FORMAL DEFINITION OF HEXAHEDRAL BLOCKING OPERATIONS USING N-G-MAPS

V. Postat[2,3] N. Le Goff[1,2] S. Calderan[1,2] F. Ledoux[1,2] G. Hutzler[3]

[1]*CEA, DAM, DIF, F-91297, Arpajon, France*
[2]*LIHPC, CEA, Université Paris-Saclay, France*
[3]*COSMO, IBISC, Université d'Évry Val d'Essonne, Paris-Saclay, France*

ABSTRACT

Nowadays for real study cases, the generation of full block structured hexahedral meshes is mainly an interactive and very-time consuming process realized by highly-qualified engineers. To this purpose, they use interactive software where they handle and modify complex block structures with operations like block removal, block insertion, O-grid insertion, propagation of block splitting, propagation of meshing parameters along layers of blocks and so on. Such operations are error-prone and modifying or adding an operation is a very tedious work. In this work, we propose to formally define hexahedral block structures and main associated operations in the model of n-dimensional generalized map. This model provides topological invariant and a systematic handling of geometric data that allows us to ensure the expected robustness.

Keywords: Computational geometry, Quad and Hex meshes, n-G-map, Sheet operation

1. INTRODUCTION

Many high-fidelity numerical simulation fields modeling impact, fluid dynamics, shocks, crash or hydrodynamics require or preferred to use hexahedral block-structured meshes for getting the expected numerical and physical results. Such meshes are efficient in highly anisotropic physical simulations (boundary layers, shockwaves, etc.), as the associated tri-linear basis has cubic terms that capture higher order variations and provide less elements, reducing simulation time. They are also very interesting in terms of performances: (1) unlike unstructured meshes, most of the mesh connectivity in a block-structured hexahedral mesh can be implicitly deduce from an underlying multi-dimensional structure. This array structure is also useful to ease fast access to adjacent node and cells considering the indexed structure of arrays.

While research on hexahedral meshing is a very active domain [1], real-case hexahedral block-structured mesh are still generated using interactive tools [2, 3, 4], which is a very-time consuming process realized by highly-qualified engineers.

They also have to provide a way to link the block structure and the final mesh to the representation of the domain Ω to be meshed. In this work, we consider CAD shapes that are represented with the BRep model [5], or *boundary representation* where the CAD shape is represented by its boundary. In order to provide a robust and reliable software, we need to be very versatile on the CAD model we consider. They often lack of accuracy as they are designed for manufacturing first (non-watertight, often only 10^{-4} arithmetic precision), or contain excessive details that are not essential for CFD/FEM analysis.

In this paper, we focus on the representation of the block structure and its link to the CAD model representation. It is the major component of an hexahedral meshing software and it must be designed to be robust and reliable. Such a component has to provide functionalities to handle and modify complex block structures with operations like block removal, block insertion, O-grid insertion, propagation of block splitting, propagation of meshing parameters along layers of blocks and so on. Such operations are error-prone and modifying or adding an operation is a very tedious work.

In this work, we focus on the reliability of the underlying representation model of those structures. We formally define hexahedral block structures and main associated operations in the model of n-dimensional generalized map [6]. This model provides topological invariant and a systematic handling of geometric data that ensure the software has the expected robustness.

1.1 State of the art

The representation of a block structure of is halfway between representing a mesh, which can contain millions of cells, and representing the topology of a geometric model composed of a few thousand volumes at most. A direct approach to representing a n-dimensional mesh is to define it as a n-tuple of cell sets, each set containing the cells of a given dimension, and the incidence/adjacency relations that are useful for the application to be developed. This is provided by generic meshing libraries [7, 8, 9, 10, 11, 12] that **explicitly represent** the mesh cells. If we consider the traditional representations used in computational geometry, the representations used in these libraries derive from the classical model of *incidence graphs* [13].

Some other libraries are based on abstract core entities which **implicitly represent** the mesh cells, such as winged edges [14], doubly Connected Edge List [15], half-edge data structure [16], surface meshes [17] or Combinatorial maps [6, 18]. The latter can be seen as a generalization of half-edge meshes which can handle higher dimensional cell complexes. Some other representations go further in the cell splitting and provide abstract core entities that are relative to vertices. Corner Tables[19] are based on vertices and are especially interesting for real time rendering. We consider the usage of n-dimensional generalized maps. Eventually n-dimensional generalized map [6] represent non-orientable and open subdivisions. Comparisons for these various data structures can be found in [18].

Our work focuses on using n-dimensional generalized maps to describe hexahedral block structures and associated operations. We believe that this model allows us to formally and rigorously defined those operations. Several previous works [20, 21, 22, 23] provided formal definition for hexahedral operations, mainly acting on sheets. A first benefit of our work is to have implementations that are equivalent to the formal definitions. A second benefit is to rigorously consider topological and geometric aspects.

1.2 Main contributions and outlook

In order to provide reliable and robust data structures and operations to handle and modify hexahedral block structures in interactive CAD-meshing software, we use n-dimensional generalized maps to represent the block structure. In this context, the main contributions of this work are: (1) We formally define how to represent the topology of the block structure and the automatic link to a BRep geometric model (see Section 2); (2) We formally define the minimal set of operations to edit hexahedral block structures (see Section 3). Those operations are the selection of sheets, the removal of sheets and the insertion of sheets; (3) We provide pseudo-code algorithms for every operations in Section 3; (4) Set of operations: Split a series of blocks, remove a series of blocks.

With the n-G-map model, for each operation, we are able propose a definition and an algorithm, which is unique whatever the dimension is (2 or 3).

2. REPRESENTING BLOCK STRUCTURES WITH N-G-MAPS

Unlike meshes, block structures contain a limited number of blocks that rarely exceeds a few thousand elements, making memory footprint and data access performances not a limiting factor in practice. Getting the right data representation for block structures raises some other specific constraints and issues.

CAD classification. Each block cell is classified onto one or several geometrical entities. Each i-dimensional block entity is classified, with $0 \leq i \leq 3$ is classified onto at least one j-dimensional geometric entity. Some rules must be ensured and checked as pre- and post-conditions before performing edition tasks. The CAD classification is essential for the meshing process.

Mesh algorithms parameters. Block cells carry the meshing parameters of each block. Among them, some parameters are local to a face side. In other words, for a face shared by two blocks, a parameter can be given to each side of the face; it can be geometric discretization laws parameters, orthogonality conditions. Implicit representations, like n-G-map are then a better option that explicit representation that do not usually store such half-faces.

Block geometric representation. It is simple and traditional to represent blocks as linear cells complexes, but this representation meets numerous issues and we need to represent and handle non-linear representations, where blocks are represented by boundary cells that are high-order polynomials. The separation of geometry and topology concerns in the n-G-map model makes possible to change the geometrical representation while keeping the topology part unchanged.

2.1 N-G-map to represent topology

The model of n-dimensional generalized maps, or n-G-maps for short, uses *darts* as core elements. Darts are purely abstract entities, that do not embed any geometrical information. Intuitively, n-G-maps come from decomposing n-dimensional objects into topological cells. Let us consider the 2D object of Fig. 1-a, which is first split into faces in Fig. 1-b connected along their common edge with a 2-link. The link is noted 2 as it connects two faces, which are 2-dimensional faces. Similarly, faces are split into edges connected with the 1-link (see Fig. 1-c) and are split into vertices by the 0-link to obtain the 2-G-map of Fig. 1-d. Vertices obtained at the end of the process do not have the meaning of those in explicit representations. They are the darts and in 2D, each of them corresponds to a "*vertex locally to an edge, itself locally to a face*". In other words, a 2D dart is a triplet *(vertex, edge, face)*[1]. The different i-links are labeled arcs, where i depicts a dimension that belongs to $[\![0; n]\!]$ for a n-G-map. They are functions mapping darts to others and they allow to retrieve usual cells. Let us now formally define the n-generalized maps, which represent n-dimensional manifolds, orientable or not, and with or without boundary [25, 26, 6].

