Boolean operations on arbitrary polygonal and polyhedral meshes

Sâm Landier

Boolean operations on arbitrary polygonal and polyhedral meshes.

Sâm Landier[a,*]

[a]ONERA, 29 avenue de la Division Leclerc, 92320 Châtillon, France

Abstract

An $O(nlogn)$ floating-point arithmetic algorithm designed for solving usual boolean operations (intersection, union, and difference) on arbitrary polygonal and polyhedral meshes is described in this paper.
This methods does not approximate the inputs which can be both volume meshes, both surface meshes or one of each. It provides exact and conformal meshes upon exit. It can be used in many pre- and post-processing applications in computational physics (e.g. cut-cell volume mesh generation or conservative remapping).
The core idea is to consider any configuration as a polygonal cloud. The polygons are first triangulated, the intersections are solved, the polyhedral cells are then reconstructed from the conformal triangles cloud and finally their triangular faces are re-aggregated to polygons. This approach offers a great flexibility regarding the admissible topologies : non-planar faces, concave faces or cells and some non-manifoldness are handled. The algorithm is described in details and some current results are shown.

Keywords: Boolean Operations, Polyhedral Meshes, Polygonal Meshes, Mesh Intersection, Cell Reconstruction, Conformity, Conservative Remapping, Cut-cell Meshing, Constrained Delaunay Triangulation, Flood Fill Algorithm.

∗. Corresponding author
URL: `sam.landier@onera.fr` (Sâm Landier)

1. Introduction

Considering two arbitrary polyhedral meshes M_1 and M_2 that are partially or fully overlapping (e.g. one is fully immersed in the other one, or both are representing the same computational domain) and a given tolerance, a floating-point arithmetic method is proposed to solve the geometric intersections and retrieve a conformal polyhedral mesh upon exit that results from the usual boolean operations (intersection, union and difference). A typical target application is related to conservative remapping of some solution fields from one mesh to another [1; 2; 3; 5; 12; 13; 14; 19]. In all these works it is assumed that the source and the target are two meshes of the same domain. But if we want to do some conservative remapping in the Chimera context [15], it is also required to handle partial overlapping, so a general algorithm should not do such assumption and treat the fully-immersion or same-boundary cases as particular cases of a partial overlapping as depicted in Fig. 1.

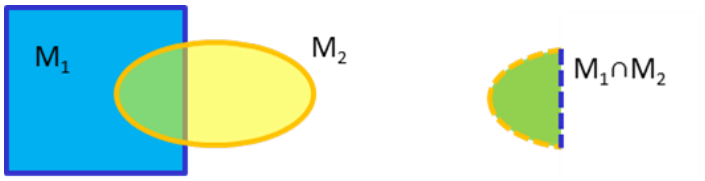

FIGURE 1: Two overlapping meshes, their intersection zone in green.

The conservative remapping problem induces necessarily a high computation cost because all the intersections between the polygons must be resolved in the intersecting zone (in green Fig. 1).

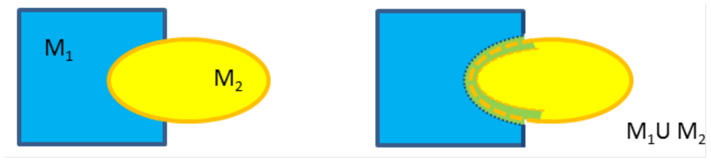

FIGURE 2: Two overlapping meshes the yellow mesh is prioritized, their intersection zone in green.

Applications like conservative interfacing [6; 7] (cf. Fig. 2) for overlapping grids require to build a conformal mesh of a domain where each subdomain

has been meshed separately. The conformal assembly is achieved prioritizing some meshes that come fit into non-priority meshes. The intersection zone is therefore reduced to a narrower region in the vicinity of the surface of the prioritized meshes.
This is similar to cut-cell meshing (cf. Fig. 3) where the impact on the cut mesh takes place in the vicinity of the cutting surface.

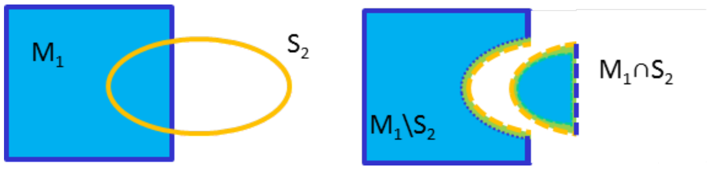

FIGURE 3: Cut-cell meshing : a surface mesh cuts a volume mesh (intersection zone in green).

Some other applications like Chimera blanking [15] or Constructive Solid Geometry evaluation [8; 9; 21; 22] works on a reduced sets of large polyhedra (polygonal surface meshes) and their intersecting zone is inherently narrower (cf. fig 4) compared to the same problem with volume meshes (even though [21] solves very efficiently dense problems with 100,000 polyhedra having millions of intersecting polygons).

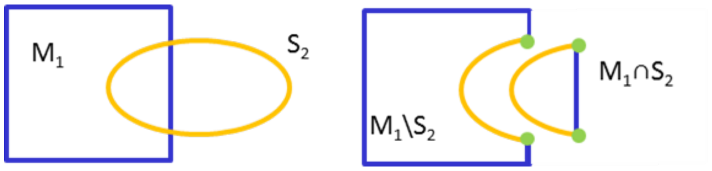

FIGURE 4: Pure surface problem as in a CSG evaluation (intersecting zone in green).

All these different problems, as for CSG evaluation, can be expressed in terms of boolean operations between two meshes :

- The conservative remapping problem is an intersection operation on meshes without prioritization.
- The interfacing problem is a union with prioritization.
- The cut-cell meshing is a mixed-type difference between a volume mesh and a surface mesh.

The proposed algorithm covers all these problems by defining boolean operations on two arbitrary meshes operands :

- polyhedral vs polyhedral (with or without prioritization)
- polyhedral vs polygonal
- polygonal vs polygonal

When a mesh is prioritized, there are two steps in the process : first its outer skin is processed considering the boolean operation between a polyhedral mesh and a polygonal mesh. Then this modified skin is reattached to its interior to get a conformal result (as explained in section 5.6).
As a closed surface mesh is a polyhedron, boolean calculation for pure polygonal inputs is seamlessly handled.

