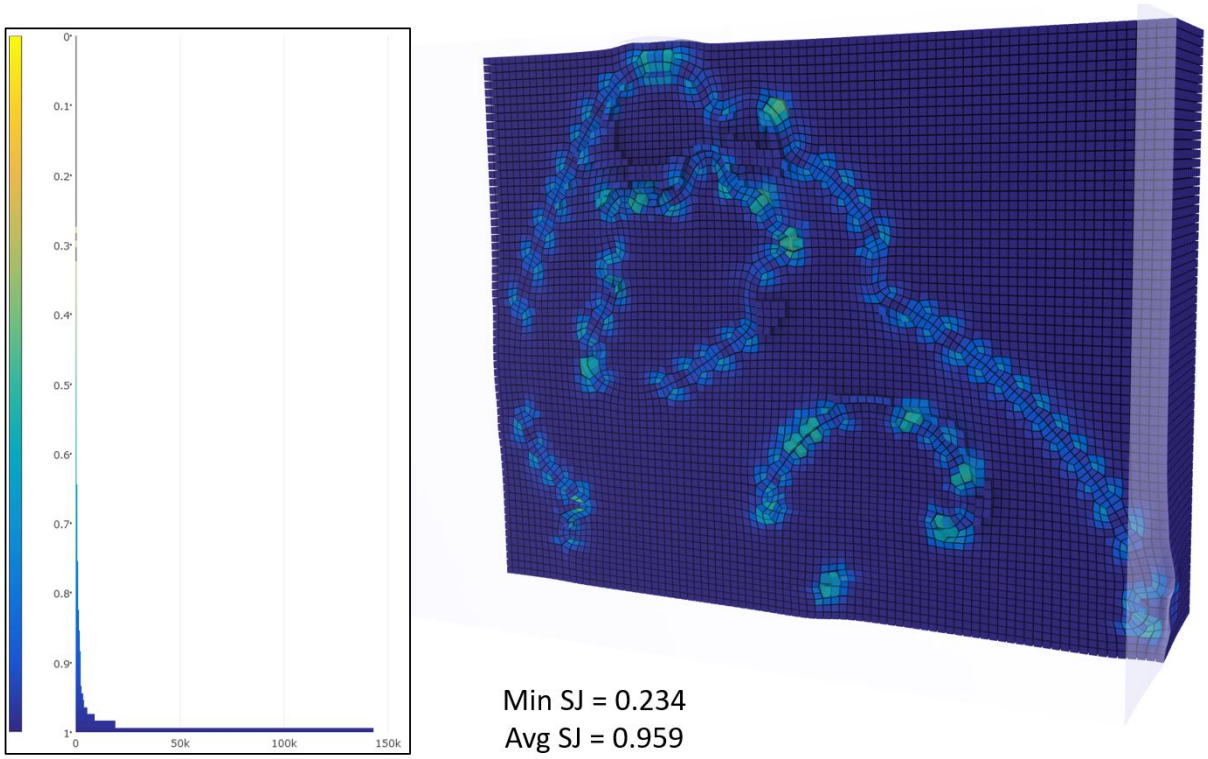


Fig. 11. Bi-material hexahedral mesh of Fertility model with its element quality.

on CAD meshes. Note that to be perfect along a feature edge, it would requires to introduce a feature edge padding, as described in [Maréchal 2016]. This procedure is quite gruesome, and we hope to find a better alternative that is global.

While the smoothing converges well on simple cases, it may not for difficult one's. More precisely, cut introduced in the previous section can introduce difficult settings to smooth, and the algorithm will sometimes need a lot of costly iterations. This is a yet to be solved issue, and we hope that a better choice of grid will solve those problems.

4.3 Grid variations

We were not able to test grid variations. We hope to test on octree grid and on grid obtained through global parametrization that doesn't fully match with the boundary.

4.4 Bi-material meshing

A major advantage of our method is that, on the grid, it gives a compatible inside and outside mesh. This is a interesting property for a lot of application such as fluid dynamic or geology. We give an example of this in Fig. 11 (same model as the teaser).

CONCLUSION

We provide an simple and robust method to extract a hexahedral mesh from an overlay grid. The mesh will match with the boundary and feature edges, and, under some constraints on the grid, will only contains valid elements. Our method still has issues such as speed or difficulty to match difficult boundaries.

REFERENCES

Guido Dhondt. 2001. A new automatic hexahedral mesher based on cutting. *Internat. J. Numer. Methods Engrg.* 50, 9 (March 2001), 2109–2126. https://doi.org/10.1002/nme.114

Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. 2019. Feature Preserving Octree-Based Hexahedral Meshing. *Computer Graphics Forum* 38, 5 (Aug. 2019), 135–149. https://doi.org/10.1111/cgf.13795

Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. 2021. Foldover-free maps in 50 lines of code. *ACM Transactions on Graphics* 40, 4 (July 2021), 102:1–102:16. https://doi.org/10.1145/3450626.3459847

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. *ACM Trans. Graph.* 21, 3 (July 2002), 339–346. https://doi.org/10.1145/566654.566586

Patrick Knupp. 2012. Introducing the target-matrix paradigm for mesh optimization via node-movement. *Engineering with Computers* 28, 4 (Oct. 2012), 419–429. https://doi.org/10.1007/s00366-011-0230-1

Marco Livesu, Luca Pitzalis, and Gianmarco Cherchi. 2022. Optimal Dual Schemes for Adaptive Grid Based Hexmeshing. *ACM Transactions on Graphics* 41, 2 (April 2022), 1–14. https://doi.org/10.1145/3494456

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (Aug. 1987), 163–169. https://doi.org/10.1145/37402.37422

Loïc Maréchal. 2009. Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features. In *Proceedings of the 18th International Meshing Roundtable*, Brett W. Clark (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 65–84. https://doi.org/10.1007/978-3-642-04319-2_5

Loïc Maréchal. 2016. All Hexahedral Boundary Layers Generation. *Procedia Engineering* 163 (2016), 5–19. https://doi.org/10.1016/j.proeng.2016.11.007

Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean Remacle, and Marco Livesu. 2022. Hex-Mesh Generation and Processing: A Survey. *ACM Trans. Graph.* 42, 2 (Oct. 2022), 16:1–16:44. https://doi.org/10.1145/3554920

Mark A. Yerry and Mark S. Shephard. 1984. Automatic three-dimensional mesh generation by the modified-octree technique. *Internat. J. Numer. Methods Engrg.* 20, 11 (Nov. 1984), 1965–1990. https://doi.org/10.1002/nme.1620201103