Definition 2.1 (Generalized map) *Let $n \geq -1$, a n-dimensional generalized map, or n-G-map, is an algebra $(D, \alpha_0, \dots, \alpha_n)$ such that D is a finite set of darts such that*

$$\forall i, 0 \leq i \leq n, (\alpha_i)^2 = id, \tag{1}$$

$$\forall i, j, 0 \leq i < i + 2 \leq j \leq n, (\alpha_i \circ \alpha_j)^2 = id. \tag{2}$$

The definition starts from dimension -1 to represent the empty G-map that only contains unconnected darts. Mapping functions α_i are involution[2] and condition (2) adds extra constraint that ensure to represent manifold cellular complexes only. For instance in 2D, it implies that $\alpha_0 \circ \alpha_2$ is an involution. It means that if two darts d and d' are linked by α_2, then the darts $\alpha_0(d)$ and $\alpha_0(d')$ are linked by α_2 too. By this way, two adjacent faces share a full edge and not just a part of it (see Fig. 2).

A dart $d \in D$ is i-free if α_i(d)=d. All the boundary darts of a n-G-map are n-free. In the remainder figures of the paper, we represent α_0 links with dot lines between two dart (), α_1 links with two circles () and α_2 links with two segment lines ().

If darts are the core elements of a n-G-map, a cellular complex is made of cells, that we want to retrieve to write many algorithms and applications. In our case, it is mandatory to get access to vertices, or 0-cells, edges, or 1-cells, faces, or 2-cells, and regions, or 3-cells. In the n-G-map representation, i-cells are a special case of orbits.

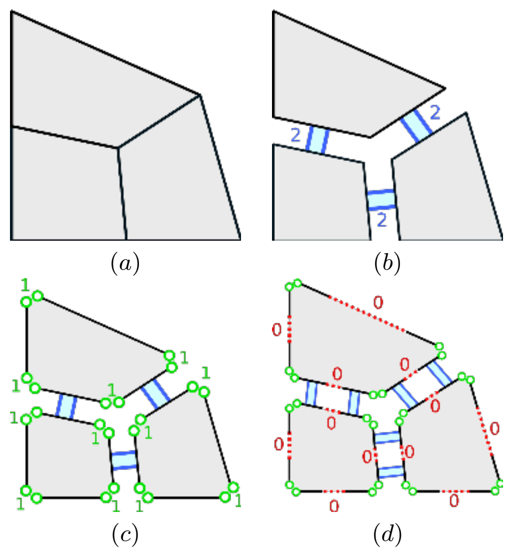


Figure 1: *Decomposition of a 2-dimensional cellular complex into a set of darts: In* (a)*, the initial complex; In* (b)*, it is split into faces; In* (c)*, every face is split along its edges; Finally, each edge locally to a face is split into its two end vertices* (d).

Definition 2.2 (orbit) *Let $\Phi = \{f_1, \dots, f_n\}$ a set of permutations*[3] *on a set E. The orbit of $e \in E$ relatively to Φ is the subset $<\Phi> (e)$ of E such that*

$$<\Phi> (e) = \{\phi(e) | \phi \in <\Phi>\}.$$

The orbit of an element $e \in E$ is made of the elements of E that can be reached by any composition of permutations of ϕ and their inverses. In the n-G-maps model, α_i functions are involution, i.e. a special case of permutations. And vertices, edges, faces and regions are special cases of orbits. For instance, considering the 2-G-map represented on Fig. 3.(a) and the dart d, the orbit $<\alpha_0, \alpha_1>$ (d) gathers all the darts that belong to the blue face. In (b), the orbit $<\alpha_0, \alpha_2>$ (d) corresponds to the darts of an edge, while in (c), the 0-cell that contains the dart d is the orbit $<\alpha_1, \alpha_2>$ (d). Formally, in a n-G-map, we define i-cells as follows.

Definition 2.3 (i-cell) *Let $G = (D, \alpha_0, \dots, \alpha_n)$ be a n-G-map, $d \in D$ and $i \in \{0, \dots, n\}$; the i-cell that contains d is the orbit*

$$<\widehat{\alpha_i}> (d) = <\alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n> (d).$$

[1] A dart is equivalent to a cell-tuple as defined in [24].

[2] A function f is an involution if and only if $f^2 = id$.

[3] f is a permutation on E iff $\forall e \in E, \exists k > 0 / f^k(e) = e$.

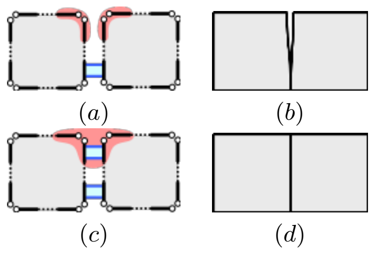


Figure 2: On the first line, we do not enforce $\alpha_0 \circ \alpha_2$ to be an involution; quad faces are partially connected, they only share one vertex. On the second line, $\alpha_0 \circ \alpha_2$ is an involution and the quad faces are wholly glued along the edge.

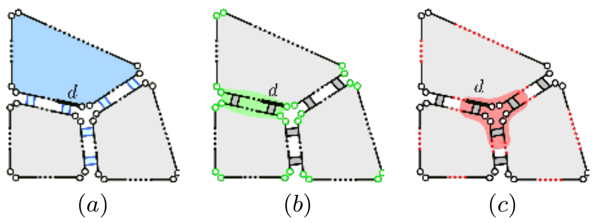


Figure 3: *Definition of i-cells in the G-maps. In* (a)*, the* 2*-cell that contains the dart d is the orbit* $<\alpha_0, \alpha_1>(d)$*; In* (b)*, the* 1*-cell that contains the dart* d *is the orbit* $<\alpha_0, \alpha_2>(d)$*; In* (c)*, the* 0*-cell that contains the dart* d *is the orbit* $<\alpha_1, \alpha_2>(d)$*.*

We can observe that applying an involution α_i onto a dart allows to go from an i-cell to another i-cell. On the contrary, applying an involution $\alpha_j, j \neq i$ makes stay in the i-cell.

For sake of readability, we note $\alpha_{ij\ldots k}$ the composition of functions. $\alpha_k \circ \ldots \circ \alpha_j \circ \alpha_i$, and by extension, we note $d\alpha_{ij\ldots k} \cong (\alpha_k \circ \ldots \circ \alpha_j \circ \alpha_i)(d)$. It allows us to read from left to right each traversal made of a series of α functions. For instance the dart $d\alpha_{12021}$ will be the dart obtained by successively applying α_1, α_2, α_0, α_2 and α_1 starting from d.

2.2 Orbits and geometry classification

Vertices, edges and faces as previously defined are only topological entities. To write algorithms, but also to associate a geometrical representation, they have to carry some pieces of data. In particular, in the context of block generation for CAD models, we need: (1) to embed/represent the topological model of n-G-map into a geometrical space and (2) to assign block entities to CAD entities. Such data assignment can be handled in the n-G-map model by defining a mapping function for each type of orbits.

For our purpose, we limit those mappings to the 0, 1, 2 and 3-cells. Let us consider a n-G-map $G = (D, \alpha_0, \ldots, \alpha_n)$ and a BRep model $M = (V, S, C, P)$ that is a set of volumes V, that are bounded by surfaces of S, curves of C and points in P. This set definition is very weak and allow us to consider non-watertight models. For each dimension $i \in [\![0; n]\!]$, we define the *geometric classification* function $gc_i : D \to S_M$, where $S_M = V \cup S \cup C \cup P \cup \emptyset$, such that:

$$\forall d \in D, \forall d' \in <\widehat{\alpha_i}>(d), gc_i(d') = gc_i(d), \qquad (3)$$

$$\forall d \in D, i \leq dim(gc_i(d')). \qquad (4)$$

Condition 1 ensures that the gc_i function assigns all the darts of an i-cell to the same geometrical entity. And with condition 2, we indicate that the dimension of this geometrical entity is greater that i. For instance a block edge can be classified onto a curve, surface or a volume of M but not onto a point.

In the context of this work, we represent blocks linearly, that is, we assign a point of $I\!R^n$ to each 0-cell, and we deduce the geometry of edges, faces, and volumes linearly. More formally, for a n-G-map $G = (D, \alpha_0, \ldots, \alpha_n)$ we define the *embedding* function $e : D \to I\!R^n$ such that:

$$\forall d \in D, \forall d' \in <\widehat{\alpha_0}>(d), e(d') = e(d). \qquad (5)$$

This property is similar to Equation 3 for the gc_i functions. We ensure the consistency of the embedding: all the darts of a 0-cell are assigned on the same geometrical point. Note that the geometric classification only allows us to update the link to the CAD model during blocking operations. It does not intrinsically handle CAD inaccuracies (gaps, surface intersections, null-size curves, etc.), but could help.