2. Previous 3D works

Among the most recent 3D algorithms, Alauzet [19] focuses on pure tetrahedral mesh intersections. The topological approach fits very well pure tetrahedrons problems. For each pair of intersecting tetrahedrons, the 32 signed distances of each vertices of one tetrahedron versus the four facets of the other one allow to build some topological information that reduces the intersection tests, prevents inconsistencies and ensures robustness. Even though this approach is pragmatic for simplices, pre-computing all the signed distances might be cumbersome for larger polyhedra and not practicable at all for surface mesh inputs since the number of signed distances to compute grows exponentially (for a small polyhedra that is a triangulated hexahedron, we should compute 192 signed distances).
Moreover, splitting the intersection zone into tetrahedrons leads potentially to engender a lot of cells and therefore a higher memory usage compared to a polyhedral approach if we want ouput the resulting conformal mesh. An algorithmic advantage however is the fact that the tetrahedrons are convex which greatly facilitates the recovery of intersecting areas (always convex too) and computing the conservative field. In a polyhedral approach, the necessary handling of concavities increases the complexity of the algorithm.
Finally, since the geometric kernel (intersection computation) and the conservative remapping are strongly linked as being part of the same iterative loop for each pair of intersecting tetrahedrons, there is no need to keep a geometric representation of the intersections : once two tetrahedrons are processed, every data related to that pair is flushed. This is a good thing for coarse

grain multi-threading but induces redundant intersection calculations.
The algorithm described in this paper focuses on the geometric construction so it aims at building an explicit polyhedral mesh representation available upon exit. As it is designed, each polygon is processed once. This choice will not prevent to multithread the current sequential implementation though.

Another 3D algorithm designed by Brenner [6; 7] deals with polyhedral (with triangular faces only) mesh intersections over overlapping grids to solve the conservative interfacing problem. It does not give an explicit representation of the resulting mesh : surface vectors and centroids (faces and elements) are instead provided. This is fair enough for the targeted conservative remapping applications but it might be difficult to get to order higher than two or three without the explicit representation. Moreover only few publications of this algorithm are available so it is difficult to analyze it further.

Farrell et al. [1; 2] have proposed an algorithm to solve the conservative remapping problem and their geometric kernel (which is separated from the conservative calculations) has some similarities with the one discussed in this paper, especially their 2D version [1] : they indeed solve first the edge intersections, and the whole set of split edges is then processed by a constrained-Delaunay triangulator to provide what is called a supermesh. The benefit of this global approach is to avoid intersections calculation redundancy. It might be not necessary though to keep the entire supermesh into memory when the algorithm is purely dedicated to conservative remapping. The 3D version adopts a local approach as in [19] (inducing redundancy) but deals only with convex elements because the clipping Eberly′s algorithm [4] is used.

Following efforts have been made by Menon and Schmidt [3] to extend the supermesh concept to 3D but the proposed algorithm is restricted to star-shaped polyhedra because of their implementation of the algorithm requirement to break down the polyhedra into tetrahedrons. A similar requirement is necessary for the algorithm proposed by Grandy [5].

The algorithm described in this paper can deal with almost any type of polyhedral topologies (i.e. useful for practical applications as described in the next section) by solving the intersection between faces avoiding the requirement of such volume decomposition.

3. Admissible topologies

Here are the characteristics of the geometric entities handled by the algorithm.

3.1. Admissible polygon (aPG)

An aPG is a 3D polygon with p vertices forming q edges satisfying the following conditions :

- Its edges are not intersecting each other : it′s a conformal set (c1)
- Its boundary (its edges excluding the non-manifold bits) form a closed contour confining a non-null surface (c2)
- Its boundary is made of a single connected polygonal line : there are no holes nor separate subdomains (c3)
- A plane exists such the projected polygon has the same topology (c4)
- A triangulation of its boundary exists such all the p vertices lie on the triangulated surface (c5)

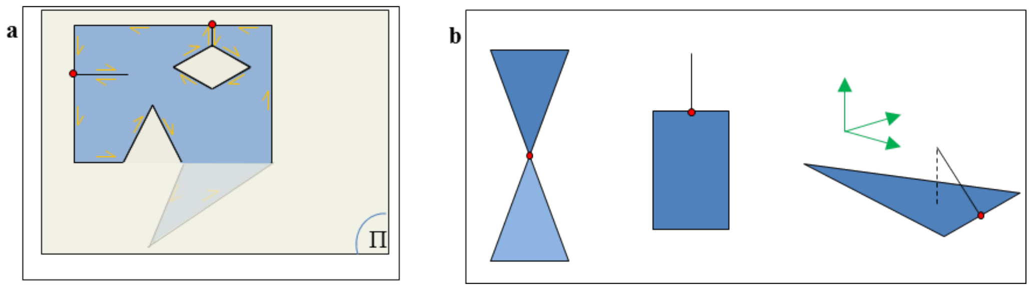

FIGURE 5: (a) aPG ; (b) three non-admissible polygons.

Conditions (c1) (c2) (c4) are necessary and sufficient conditions to be able to triangulate the polygon as it is required for the algorithm as explained later. Condition (c3) is not strictly necessary but added for convenience and to ease the implementation : with this condition it is always possible to represent the polygon as a sorted vertices list of size q such any two consecutive vertices on this list forms a polygon edge (the first and last too). Condition (c5) is to ensure consistency between the polygon and its triangulated representation. Oddities such as in Fig. 5b are not admissible.

3.2. Admissible polyhedron (aPH)

An aPH is a polyhedron with f faces satisfying the following conditions :

- Its faces form a conformal set of aPGs (c6)
- Its boundary (its faces excluding the non-manifold bits) forms a closed contour confining a non-null volume (c7)
- Its boundary is made of a single connected polygonal surface : there are no holes (c8)
- Any non-manifold entity must be an aPG and has to share an edge with the boundary (c9)

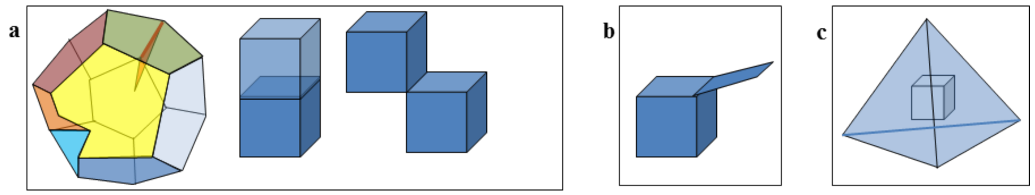

FIGURE 6: (a) and (b) some aPHs ; (c) non-admissible case : a tetrahedron with cubic hole.