2.3 Atomic modification operations

There exist four atomic operations that we can perform in an n-G-map $G = (D, \alpha_0, \ldots, \alpha_n)$. Each of this operation will have a potential impact on the geometrical classification.

Dart creation The first operation consists in adding a new dart d in D. For all $i \in [\![0; n]\!]$, we have $d\alpha_i = d$, $g_i(d) = \emptyset$ and $e(d) = (0, 0)$ in 2D and $(0, 0, 0)$ in 3D. In other words, d is totally unconnected to the other darts.

Dart removal Removing a dart d from D, will have an impact on the mapping functions $\{\alpha_i\}_{i=0..n}$. First of all, for all $i \in [\![0; n]\!]$, the dart that is i-linked to d becomes i-free. Second, as G remains an n-G-map after removing d, the second condition of Def. 2.1 induces that some other darts will be i-free after removing d. For instance, removing a dart d from a 2-G-map implies that $d\alpha_2$ will become 2-free, but also the darts $d\alpha_0$ and $d\alpha_{02}$.

Dart unsewing We i-*unsew* a dart d when we set $d\alpha_i = d$. Like for the dart removal, as G remains an n-G-map after i-unsewing d, the second condition of Def. 2.1 implies that other darts are i-free after i-unsewing d.

Dart sewing If we consider a couple of i-free darts $(d, d') \in D \times D$, then i-sewing d and d' means to have $d\alpha_i = d'$. The first condition of Def. 2.1 implies that $d'\alpha_i = d$ and the second condition implies that some other links are created.

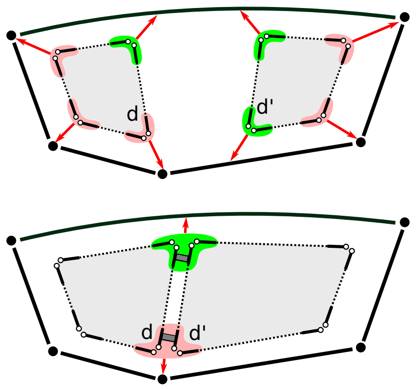


Figure 4: A 2-G-map is classified onto a geometric model. On the top, darts of 0-cells that are classified on geometric points are colored in red, while those classified on curves are colored in green. On the bottom, darts d and d' are 2-sewed and their 0-cells are fused.

Sewing darts has a very strong impact on functions gc_i and e too. Equations 3 and 5 enforces those functions to return the same value for darts that belongs to same cells. For instance, sewing two darts d and d' by α_2 in Figure 4 merge the cells $<\widehat{\alpha_0}>(d)$ and $<\widehat{\alpha_0}>(d')$ into a single one. It has an impact both on functions e and gc_0. Both function gave different values before sewing for the darts of cells $<\widehat{\alpha_0}>(d)$ and $<\widehat{\alpha_0}>(d')$. After the sewing operations, all those darts belongs to the same 0-cell. And so functions e and gc_0 must give the same values for those darts now.

Ensuring the consistency of functions gc_i and e when sewing operations are performed is automatically done by providing *merging rules* that depends on the intrinsic semantics of gc_i and e. We apply the following rules. When merging two i-cells c_i^1 and c_i^2 to create the cell c_i^f, then with $d \in c_i^1$ and $d' \in c_i^2$, we have:

- If $gc_i(d) = gc_i(d')$ then for all $d" \in c_i^f, gc_i(d") = gc_i(d)$ and $e(d")$ is the projection of $\frac{e(d)+e(d')}{2}$ on the geometrical entity $gc_i(d)$;
- If $dim(gc_i(d)) < dim(gc_i(d'))$ then for all $d" \in c_i^f, gc_i(d") = gc_i(d)$ and $e(d") = e(d)$;
- If $dim(gc_i(d)) > dim(gc_i(d'))$ then for all $d" \in c_i^f, gc_i(d") = gc_i(d')$ and $e(d") = e(d')$.

Otherwise it means that the two i-cells are classified on distinct geometrical entities of same dimension and the sewing operations is so impossible.

3. HEXAHEDRAL BLOCKING OPERATIONS

The block structure that we handle is full-hexahedral. As a consequence, updating the block structure consists in modifying the topology and geometry of a hexahedral mesh. We consider here two type of operations, which are sheet removal and sheet insertion [27]. In this section, we present each operation, the definition in the n-G-map model, and the corresponding pseudo-code algorithm. We begin with the sheet selection which gives us all the cells that belongs to a sheet.

3.1 Sheet selection

We equally consider a 2D quad block structure or a 3D hexahedral block structure. We define a sheet S, or layer of cells, as a subset of cells (quads in 2D, hexes in 3D). Starting from an edge e of the mesh, we define E_e as being the smallest subset of E that verifies:

$$e \in E_e \text{ and } \forall e_i \in E_e \Rightarrow E_{e_i}^{/\!/} \subseteq E_e,$$

with $E_{e_i}^{/\!/}$ the set of edges opposed to e_i in the cells that are incident to e_i. The sheet S_H is the set of cells that are incident to an edge of E_e at least. Examples of 3D sheets are given on Figure 5 where we can see three types of sheets: in (a), a simple sheet is depicted; in (b) the sheet intersects itself along a complete chord of hexahedral cells; in (c), the sheet touches itself along several faces. Second and third sheets are respectively qualified as being *self-intersecting* and *self-touching*.

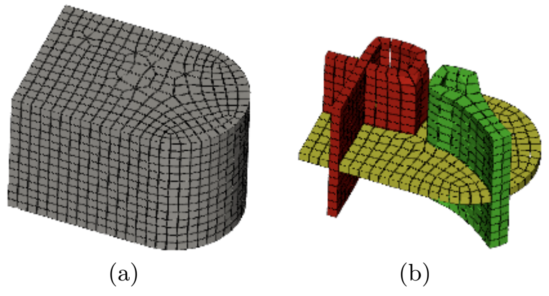


Figure 5: *Example of 3D sheets in a hexahedral mesh. In (a), the full mesh, in (b) three sheets are represented: a regular sheet (yellow), a self intersecting sheet (red) and a self-touching sheet (green).*

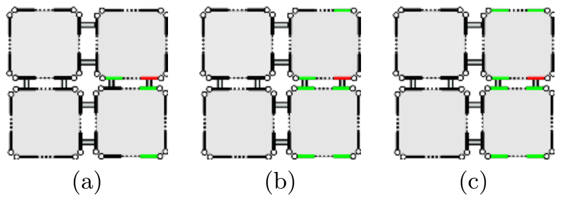


Figure 6: *Building opposite sheet dart set S_d (see Algorithm 1). Starting from a first marked dart (in red) the front F is propagated via the alpha-links described lines 5,6 and 8 and the darts are then added to S_d. (a, b and c) represent S_d after several iterations.*

In the n-G-map model, the edge selection process consists in picking a dart d, that defines the edge $<\widehat{\alpha_1}>(d)$ and then the set of edges that are topologically "opposite" to $<\widehat{\alpha_1}>(d)$. The expected set of darts, called S_d is given by Definition 3.1. This set of dart consists in a specific orbit where all the darts of an edge are reached, using mappings α_0, α_2 in 2D and mappings $\alpha_0, \alpha_2, \alpha_3$ in 3D, and mapping $\alpha_{(n-1)\ldots 0 \ldots (n-1)}$ allows us to jump from an edge to another one.

Definition 3.1 (Opposite sheet dart set) *Let $G = (D, \alpha_0, \ldots, \alpha_n)$ be a structured n-G-map, with $n = 2$ or 3, and $d \in D$, the opposite sheet dart set defined by d is the orbit $<\alpha_{n(n-1)\ldots \widehat{0 \ldots (n-1)}}, \alpha_0, \alpha_2>(d)$.*

In this definition and in the remainder, all the n-cells of a structured n-G-map are quadrilaterals for $n = 2$ and hexahedra for $n = 3$. The whole set of darts that belongs to the n-cells of the sheet expanded from dart d is given by getting the n-cells of the darts that belong to S_d. Starting from a dart d, it indicates in 2D that dart $d\alpha_{2101}$ belongs to the sheet set too. In 3D, it is the mapping α_{321012} that allows us to reach an opposite edge. Algorithm 1 gives us a naive but straightforward implementation to build the set of darts S_d both in 2D and 3D.