Condition (c6) is necessary to build a one-to-one polygons neighborhood information and condition (c8) (c9) ensure that one is able to use that information to traverse totally the polyhedron through its faces. So it implies that if a face has a non-manifold edge, it must be the edge of another face.

We define a minimal aPH (maPH) as an aPH satisfying the following extra condition :

- There are no separate subdomains : any two inside points can be linked by a strictly interior polygonal line that does not intersect the faces (c10)

This condition (c10) is to define the polyhedra upon exit due to the element building process explained later on. The cubes sharing an edge or a face on Fig. 6a are four separate maPHs.

3.3. Admissible mesh (aMH) and minimal admissible mesh (maMH)

An aMH (maMH) is a mesh satisfying the following conditions :

- Its elements form a conformal set of aPHs (maPH) (c11)

- The intersection volume between two of its elements is null (i.e. no penetration) (c12)
- The outer skin of the mesh (polygonal surface) is manifold in the outer space (c13)

Condition (c13) means that for any polyhedron of the outer layer, its non-manifoldness must be inside (like the Triskaidecahedron′s red face on figure 6a) because any outside one (quadrangular face on the cube Fig. 6b) will be lost in the element building process. The algorithm handles as operands any two aMH upon entry and creates a maMH upon exit. As a closed surface mesh is a polyhedron, boolean calculation on surface meshes or mixed types (volume vs. surface) admissible operands are a particular case seamlessly handled.

4. Boolean operations

Considering two sets of arbitrary elements M_1 and M_2, let′s recall first what is meant by boolean operation in algebra of sets :

- $M_1 \cap M_2$: common elements to both M_1 and M_2
- $M_1 \setminus M_2$: elements inside M_1 and outside M_2
- $M_1 \cup M_2$: elements inside M_1 or M_2

These definitions assume that an element can always be classified, i.e. one can tell without any ambiguity this element is either inside or outside a set. Boolean operations make only sense if a predicate "in or out" can be defined. But in our geometrical context dealing with Boundary-Representation volume entities in arbitrary positions, we have to cope with ambiguity, i.e. an entity will often be partially inside. Hence we need to transform the input operands, and in fact the concatenation of them $M_1{+}M_2$, to make classification possible before being able to answer to any boolean operation.
The first step of the transform is to make conformal all the polygons, i.e. solving all the intersections. The second step is to create from that conformal cloud all the possible maPHs. The resulting set is a maMH created from the input operands. Hence for any maPH created it is always possible to find an element of M_1 and/or M_2 that geometrically includes it. A predicate definition P is then possible.
Let′s note ADM the operator that converts $M_1{+}M_2$ into a maMH. We will see that we won′t define our predicate that way but for now, its existence is enough to formalize in our context the boolean operations :

- $M_1 \cap M_2$: elements e of ADM(M_1+M_2) / P(e, M_1) and P(e, M_2) are both true.
- $M_1 \setminus M_2$: elements e of ADM(M_1+M_2) / P(e, M_2) is false
- $M_1 \cup M_2$: ADM(M_1+M_2)

The main difference with set algebra is hence the requirement of applying the ADM operator to the concatenated mesh. Another difference with this formalism is that upon exit we only have a set of volume entities. So for instance if we compute the intersection of two polyhedra sharing a polygon the result will be empty rather than giving the shared polygon. But this is fine with the current target applications.
A common point between the two worlds, the union operation doesn′t need a classification step : all considered elements are in the result.
Although we will define the ADM operator that turns some meshes into minimal admissible ones, it cannot handle polygons with holes or polyhedra with non-unique contour which won′t appear if they are not in the input, so admissibility is a prerequisite to discard those topologies.

5. The algorithm

5.1. Overview

The concatenated mesh M_1+M_2 as global and unique input needs to be transformed into a maMH. Defining the ADM operator is the main part of the algorithm. Defining the predicate (based on normals orientations) and the consequent classification is more straightforward. The ADM operator breaks down into four steps.
The first step is an initialization that focuses on the intersecting zone by discarding any non-colliding polyhedron. An additional but optional treatment can be applied to ensure that the reduced set is oriented consistently meaning that each polyhedra′s facet has a normal pointing outward. This treatment is not applied if the input data set is known to be oriented properly.
The second step is to make conformal all the polygons seen as a cloud in arbitrary position. Solving intersections for non-planar polygons leads to non-unique solutions because of the multiple definitions of their surface support : we′ve therefore made the usual choice to deal with intersections processing at the triangle level and build a conformal triangles cloud first.
The third step is to reconstruct the polyhedral elements from the triangles

cloud. This is the key step that builds maPHs with triangular faces by detecting any smallest enclosed cavity.
The last step is to aggregate the triangular faces to coming from the same initial polygon as long as the aggregate is minimal admissible. These obtained polyhedra are therefore maPHs. The whole process is depicted in the Fig. 7.

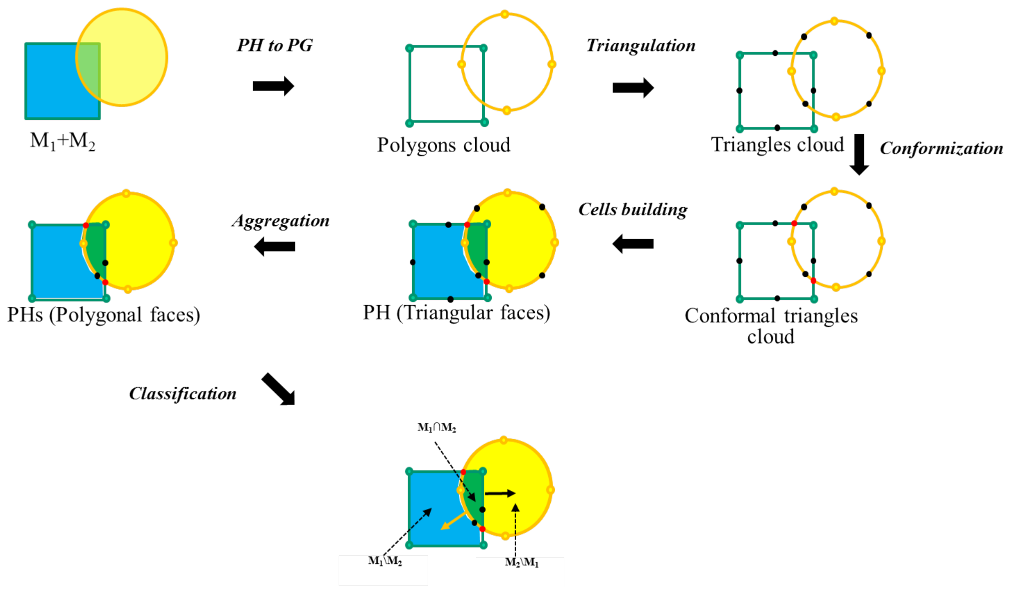

Figure 7: Algorithm's scheme.