Algorithm 1: Opposite Sheet dart set

```
Data: A structured $n$-G-map
      $G = (D, \alpha_0, \ldots, \alpha_n)$ and a dart $d \in D$
Result: $S_d$ a set of darts
1  $F \leftarrow \{d\}$;
2  while $F \neq \emptyset$ do
3      $d_i \leftarrow F.first()$; // Pop up the head of $F$;
4      if $d_i \notin S_d$ then $S_d \leftarrow S_d + \{d_i\}$;
5      if $d_i\alpha_0 \notin S_d$ then $F \leftarrow F + \{d_i\alpha_0\}$;
6      if $d_i\alpha_2 \notin S_d$ then $F \leftarrow F + \{d_i\alpha_2\}$;
7      if $d_i\alpha_{n(n-1)\ldots 0 \ldots (n-1)} \notin S_d$ then
8          $F \leftarrow F + \{d_i\alpha_{n(n-1)\ldots 0 \ldots (n-1)}\}$;
9      end
10 end
```

3.2 Sheet collapse

Collapsing a sheet consists in removing it entirely from the block structure. Figure 7 illustrates the overall procedure in a simple case where the sheet represented by red quadrilateral faces in Fig. 7-a will be totally removed in Fig. 7-f. We define the resulting n-G-map as follows:

Definition 3.2 (Sheet collapse) *Let $G = (D, \alpha_0, \ldots, \alpha_n)$ be a structured n-G-map, with $n = 2$ or 3, $d \in D$, and S_d, the the opposite sheet dart set defined by d. Collapsing the sheet defined from d gives the n-G-map $G' = (D', \alpha'_0, \ldots, \alpha'_n)$ such that:*

1. *$D' = D - <\widehat{\alpha_n}>(S_d)$,*
2. *$\forall i \in [\![0; n-1]\!]$, $\forall d' \in D'$, $d'\alpha'_i = d'\alpha_i$,*
3. *$\forall d' \in D', d'\alpha'_n = \begin{cases} d'\alpha_n & \text{if } d'\alpha_n \notin <\widehat{\alpha_n}>(S_d), \\ d_k\alpha_n & \text{otherwise,} \end{cases}$*

   with
   $$d_k = d'(\alpha_{n(n-1)..0..(n-1)})^k$$
   and k the smallest positive integer such that
   $$d'(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n \notin <\widehat{\alpha_n}>(S_d).$$

This definition deserves some comments. Item 1 gives the set of darts composing the new n-G-map as being the initial darts without the darts of $<\widehat{\alpha_n}>(S_d)$ as defined by Definition 3.1. Item 2 indicates that mappings α_0 to α_{n-1} are unchanged. Only the mapping α_n is modified for the darts that belong to $D - <\widehat{\alpha_n}>(S_d)$ and that are n-linked to a dart of $<\widehat{\alpha_n}>(S_d)$ (see Fig. 7). After the collapse, such darts are n-linked to the first dart that do not belong to $<\widehat{\alpha_n}>(S_d)$ and that we can reach by applying a composition of mappings $(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n$ with $k > 0$. This composition of mappings is mandatory to collapse self-touching sheets. Let us note that the sheet to collapse can be a boundary sheet (see Figure 8-a). In this case, the dart reached by applying $(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n$ on a dart d is d itself. It is not an issue from a topological point of view but raises some concerns for the geometry preservation. It should require a careful update of the dart attribute.

In order to avoid a specific case to handle attribute when we collapse a boundary sheet, the algorithm we develop follows a two-stage approach after initialization phase, shown in Algorithm 2:

- **Initialization** - We first define all the darts that will be used in the collapse; Line 1, we get the opposite darts in green in Fig. 7. From lines 6 to 10, we mark the darts of the n-cells that are in the sheet. Then we retrieve the red darts by applying

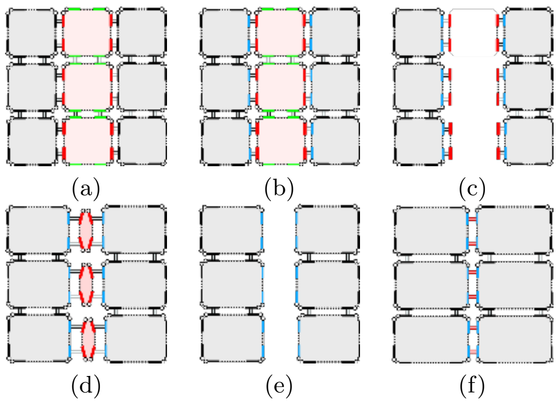


Figure 7: Darts of S_d are colored in green then extended with red darts to get all the darts of $<\widehat{\alpha_2}>(S_d)$ in (a); From this set of darts, we can reach blue darts highlighted in (b) using mapping α_2; Then green darts are removed from the G-map in order to free red dart of α_1 links in (c); The α_1 link is recreated between opposite red darts in (d); Red darts are the removed in (e) which free α_2 of blue darts that are sewed to the opposite dart in (f).

α_1 on green darts and we store the opposite red darts in the 1-cell. From the red darts, we get the blue ones in Fig. 7-b in the neighbor n-cell by applying α_2 in 2D or α_{232} in 3D and store the opposite blue dart by applying $(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n)$ until reaching an unmarked dart in the sheet. This corresponds to the item 3 of Definition 3.2 in lines 12 to 19.

- **Phase (1)** - Then starts the first stage of the collapse operation. We first remove darts of the opposite sheet darts (i.e. green darts) as shown in Fig. 7-c. This free α_1 link of remaining red darts (line 20). We can now 1-sew opposite red darts and thus create a n-cell in line 21 (in Fig. 7-d).

- **Phase (2)** - The second collapse stage starts by removing n-cells containing red darts in Fig. 7-e. With this step we verify item 1 of Definition 3.2 at line 22. As we have removed all darts in the sheet to collapse, blue darts are n-free. The last step consist in sewing them to their opposite blue darts (lines 23) to get the result shown in Fig. 7-f.

For the whole duration of the algorithm the only modifications on links of darts in the n-G-map not in the sheet to collapse are n-links of darts neighbors of the sheet. This verifies the item 2 of Definition 3.2.

This two-stage approach is designed to handle particular cases of sheet configurations as shown in Fig. 8: self-intersecting sheets in (a), self-touching sheets in (b) and boundary sheets in (c). Self-intersecting and self-touching sheets configurations are handled in the initialization phase. Line 14 ensures that no inner red darts is taken as blue dart and line 16 ensure that blue darts are getting the opposite one through multiple layers of self-touching sheet. The definition of the sewing operation coupled with the merging rules for functions gc_i and e (Section 2.3) ensures the geometry modification. In particular, we don't loose lower dimension geometric entities and thus allows us to handle the boundary collapse case as shown in Fig. 8-c where the red darts sewing snaps the right side on the left boundary.

Algorithm 2: Sheet Collapse

```
Data: n-G-map $H = (D, \alpha_0, \dots, \alpha_n)$, a dart
      $d \in D$
1  $S_d \leftarrow$ sheetSelection($d$) ;   /* Algorithm 1 */
2  $in\_sheet(x) : \{d \in D\} \rightarrow \{false, true\}$;
3  $\forall d \in D, in\_sheet(d) \leftarrow false$;
4  $collapse \leftarrow \emptyset$;
5  $to\_collapse(x) : \{d \in D\} \rightarrow \{d' \in D\}$;
6  for $d \in S_d$ do
7      $in\_sheet(<\widehat{\alpha_n}>(d)) \leftarrow true$;
8      $collapse \leftarrow collapse + \{d\alpha_1\}$;
9      $to\_collapse(d\alpha_1) \leftarrow d\alpha_{01}$;
10 end
11 $sew \leftarrow \emptyset$; $to\_sew(x) : \{d \in D\} \rightarrow \{d' \in D\}$;
12 for $d_c \in collapse$ do
13     $d_X \leftarrow d_c\alpha_X$; //$X = 232$ in 3D, $X = 2$ in 2D;
14     if $d_X \notin collapse$ then
15         $sew \leftarrow sew + \{d_X\}$; $k \leftarrow 1$;
16         while $in\_sheet(d_X(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n)$
            do $k \leftarrow k + 1$;
17         $to\_sew(d_X) \leftarrow d_X(\alpha_{n(n-1)..0..(n-1)})^k\alpha_n)$;
18     end
19 end
20 for $d \in S_d$ do remove($d$);
21 for $d_c \in collapse$ do 1-sew($d_c, to\_collapse(d_c)$);
22 for $d_c \in collapse$ do remove($<\widehat{\alpha_n}>(d_c)$);
23 for $d_c \in sew$ do $n$-sew($d_s, to\_sew(d_s)$);
```

We can apply Algorithm 2 to collapse sheet in 3-G-map in Fig. 9. The only change is in line 13 when we want to get the dart in neighbor n-cell. We define X where $X = 2$ in 2D and $X = 232$ in 3D and so we apply α_X, i.e. α_2 or α_{232}. This allow us to get the blue darts in Fig. 9-b in 3D the same way as in 2D.