5.2. *Initializing and Pre-conditioning*

The two main purposes of this step are to focus on the intersecting zone (keeping apart the elements which are not involved in the intersection) and to prepare the classification step.
The latter requires to know which are the outer polygons (all of them in case of a surface mesh, the ones that are not shared in case of a volume mesh) and to have them consistently outward oriented.

A primary coarse decimation is done using the bounding box of the input meshes : any element that is not intersecting or inside both boxes is discarded.

A finer refinement is then applied to the remaining elements using a bounding box tree constructed for the smallest mesh. It is indeed more efficient to construct a small tree and query it intensively rather than the opposite. Any element that is colliding with at least one other element is kept.
At the end we get a set of intersecting zones ; one of them is depicted on Fig. 8b.
Any outer polygon of these sets that is not already flagged as "pure skin" is flagged as "connection skin" (meaning connected to a discarded set).
The discarded elements can already be classified as outside elements and will be agglomerated after the classification step to generate the boolean operation ouput.

The final step of the initialization is to merge the coincident nodes in the set of retained points using a kd-tree. This step facilitates the intersection process by removing one type of degeneracies (cf. 5.4.1). But when two nodes need to be merged, which one has to be moved ? We have chosen to preserve one mesh with respect to the other one : its nodes are attractors in case of merging. This is what is required for instance for a cut-cell mesh : the cutting surface has to be preserved. For a conservative remapping, the target mesh needs as well to be preserved with respect to the source mesh.

Optional reorientation step. Each individual set of skin polygons (pure or connection) forming a closed surface is reoriented consistently (outward) in regard with a reference polygon (one on each set) with a neighborhood traversal algorithm : starting by the reference polygon, we reverse any neighbor with the opposite orientation and so on. For a closed surface it is always possible to find a polygon having a ray perpendicular to it and passing by an arbitrary point on it that does not intersect the surface either above or under the polygon : this indicates the outward so the polygon is flagged and used as a reference after reorienting it properly. This reorientation is crucial for the classification step.

The algorithmic complexity of this initialization step is in O(nlog(n)) (where n is the number of kept cells) because of the bounding box tree constructions and queries. An alternative to bounding box trees through the neighboring's information use is discussed in [2; 12] and might be more efficient for preconditioning in the particular case of a boolean intersection of two meshes of the same domain (which is the context of their conservative remapping).

They will be considered in future work.

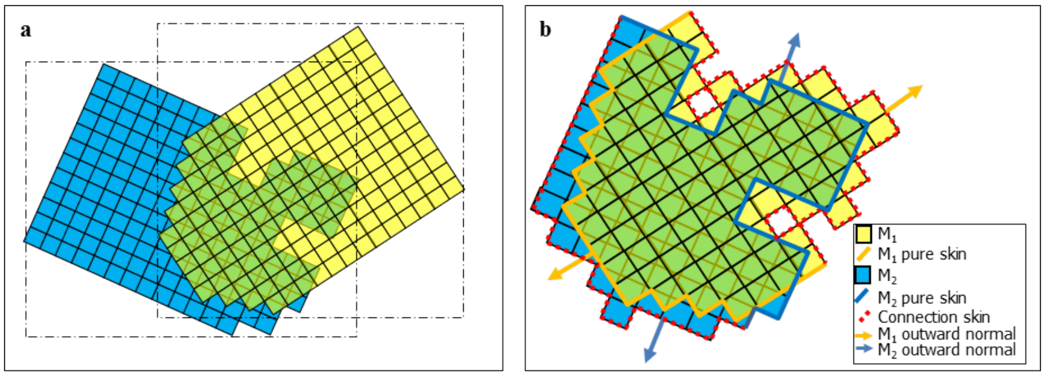

FIGURE 8: (a) initial aMHs M_1 and M_2 ; (b) working set for intersection after initializing a pre-conditioning for a pure polyhedral problem.

5.3. From the polygons to a conformal triangles cloud

Each polygon in the cloud is triangulated. Many strategies are available but the choice to use a 2D-Constrained Delaunay approach [11] has been made for its robustness dealing with any kind of polygons topology providing they fall into the admissible polygons family. Basic polygons (e.g. convex quads) are obviously processed trivially. Cheaper and therefore faster solutions will be applied in the future to eligible polygons in the cloud : a star-shaped one can be triangulated by linking the centroid to the boundary edges but this tends to produce more triangles and therefore more intersections computations. Other approaches like the Ear clipping algorithm [23; 24] could generate less triangles with an acceptable efficiency. This preliminary step is not a current bottleneck anyway. Once we have a triangle cloud and built the mapping table ancPG giving for each triangle the polygon it belongs to, we can go to the intersections solving process. When this process is done, ancPG is updated accordingly to refer to the new set of triangles that will be used to aggregate back the triangles to make aPGs from them as described in 5.7.

5.4. Intersection solving process

If we consider a cloud of intersecting triangles in arbitrary positions (cf. Fig. 9a), the intersection traces in each triangle are of two types : a point or a segment. In case of overlapping, the trace is a polygon so a set of segments too (cf. Fig. 9b).

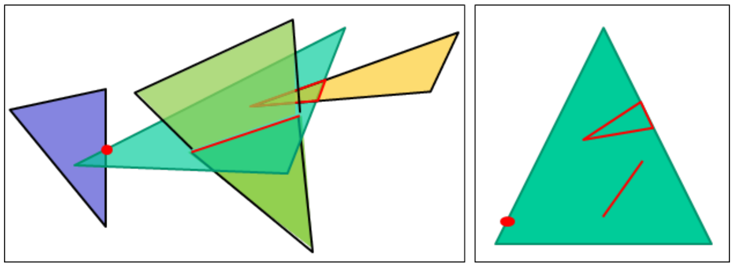

FIGURE 9: (a) Triangles cloud ; (b) Intersections traces.