3.3 Sheet insertion

Sheet insertion is much more complex to define than sheet collapse. The traditional way to insert sheets is to perform a *pillowing* operation [27]. After selecting a manifold connected set of n-cells P, called a *pillow set*, the pillowing operation consists in inserting a new

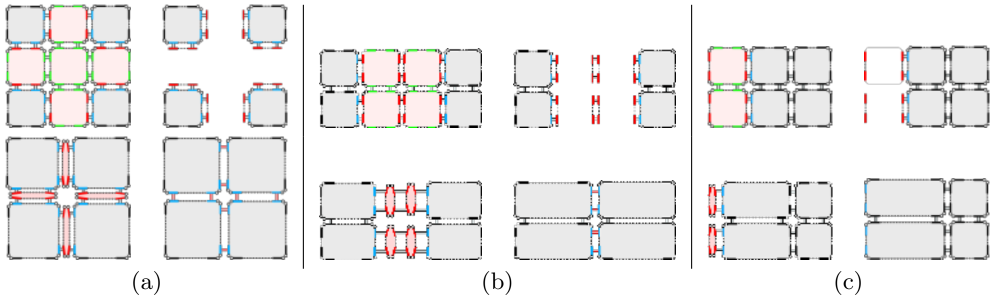


Figure 8: The four major steps of sheet collapse for self-intersecting sheets (a), self-touching sheets (b) and boundary sheets (c). On the top left, we define the darts to remove (green, red) and to collapse (blue); on the top right, we remove green darts; on the bottom left, we sew red darts and on the bottom right, we eventually sew blue darts.

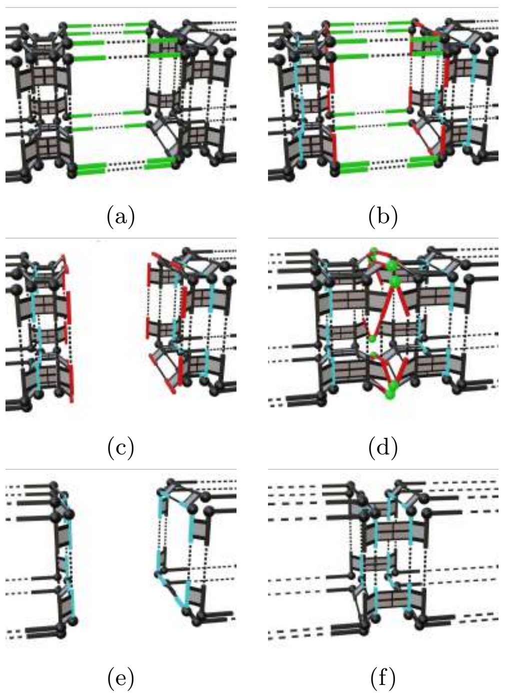


Figure 9: A close up view of a sheet collapse operation in 3D, following the same steps as Fig. 7.

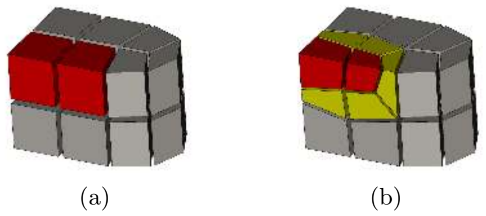


Figure 10: Example of pillowing operation where the red blocks are selected in (a). The pillowing operation isolates those two blocks from the other blocks and inserts a complete sheet in yellow (b).

layer of n-cells, i.e. a pillow, that surrounds P and isolate it from the remainder of the block structure (see Fig. 10). Having a set of hexahedral elements as an input does not allow us to insert self-intersecting and self-touching sheets. To this end, we propose to select a connected set of $(n-1)$-cells, or hyperplane, that verify several conditions. To express those conditions, we first introduce the ghost layered n-G-map. By analogy with ghost layer of cells used in HPC simulation for duplicating boundary cells across the computational units, thus avoiding useless interprocess communications, we enrich an hexahedral n-G-map with extra darts on the boundary. Those darts allow us to avoid dealing with particular cases for boundary elements.

Definition 3.3 (ghosted n-G-map) *Let $G = (D, \alpha_0, \ldots, \alpha_n)$ be a structured n-G-map, with $n = 2$ or 3 and $\partial_n D$ be the set of n-free darts of G. Let $\{f_i : \partial_n \to D_i\}_{i \in [\![1;n]\!]}$ be n mapping functions such that*

$$D_i \cap \partial_n D = \emptyset \text{ and } D_i \cap D_j = \emptyset, \forall (i,j) \in [\![1;n]\!]^2$$

with $i \neq j$. The ghosted n-G-map of G, noted G^g is the n-G-map $(D^g, \alpha_0^g, \ldots, \alpha_n^g)$ such that:

1. $D^g = D + D_1 + \ldots + D_n$,
2. $\forall d \in D,\ \forall i \in [\![0; n-1]\!],\ d\alpha_i^g = d\alpha_i$,

3. $\forall d \in D^g, d\alpha_n^g = \begin{cases} d\alpha_n & \text{if } d \in D - \partial_n D, \\ df_n & \text{if } d \in \partial_n D, \\ d & \text{if } d \in f_i(\partial_n D), i < n \end{cases}$

4. $\forall d \in \partial_n D, \forall i \in [\![1; n-1]\!], df_i\alpha_i^g = df_{i+1}$,

5. $\forall d \in f_1(\partial_n D), d\alpha_n^g = d'$, *with* $d' \in \partial_n D$ *and*

$$\exists k > 0 / d(\alpha_{n-1}^g \alpha_n^g)^k \alpha_{n-1}^g = d'$$

The ghosted n-G-map G^g enriches the n-G-map G by adding a pillow layer of partial n-cells around G. Figure 13.(b) illustrates the set of darts that has been added. For each boundary dart d of G, two darts d_1 and d_2 are added such that $d_1 = d\alpha_2$ and $d_2 = d_1\alpha_1$ (items 1, 2, 3 and 4). Item 5 ensures to zip the vertices containing new darts in 2D (and edges in 3D). We can now define the conditions that a connected set of $(n-1)$-cells must verify in order to be used for sheet insertion.

Definition 3.4 (admissible hyperplane) *Let* $G = (D, \alpha_0, \ldots, \alpha_n)$ *be a structured* n*-G-map, with* $n = 2$ *or* 3 *and* $\mathcal{H}$ *a subset of* D*. Let* $G^g = (D^g, \alpha_0^g, \ldots, \alpha_n^g)$ *be the ghost layered* n*-G-map of* G*. Then* $\mathcal{H}$ *defines an admissible hyperplane in* G *if and only if:*

1. $d \in \mathcal{H} \Rightarrow <\alpha_0, \ldots, \alpha_{n-2}>(d) \subset \mathcal{H}$,

2. $\forall d \in \mathcal{H}, |<\widehat{\alpha_{n-2}^g}>d \cap \mathcal{H}| = 2$ *or* 4

3. $\forall d \in \mathcal{H}, |<\widehat{\alpha_{n-2}^g}>d \cap \mathcal{H}| = 2 \Rightarrow |<\widehat{\alpha_{n-1}^g}>d \cap \mathcal{H}| = 0$.

The first item ensures that if a dart of an half-edge in 2D, or an half-face in 3D, is in $\mathcal{H}$ then all the darts of the whole half-edge in 2D, respectively half-face in 3D, are in $\mathcal{H}$. By half, we mean here a "*side*", i.e an edge seen from a face in 2D and a face seen from a hex in 3D. Fig. 11(a) and 11(b) illustrate it in 2D: the darts of four "half"-edges are selected. The second item indicates that around a vertex in 2D, respectively an edge in 3D, the hyperplane can intersect at most once. If $|<\widehat{\alpha_{n-2}^g}>d \cap \mathcal{H}| = 2$, it does not intersect at this $(n-2)$-cell. If $|<\widehat{\alpha_{n-2}^g}>d \cap \mathcal{H}| = 4$, it intersects at this $(n-2)$-cell. It is the case for Figures 11.(a) and 11.(b) but not for Figures 11.(c) and (d). The last item is introduced to avoid the particular case where two half-cells would have been selected while belonging to the same cell (see Fig. 11.(d)).