Rather than detecting and resolving on the go one intersection at a time and putting the split triangles back to the pool of triangles to be processed, we′ve chosen a two-step algorithm :

- Computing and storing the traces for each triangle
- Splitting each intersected triangle along the intersection traces

The drawback of this approach is the storage of the traces but the benefit is to avoid an iterative algorithm that would imply redundant queries to the localizing structure (e.g. a bounding box tree).
Here, each triangle has a data structure that stores the segments reflecting the intersections traces and the isolated points (degenerated segments) for each pair of intersecting triangles. Each edge has also a list of intersection points that is appended during the process.
Once all the traces have been computed, the split can be done (cf Fig. 10a).

Storing the edges (rather than the intersection points only) are necessary because we aim to deal with concave polyhedra, either as input, or as resulting from e.g. a boolean difference between to input meshes having a convex outer surface. It is not therefore possible to assume (as it is done in [19] that the intersection traces and their resulting polygons are convex. This is only true for a boolean intersection over convex polyhedra.

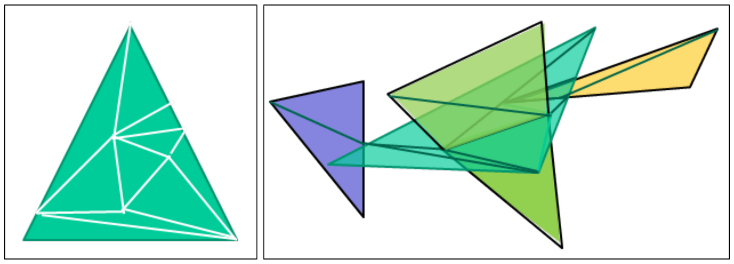

Figure 10: (a) Split triangle ; (b) Resulting conformal cloud.

5.4.1. Tolerance and degeneracies handling

An epsilon-approach is used to cope with numerical errors and an interference zone (noted I_z) surrounding any triangle is considered as in [10].

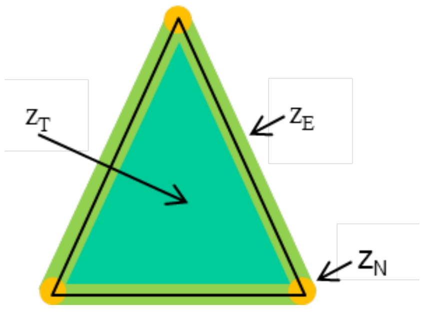

Figure 11: Interference zone for a triangle : the location of points at a distance less than the tolerance. z_N is a ball around each vertices, z_E is a partial cylinder around each edges.

In order to get a robust answer and to minimize the point creation, it is required to handle degeneracies carefully and to prioritize the tests by detecting first if any intersection point is in z_N, then in z_E and finally in z_T. This is approach is opposed to [20] where degeneracies are rejected : for an intersection problem, degeneracies contain the sufficient intersections information (cf. Fig. 12).
The degenerated situations are :

- A vertex falling either on z_N, z_E or z_T of another triangle
- An edge intersecting z_E or lying on z_T of another triangle
- Two triangles having their z_T interfering (overlapping triangles)

Because of the initial nodes merging at the initialization step, if we have two coincident vertices, it means that at least one of them is a created intersection point.

The five following algorithms are necessary and sufficient bricks to cope with all degeneracies and construct a robust Triangle-Triangle intersection algorithm.

Vertex-Inside Edge intersection (same support line). Let P_0P_1 be the edge, L its length, Q the vertex and $s = \frac{||\mathbf{P_0Q}||}{L}$. As it is assumed that P_0, P_1 and Q are aligned in this test,

$$\begin{cases} Q \in I_z \iff s \in \left]-\frac{\epsilon}{L}, 1+\frac{\epsilon}{L}\right[\\ Q \in z_N \iff s \in \left]-\frac{\epsilon}{L}, \frac{\epsilon}{L}\right[\cup \left]1-\frac{\epsilon}{L}, 1+\frac{\epsilon}{L}\right[\\ Q \in z_E \iff s \in I_z \setminus z_E \end{cases}$$

There is no intersection otherwise.

Vertex-Inside Triangle intersection (coplanar case). The three signed distances between the vertex Q and the edges are computed.

1. $Q \in z_N \iff$ two of them have an absolute value lesser than ϵ.

2. $Q \in z_E \iff$ one of them has an absolute value lesser than ϵ and the two others are strictly greater than ϵ.

3. $Q \in z_T \iff$ the three are strictly greater than ϵ.

4. There is no intersection otherwise.

Non coplanar Edge-Plane intersection. Let P_0P_1 be the edge, L its length. Let′s note Q_0 and $\mathbf{n}$ the director of the plane. The intersection point I is then given by I $= P_0 + s\mathbf{P_0P_1}$, where $s = \frac{\mathbf{n.Q_0P_0}}{\mathbf{n.P_1P_0}}$.
Then the edge intersects the plane if and only if $s \in \left]-\frac{\epsilon}{L}, 1+\frac{\epsilon}{L}\right[$.

Overlapping Edge-Edge intersection. Assuming that four points are supported by the same line, two dot product allows to simply sort the vertices on the line and to retrieve none, one or two vertices for respectively non-overlapping, single-shared vertex (z_N interference) and overlapping cases (z_E

interference).