Let us underline that by limiting $|<\widehat{\alpha_{n-2}^g}>d \cap \mathcal{H}|$ to 4 in condition 2 of Def. 3.4, we only consider intersection of four $(n-1)$-cells that will create a single quad in 2D or a hexahedron in 3D at the intersection. An admissible hyperplane $\mathcal{H}$ in an n-G-map H is defined using the ghost extension of H. Definitions 3.5 to 3.6 gives us core notions to formally define the sheet insertion process in Definition 3.8 and the corresponding algorithm in Algorithm 3. First of all, to perform sheet insertion, we extend the set of darts of $\mathcal{H}$ into the extended map.

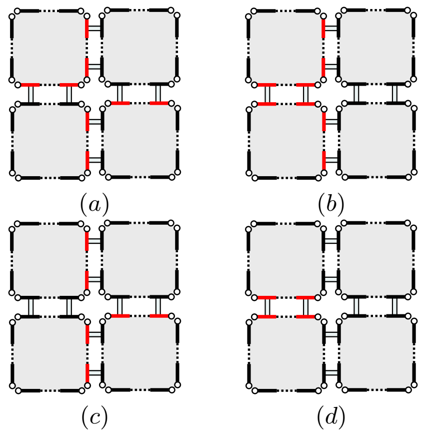


Figure 11: Examples of 2D admissible hyperplanes in (a) and (b) and non admissible ones in (c) and (d).

Definition 3.5 (ghosted admissible hyperplane) *Let* $G = (D, \alpha_0, \ldots, \alpha_n)$ *be a structured* n*-G-map, with* $n = 2$ *or* 3 *and* $\mathcal{H}$ *an admissible hyperplane in* H*. We extend* $\mathcal{H}$ *into the ghosted* n*-G-map* G^g *as follows: (1)* $H^g = \mathcal{H}$ *on* D*; (2)* $\forall d \in H^g$*, if there exists* $d' \in <\widehat{\alpha_0^g}>(d)$ *such that* $d' \in\subset \mathcal{H}$*, then* $d \in\subset \mathcal{H}^g$*.*

The second property of Def. 3.5 ensures that if a dart d belongs to the hyperplane, then all the darts of the vertex containing d are also in the hyperplane. It give us a homogeneous configuration for applying the same pattern to every dart of $\mathcal{H}^g$. We have one pattern in 2D and another one in 3D (see Fig. 12). The two following definitions formally introduce those patterns.

Definition 3.6 (2D Insertion Pattern) *We define* $P_2 = \{d_0, \ldots, d_5\}$ *the pattern made of 6 darts such that all those darts are 0, 1 and 2-free but* $d_1 = d_0\alpha_1$,$d_2 = d_0\alpha_{10}$, $d_3 = d_0\alpha_{12}$, $d_4 = d_0\alpha_{102}$ *and* $d_5 = d_0\alpha_{101}$ *(see Figure 12-a).*

Definition 3.7 (3D Insertion Pattern) *We define* $P_3 = \{d_0, \ldots, d_{11}\}$ *the pattern made of 12 darts such that all those darts are 0, 1, 2 and 3-free but* $d_{11} = d_0\alpha_{21012}$, $d_1 = d_0\alpha_2$, $d_2 = d_0\alpha_{21}$, $d_3 = d_0\alpha_{210}$, $d_4 = d_0\alpha_{2101}$, $d_5 = d_0\alpha_{2131}$, $d_6 = d_0\alpha_{213}$, $d_7 = d_0\alpha_{2103}$, $d_8 = d_0\alpha_{21013}$, $d_9 = d_0\alpha_{2132}$ *and* $d_{10} = d_0\alpha_{21032}$ *(see Figure 12-b and 17.a).*

We can now formally define the sheet insertion operation. In the following, we note $p(d)$ the set of darts

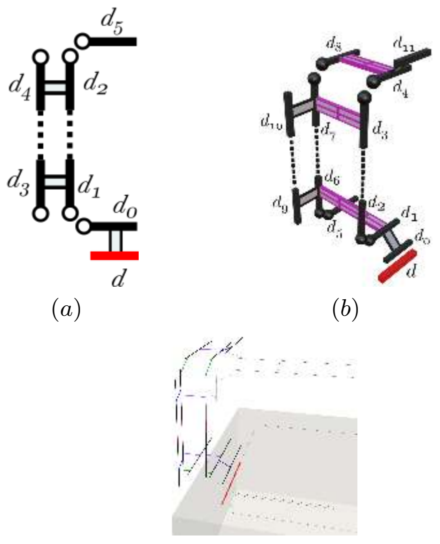


Figure 12: Insertion patterns in 2D (a) and 3D (b). For the red dart, we insert a configuration of 6 darts in 2D and 12 in 3D (with α_3 links). (c) the 3D pattern where the α_0 links to darts spawned by other marked darts are established based on $d\alpha_0 d$ (α_1 respectively).

that corresponds to the pattern P_n for dart d. The notation $p(d).f$ gives access to the first dart of P_n, $p(d).\ell$ gives access to the last dart of P_n and $p_i(d)$ gives access to the ith dart.

Definition 3.8 (Sheet insertion) *Let* $G = (D, \alpha_0, \ldots, \alpha_n)$ *be a structured n-G-map, with* $n = 2$ *or* 3, *and* $\mathcal{H}$ *and admissible hyperplane in* G. *The insertion of the sheet defined by* $\mathcal{H}$ *is the n-G-map* $G' = (D', \alpha'_0, \ldots, \alpha'_n)$ *such that:*

1. $D' = D + \{p(d)\}_{d \in \mathcal{H}}$,
2. $\forall i \in [\![0; n-1]\!],\ \forall d \in D,\ d\alpha'_i = d\alpha_i$,
3. $\forall d \in D'$,

$$d\alpha'_n = \begin{cases} d\alpha_n & \text{if } d \in D- <\widehat{\alpha_{n-1}}> (\mathcal{H}^g) \\ p(d).f & \text{if } d \in \mathcal{H}^g, \\ p(d').\ell & \text{if } d' \in \mathcal{H}^g \wedge d\alpha_n = d'. \end{cases}$$

4. $\forall d \in \{p(d)\}_{d \in \mathcal{H}^g}, \exists k_1 > 0 / d(\alpha'_{n-1}\alpha'_n)^{k_1} = d$ *and* $\exists k_2 > 0 / d(\alpha'_{n-2}\alpha'_{n-1})^{k_2} = d$.

In order to try and explain this definition, let us consider the 2D case shown on Figure 13. Starting from the hyperplane $\mathcal{H}$ (red darts) given as an input in Fig. 13.a, we show the ghost layer extension in (b) and the ghosted hyperplane $\mathcal{H}^g$. Then for each dart of $\mathcal{H}^g$, we show the inserted 2D pattern (see Def. 3.6) in Fig. 13.c. We have here all the darts of D' (item 1 of Def. 3.8). Item 2 indicates that we preserve all the α_i link for the darts of G, for all $i \in [\![0; n-1]\!]$.

The third item indicate that we also preserve α_2 links for the darts that are not involved in the sheet insertion process (those that are not connected to a dart of $\mathcal{H}^g$). We also reconnect the inserted patterns via $p(d).\ell$ and $p(d).f$ (see Fig. 13.d). As G' is a 2-G-map, some α_0 link are implicitly performed to ensure that α_{02} is an involution (see Fig. 13.e).

The fourth item close some open cells. In 2D the darts $p_3(d)$ and $p_4(d)$ are 1-linked with the first dart 1-free m of their orbits $<\alpha_1, \alpha_2>(d), d \neq m$ (see Fig. 13.d). (around vertex in 2D and around edge in 3D). It gives us the result of Fig. 13.e. It remains then to remove[4] some *flat* or *compressed* $(n-1)$-cells to get from Fig. 13.e to Fig. 13.f. In fact, Definition 3.8 allows us to define the result of Fig. 13.f without the ghost darts.

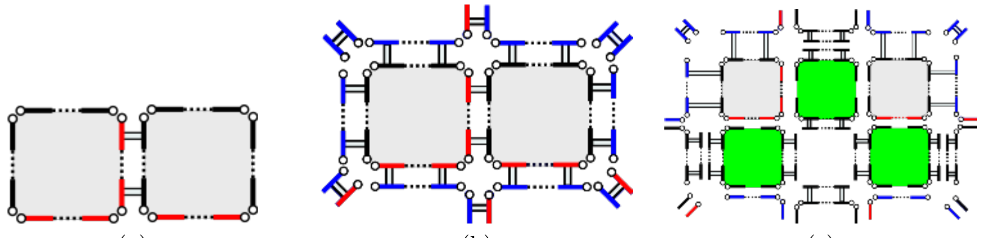
Algorithm 3: n-Insertion

```
Data: A n-G-map $H = (D, \alpha_0, \ldots, \alpha_n)$, a set of
      selected darts $D_e$
1 Add a ghost-layer $g$ on the boundary of $H$;
2 $D_e \leftarrow D_e +$ darts selected in $g$ (see Def. 3.5) ;
  /* Fig. 13.(b)                                  */
3 Memorise $\alpha_n^p, \forall d \in D_e$ ;
4 $n - unsew$ all darts $\forall d \in D_e$ ;
5 Generate local nD-pattern $\forall d \in D_e$;
  /* Fig. 3.6 Fig. 3.7, Fig. 13.(c)               */
6 Link patterns ;
  /* Fig. 13.(d)-(e)                              */
7 Collapse compressed n-cells ;
8 Remove $g$ ;
  /* Fig. 13.(f)                                  */
```

From Def. 3.8, we derive the **Algorithm 3** that defines the sheet insertion both 2 and 3D. Only the link stage differs between 2D and 3D (line 6). We keep using Fig. 13 to explain the algorithm. Given a selection set D_e, that is an admissible hyperplane (see Def. 3.4), we first insert a ghost layer (line 1), as defined in Def. 3.3, to get from Fig. 13.a to Fig. 13.b. To ease the final suppression of the ghost layer, we mark the dart of a pattern generated for the ghost layer as to be removed later. The usage of the ghost layers and the ghosted hyperplane (line 2) allows us to write a generic algorithm without having to consider specific cases for boundary darts.

Note that unlike Def. 3.8, we incrementally update

[4]We do not formally define this operation, which is quite general, for a lack of space.


(a) (b) (c)

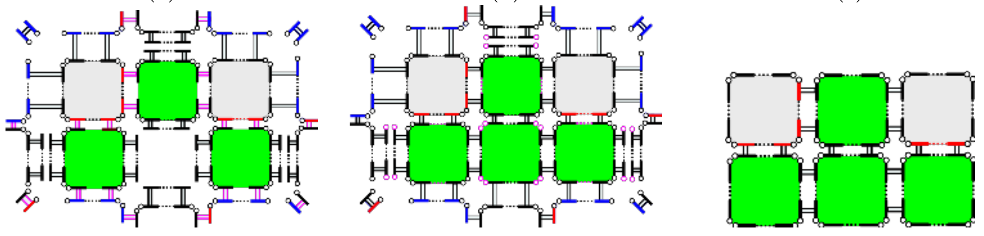

(d) (e) (f)

Figure 13: Pipeline of the 2-Insertion for selected darts in red (see Algorithm 3). Links added are in pink (a) Selection in red (b) Add the ghost-layer in blue, extend the selection in red with darts of ghost layer (c) Memorize α_n^p, $2 - unsew$ darts selected (d) 2-link ▨ $p(d).\ell$ and $p(d).f$ (e) 0-link ⁘ $p(d).f$,$p(d).\ell$ and 1-link ○ p_3, p_4 (f) Result with collapsed 2-compressed faces and removed ghost layer.

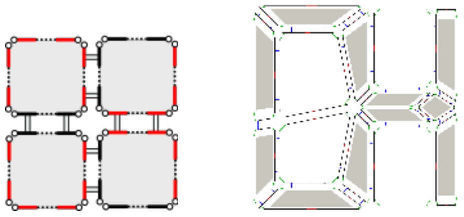
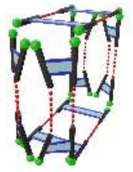

(a) (b)

Figure 14: (a) Marked darts; (b) after an insertion operation and collapse of the central inserted face (see Fig. 15).

the n-G-map in the algorithm[5]. So we store the initial function α_n, noted α_n^p,to be used later to insert the patterns (line 3). After that, we unsew all the darts of D_e for α_n. Then the pattern is inserted for every dart $d \in D_e$ but not n-sew to the initial n-G-map already (see Fig. 13.c). Line 6 of Algo. 3 differs in 2D and 3D. It corresponds to the fourth item of Def. 3.8. We give some details about their implementation afterward.

At line 7, we collapse compressed n-cells. The 2-insertion pattern, given in Def. 3.6 generates compressed 2-cells (see Fig. 16). We get the same kind of compressed 3-cells in 3D (see on the right). In both dimension, we detect such cells as they own at least one dart d such that $d\alpha_{0101} =$ d. Once detected, we remove every compressed n cell. In 2D, a compressed 2-cell C_2. In 3D, a compressed 3-cell C_3 are removed and we 3-sew $\alpha_{23}(d)$ with $\alpha_{123}(d)$ for d a dart of C_3. Finally the ghost layer is removed (line 8).



[5]A side effect is that n-G-map properties are not necessarily verified during the algorithm, but just at the end.

Links in 3D, we first 0-link darts $\{0, 1, 4, 5, 8, 11\}$ of a selected dart d with darts of $p(d\alpha_0)$. For example dart $p_0(d)$ is 0-linked with $p_0(d\alpha_0)$. After that, darts $p(d).f$, $p(d).\ell$ and $p_{2,3}(d)$ are respectively 1- and 2-linked with darts of $p(d\alpha_1)$ (see Fig. 12.c).

We now link darts $p_{5,9}(d)$ to the pattern spawned by the marked dart d' found in the orbit $<\alpha_2, \alpha_3>$ $(d), d' \neq d$. Dart $p_5(d)$ is 2-linked to $p_5(d')$ while $p_9(d)$ is 1-linked to $p_9(d')$; if there is no such d' nothing is done. We then look for the marked dart m ($m \neq d$) in orbit $<\alpha_2, \alpha_3>$ $(d\alpha_3^p)$. If m is found we 1-link $p_{10}(d)$ with $p_9(m)$ and 2-link $p_8(d)$ with $p_5(m)$. We proceed similarly with d'. If there is no such dart m, we are in the usual case illustrated in Figure 17.e where we form a flat 3-cell between $p(d)$ and $p(d')$. Figure 17.f shows the auto-intersecting case where the patterns open-up to form an hexahedron (see Fig. 17.a and b).

We then close the orbit $<\alpha_0, \alpha_1>$ $p_{10}(d)$ by 1-linking together the two α_1-free darts f and f', and we 2-link $f\alpha_{21}$ and $f'\alpha_{21}$ which closes the flat 3-cells and the hexahedra. After the link done in n-D we have to remove the compressed n-cell; in 3D, we remove the compressed 3-cells and 3-sew into a chord the hexahedra created on the auto-intersection. For a 3-free dart d of such an hexahedron, we 3-sew it with the other 3-free dart of $<\alpha_2, \alpha_3>$ (d). To address the case of a

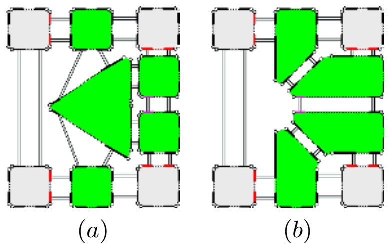


Figure 15: Collapse a 2-cell to generate a self-touching pattern.

self-touching self-intersection (see Fig. 14.a) in 2D, we define an operation which collapse a 2-cell when two new 2-cells inserted are 2-linked Fig. 15.b.

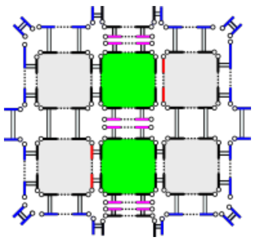


Figure 16: Simple insertion of two 2-cell (green). Darts in pink form three compressed 2-cells.

4. CONCLUSION AND FUTURE WORKS

In this work, we formally defined and implemented hexahedral blocking operations using the n-G-map models. Using this model brings us many benefits: (1) the ability to get a unique definition in 2D and 3D for our operations; (2) the clear separation of concerns between topology and geometry; (3) formal pre- and post-conditions to validate the block structure during blocking operations; (4) the usage of orbits, which are much more general than regular cells.