Coplanar Edge-Edge intersection. Let P_0P_1 and Q_0Q_1 be two edges, L_1 and L_2 their respective lengths. This algorithm is based on the traditional parametric approach [25] for 3D lines :

1. Compute the cross product $\mathbf{w} = \mathbf{P_0P_1} \times \mathbf{Q_0Q_1}$ and its square norm w^2.
2. If $w^2 < \epsilon^2$ then the support lines are parallel, the distance between them is then given by : $\mu = ||\mathbf{P_0Q_0} - \frac{\mathbf{Q_0Q_1}.\mathbf{P_0Q_0}}{||\mathbf{Q_0Q_1}||}||$
   (a) if $\mu < \epsilon$ the edges are supported by the same line. Do the overlapping Edge-Edge test.
   (b) Otherwise there is no intersection
3. Otherwise lines are intersecting at a unique point since they are assumed to be coplanar so it cannot be an agonic situation. Hence the two following computed parameters s and t give the common intersecting point :
$$\begin{cases} P_0 + s\mathbf{P_0P_1} = Q_0 + t\mathbf{Q_0Q_1} \\ s = (\mathbf{P_0Q_0} \times \mathbf{Q_0Q_1}).\mathbf{w}/w^2 \\ t = (\mathbf{P_0P_1} \times \mathbf{Q_0P_0}).\mathbf{w}/w^2 \end{cases}$$
   (a) If $s \in \left]-\frac{\epsilon}{L_1}, 1 + \frac{\epsilon}{L_1}\right[$ and $t \in \left]-\frac{\epsilon}{L_2}, 1 + \frac{\epsilon}{L_2}\right[$, there is an interference (intersection or coincidence).
      i. If $s \in \left]-\frac{\epsilon}{L_1}, \frac{\epsilon}{L_1}\right[$ or $t \in \left]-\frac{\epsilon}{L_2}, \frac{\epsilon}{L_2}\right[$ (z_N interference), no point creation is required and the corresponding vertex is returned as intersecting point.
      ii. If $s \in \left]\frac{\epsilon}{L_1}, 1 - \frac{\epsilon}{L_1}\right[$ and $t \in \left]\frac{\epsilon}{L_2}, 1 - \frac{\epsilon}{L_2}\right[$ (z_E interference), the intersecting point is created.
   (b) In any case the intersecting point is appended to an edge′s node list if it belongs strictly to its interior.

5.4.2. Triangle/Triangle intersection

The triangle/triangle intersection algorithm is inspired from the intersection predicate [16] extended to provide the intersection segments. The six signed distances of each vertex to the planar support of the other triangle are first computed. A quick analysis of these values allows discarding any non-intersecting configuration : either one of the two triangles is at a distance

from the other triangle′s support plane greater than the given tolerance or both are reciprocally positioned as in case b on Fig. 12 meaning that they could share at best a vertex. Fig. 12 illustrates the six possible positions of a triangle regarding the other triangle′s plane.
The algorithm deals with case f apart from the other cases.

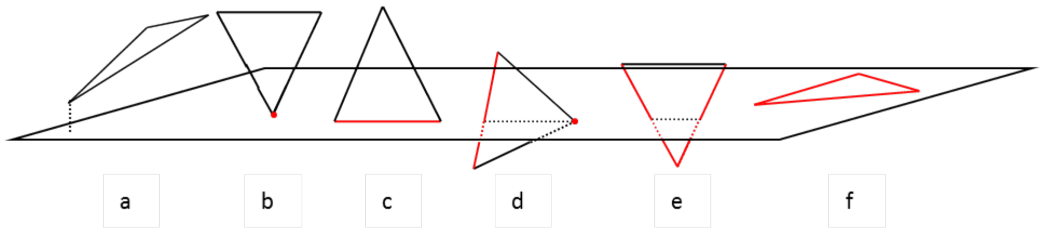

FIGURE 12: one triangle′s entities (in red) that are considered for intersection tests with any triangle supported by the plane.

Case b to e (non-overlapping supporting planes). For each triangle we compute the segment trace by computing the required Plane-Edge intersection points. In case of degeneracies the segment is partially (case d) or fully available (case b and c). So we basically need from zero to four points to create using a Plane-Edge intersection test to get the right intersection trace. We then do an Edge-Edge overlapping test, or a Vertex-Inside-Edge test, or a Vertex-Inside-Triangle test to actually confirm the intersection and update the trace data structure. The trace (point or segment) is appended to the traces of the triangles falling in case d and e. The trace vertices are appended to the appropriate edges′ nodes list.

Case f (both triangles having the same support plane). We first compute the 18 signed distances between each vertex to the other triangle's edges. If none of the triangles is inside the other one, we compute the appropriate Coplanar Edge-Edge intersections to get the set of intersection points and associated segments. This is the most costly case as we can proceed up to 6 Edge-Edge intersection tests.
If any intersection point falls in the edge interior, it is appended to its list. If any intersection point falls inside a triangle interior, the segment is appended to its list as well.

5.4.3. Nodes merging

Coincident nodes need to be merged to ensure overall conformity. The intersection process has produced many redundant points even if a special care has been taken to avoid redundant point creation at the Triangle-Triangle level. Two triangles sharing an edge will indeed produce twice the same intersection point if the shared edge intersects another triangle. This could have been addressed by keeping a record of any Triangle-Edge intersection but this solution is quite expensive in CPU. It has been chosen to proceed with an overall nodes merging (based on a kd-tree structure).

The tolerance used for merging must be the same ϵ as for intersection tests to avoid that any edge or triangle gets out of its initial interference zone. It is ensured that way to not create new intersections due to the moving of nodes. This is a major improvement since the initial version [26], the intersection solving is no more iterative.

5.4.4. Splitting the triangles

The current implementation of this part is based on a constrained Delaunay triangulation [11] applied to each triangle to split with its traces. This approach is very robust but too slow in regard with the targeted applications. In the conclusion section the ideas of the future way of proceeding will be given.

5.5. Building maPH elements with triangular faces

This geometrical step is done by traversing the conformal triangular cloud and gathering the appropriate set of triangles forming enclosed volumes. Since the cloud is conformal, we can create a neighborhood table that gives for any triangle its neighbors sharing an edge with it. But most of the cloud′s edges are non-manifold, i.e. shared by more than two triangles. So we need to select for any triangle and for each orientation of it (both sides) the three right neighbors that will contribute to an enclosed volume. These right neighbors for a given oriented triangle t are those that minimize their dihedral angles with t (cf. Fig. 13.a).

Rather than computing this minimum per edge and per triangle that would lead to a lot of redundant calculations, we sort the triangles (taking into account the two orientations) sharing an edge E_0E_1 (cf. Fig. 13b) like the pages of a book taking arbitrarily one of them as page number one. In this even sorted list, each pairs of consecutive elements gives the right mutual

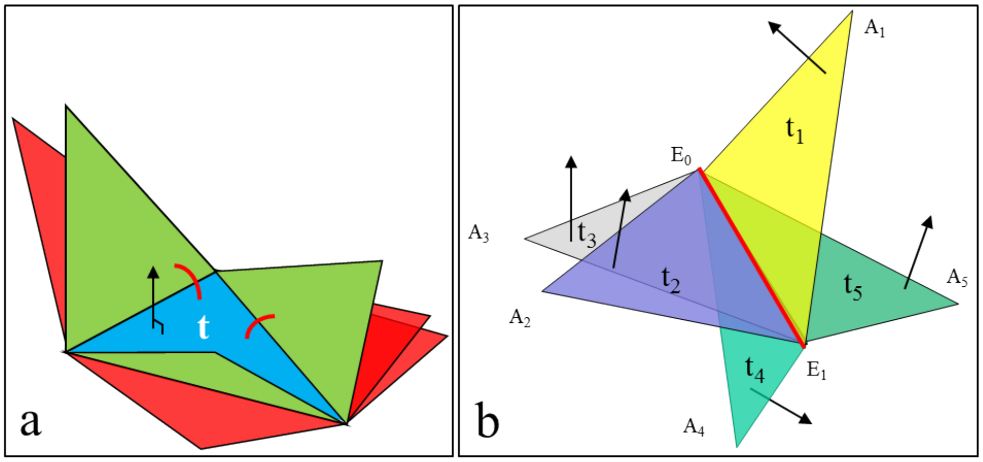

FIGURE 13: (a) t′s right neighbors (green) ; (b) sorting the triangle sharing an edge.

neighbors. We also add to this list a pair joining the opposite triangles of the first and the last one to close the loop like joining the book′s cover. If we note t′ the opposite orientation of t, the five pairs for the case on Fig. 13b will be : (t_1, t_2), (t'_2, t_3), (t'_3, t'_4), (t_4, t'_5), (t_5, t'_1).

For the sorting we need to build a monotonically increasing function q for any triangle pair (t_i, t_j) sharing an edge E_0E_1. Let′s note n_i and n_j their normals.
The algorithm proposed here is more robust than the one proposed in [26]. The split of triangle can indeed generate very poor quality triangles that induce failure on normal computing (cross product error) and orientation predicates. So we rather preserve normals computed on the initial triangles (before splitting) and assign them to split triangles. The inputs for computing the q function are therefore non-degenerated and produce valid results :

$$q = \pi + \mu * ATAN2(s, c) \tag{1}$$

where :

$$\begin{cases} n_k = n_i \times n_j \\ \mu = SIGN(\mathbf{E_0E_1}.\mathbf{n_k}) \\ s = ||\mathbf{n_k}||(sine) \\ c = ||\mathbf{n_i}.\mathbf{n_j}||(cosine) \end{cases}$$

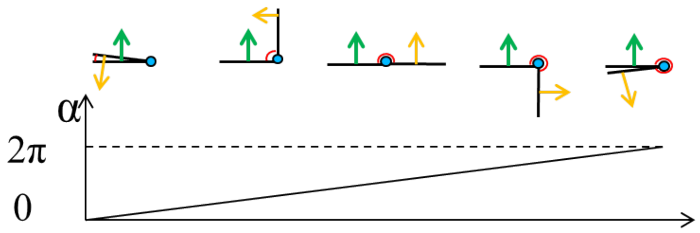

FIGURE 14: monotonic angular function q.

Having a sorted list of triangle pairs, we can create a manifold neighboring for the triangles in both orientations by mutually associating as neighbor the triangles of each pair. This information is then used to traverse the triangles cloud with a flood fill algorithm to gather the triangles by element : any triangle set with the same color is a maPH.

5.6. *Building maPH elements for a prioritized mesh*

When a mesh is prioritized (the yellow mesh on Fig. 15), it means that is has to be embedded into another mesh. In this case, only its outer skin is involved in the intersection process with the entire embedding mesh (Fig. 15b) because it has to be preserved as much as possible. Once its skin has been imprinted into the embedding mesh, its inner topology needs to be modified in the vicinity of this new surface to match it (Fig. 15c) in order to get a conformal result (Fig. 15d).

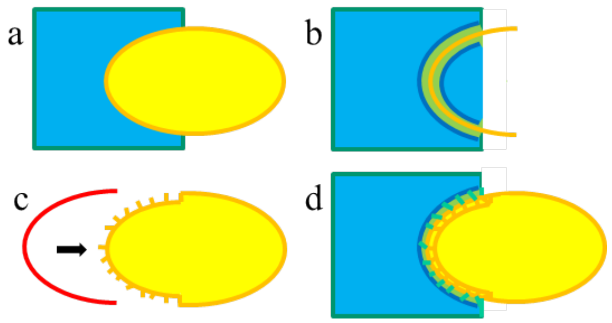

FIGURE 15: (a) a prioritized mesh hiding another mesh ; (b) polygon-polyhedron intersection process ; (c) gluing the new skin and the interior mesh ; (d) conformal result.

Rather than computing the intersections again between the new skin and the interior mesh, a topological approach has been chosen : we gather the triangles on the new skin coming from the same initial polygons. We can then tap individually each open cell on the vicinity of the modified surface by refining the outer contour of these open cells (Fig. 16).

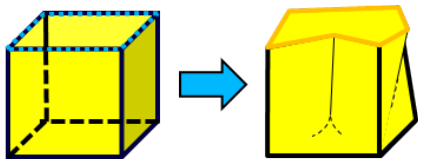

FIGURE 16: Tapping the open cells to match the new polygonal surface.

5.7. Aggregating the triangles : building final aPH elements

For a given maPH with triangular faces, we want to aggregate its triangles to get back to a polygonal representation. The resulting polygons must be admissible and conformity between them must be preserved so we gather the triangles satisfying the following rules :

1. They have the same polygon ancestor (using ancPG mapping, cf. 1.6)
2. They form a connected set with a single boundary loop
3. They are shared by the same two maPHs (for preserving conformity)

We first create a table giving the two polyhedral cells that share each triangle. For each triangle set coming from the same initial polygon, we separate the triangles into smaller subsets that share the same cells. A flood fill algorithm ensures to have at the end only subsets with a single boundary loop.