The first three items were very important to be able to get a clean implementation of the sheet operations, especially the sheet insertion. We are able to insert self-intersecting and self-touching sheets in a robust manner. The merging rules, introduced to handle geometrical classification and vertex location, coupled with the sewing and unsewing operations helped us guarantee the robustness of the operations.

In order to go further, we expect to allow more complex sheet insertion patterns. We also plan to formally prove the robustness of our approach by deriving from the definitions we proposed in Section 3 a system of rules using, for instance, the Jerboa framework [28]) to ensure our definitions and algorithms are correct.

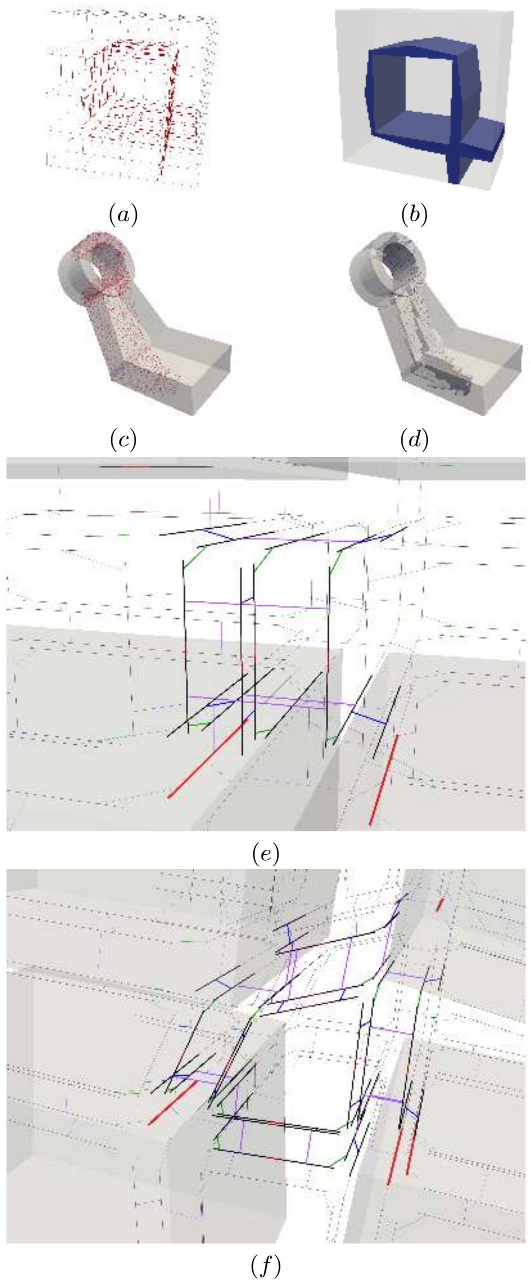


Figure 17: Marked darts of a grid mesh along which an auto-intersected sheet will be inserted (a) and the resulting mesh with the highlighted sheet (b). In (c) a more complex case with (d) a cross-section to show the inserted sheet. The two highlighted 12-darts patterns spawned by the two red darts (see Fig. 12) form a flat 3-cell in (e) (the same is illustrated in 2D in Figure 16). In (f) where the sheet auto-intersects the four patterns form an additional hexahedron 3-cell.

References

[1] Pietroni N., Campen M., Sheffer A., Cherchi G., Bommes D., Gao X., Scateni R., Ledoux F., Remacle J.F., Livesu M. "Hex-Mesh Generation and Processing: A Survey." *ACM Trans. Graph.*, jul 2022. URL `https://doi.org/10.1145/3554920`. Just Accepted

[2] Cubit. "Sandia National Laboratories: CUBIT Geometry and Mesh Generation Toolkit." URL `https://cubit.sandia.gov/`

[3] Smith M. *ABAQUS/Standard User's Manual, Version 6.9.* Dassault Systèmes Simulia Corp, United States, 2009

[4] ANSYS. "ANSYS Fluent - CFD Software — ANSYS.", 2016. URL `http://www.ansys.com/`

[5] Mäntylä M. *An Introduction to Solid Modeling.* Computer Science Press, Inc., USA, 1987

[6] Lienhardt P. "N-Dimensional Generalized Combinatorial Maps and Cellular Quasi-Manifolds." *Int. J. Comput. Geom. Appl.*, vol. 4, 275–324, 1994

[7] Remacle J.F., Shephard M. "An Algorithm Oriented Mesh Database." *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, 2003

[8] Seol E.S. *FMDB: Flexible Distributed Mesh Database for Parallel Automated Adaptive Analysis.* Ph.D. thesis, Rensselaer Polytechnic Institute, 2005

[9] Garimella R.V. "Mesh data structure selection for mesh generation and FEA applications." *International Journal for Numerical Methods in Engineering*, vol. 55, pp. 451–478. 2002

[10] Garimella R.V. *MSTK: MeSh ToolKit, v 1.3 User's manual.* Los Alamos National Laboratory, 2012. LA-UR-04-0878

[11] Tautges T., Ernst C., Merkley K., Meyers R., Stimpson C. "Mesh Oriented datABase (MOAB).", 2005. Http://cubit.sandia.gov/cubit

[12] Ledoux F., Bertand Y., Weill J.C. *Generic Mesh Data Structure in HPC Context*, vol. 26 of *Computation Technologies and Innovation Series*, chap. 3, pp. 49–80. Saxe-Coburg Publications, Stirlingshire, 2010

[13] Edelsbrunner H. *Algorithms in Combinatorial Geometry.* Springer-Verlag, New-York, 1987

[14] Baumgart B.G. "A Polyhedron Representation for Computer Vision." *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, p. 589–596. Association for Computing Machinery, New York, NY, USA, 1975. URL `https://doi.org/10.1145/1499949.1500071`

[15] Muller D.E., Preparata F.P. "Finding the Intersection of two Convex Polyhedra." *Theor. Comput. Sci.*, vol. 7, 217–236, 1978

[16] Weiler K. "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments." *IEEE Computer Graphics and Applications*, vol. 5, no. 1, 21–40, 1985

[17] Sieger D., Botsch M. "Design, Implementation, and Evaluation of the Surface_mesh Data Structure." *IMR.* 2011

[18] Damiand G., Lienhardt P. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing.* A K Peters/CRC Press, September 2014

[19] Rossignac J. "3D compression made simple: Edgebreaker with ZipandWrap on a corner-table." *Proceedings International Conference on Shape Modeling and Applications*, pp. 278–283. 2001

[20] Tautges T.J. "Local Topological Modifications of Hexahedral Meshes using Dual-Based Operations." *8th U.S. National Conference on Computational Mathematics.* July 2005

[21] Shepherd J.F., Johnson C.R. "Hexahedral mesh generation constraints." *Engineering with Computers*, vol. 24, no. 3, 195–213, 2008

[22] Ledoux F., Shepherd J.F. "Topological and geometrical properties of hexahedral meshes." *Engineering with Computers*, vol. 26, no. 4, 419–432, 2010

[23] Ledoux F., Shepherd J.F. "Topological modifications of hexahedral meshes via sheet operations: a theoretical study." *Engineering with Computers*, vol. 26, no. 4, 433–447, 2010

[24] Brisson E. "Representing Geometric Structures in D Dimensions: Topology and Order." *Symposium on Computational Geometry*, pp. 218–227. 1989

[25] Lienhardt P. "Subdivisions of n-dimensional spaces and n-dimensional generalized maps." *Annual ACM Symposium on Computational Geometry*, pp. 228–236. 1989

[26] Lienhardt P. "Topological models for boundary representation: a comparison with n-dimensional generalized maps." *Computer Aided Design*, vol. 23, no. 1, 59–82, 1991

[27] Staten M.L., Shepherd J.F., Ledoux F., Shimada K. "Hexahedral Mesh Matching: Converting non-conforming hexahedral-to-hexahedral interfaces into conforming interfaces." *International Journal for Numerical Methods in Engineering*, vol. 82, no. 12, 1475–1509, 2010

[28] Belhaouari H., Arnould A., Le Gall P., Bellet T. "JERBOA: A Graph Transformation Library for Topology-Based Geometric Modeling." H. Giese, B. König, editors, *7th International Conference on Graph Transformation (ICGT 2014)*, vol. 8571. Springer, York, United Kingdom, Jul. 2014. URL `https://hal.archives-ouvertes.fr/hal-01012851`