Optional Concave polygons split. Because concavity can be an issue for some applications (e.g. Finite Volume Method for CFD) requiring having the center of the faces inside the face to be more accurate, an extra feature has been added to optionally split the concave sets into convex ones. Because the entities to split are triangulations rather than meshes the simple following algorithm is enough to get a natural cut for the concave polygons.
The algorithm is depicted on Fig. 18 : it starts by finding the worst concavity over the contour (point C). From the oriented edge E ending on C (orange edge) the triangles sharing C are traversed to detect the best cutting edge :

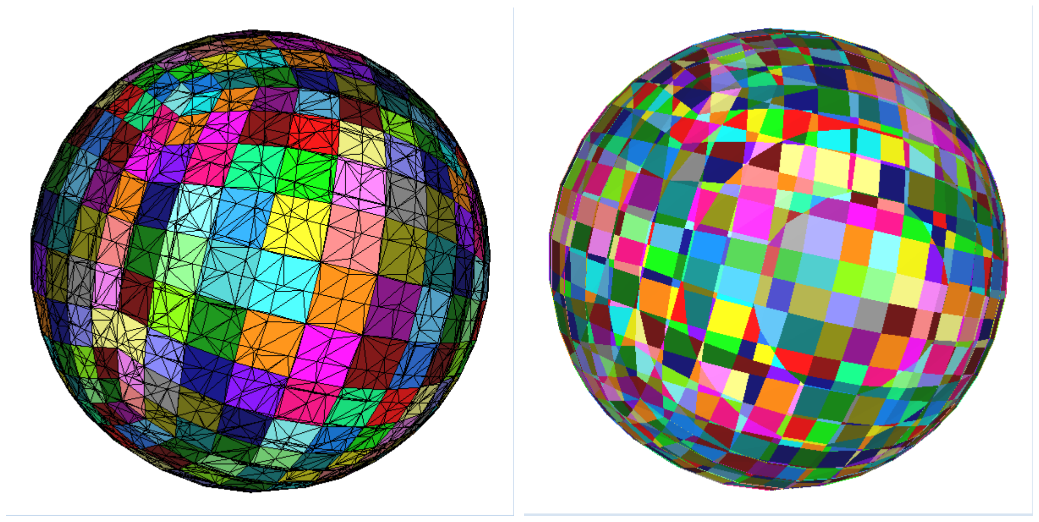

FIGURE 17: (a) split triangles on a sphere colored by initial quads ; (b) convex aggregation ensuring conformity with the imprinting mesh.

the one which makes an angle smaller but as close as possible to 180° with E (green edge E′). If E′ ends on a contour node, a convex bit has been obtained and we start again with the remaining contour. Otherwise E becomes E′ and the cutting is carried on.
At the end of this aggregation step, elements are maPH with polygonal faces and we have a new mapping ancPGa giving for each maPH face its original polygon ancestor.

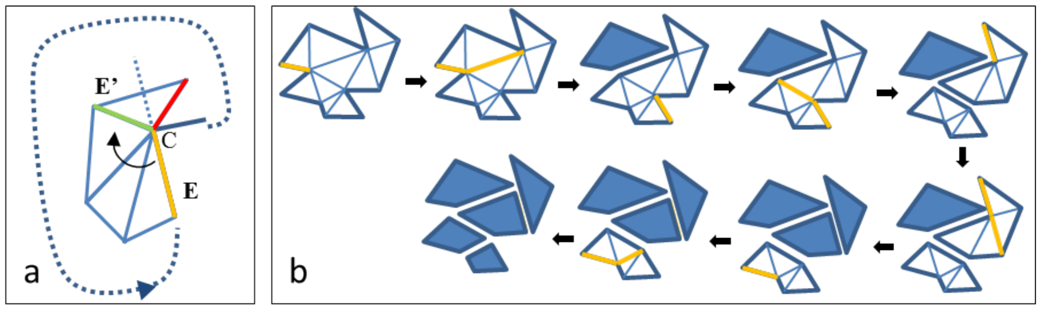

FIGURE 18: (a) getting the best cutting edge ; (b) split process.

5.8. Classification

The classification is not done individually for each maPH. This would imply to geometrically localize the maPH which can be costly. We rather classify them topologically in two steps. First maPHs attached to the pure skin (cf. 1.5) are considered and classified and second the other elements are straightforwardly classified by a flood fill algorithm for which the delimiting boundaries are pure skin polygons (Fig. 19).

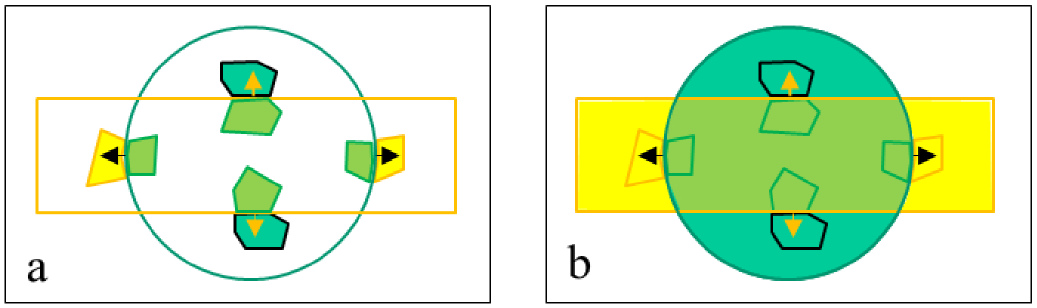

FIGURE 19: (a) skin elements classification ; (b) flood filling.

The classification predicate is based on skin′s normal orientation in regards with the considered skin elements. If a skin element has the skin normal inward (yellow and blue elements on Fig. 19), this element is localized without ambiguity as being outside the mesh having the considered skin. Elements sharing the skin face are therefore in the common part (green elements). If no outside elements are found, it means that the input meshes are two meshes of the same domain : their skin fully overlap. We can end up with two zones (rather than three for a partial overlapping) if one mesh is fully inside the other one or if they have a partial contact, i.e. their skin is locally supported by the same surface (and none is inside the other one).

5.9. Conservative remapping

Let′s consider two intersecting meshes M_1 and M_2 (cf. Fig. 20a) and the mesh $M_1' = (M_1 \setminus M_2) \cup (M_1 \cap M_2)$ (cf. Fig. 20b). M_1' represents the new mesh of the domain occupied by M_1. Having the mapping ancPGa (cf. 5.7), we can detect in M_1' any polygon bit from M_1. We can then use the neighbor graph for M_1' cells to retrieve the mapping $M_1 \leftrightarrow M_1'$ using again a flood fill algorithm for which the delimiting boundaries are M_1 polygon bits. Swapping

indices in the preceding gives the $M_2 \leftrightarrow M_2'$ mapping.
The break down for a given M_1 cell is shown on figure 20c.

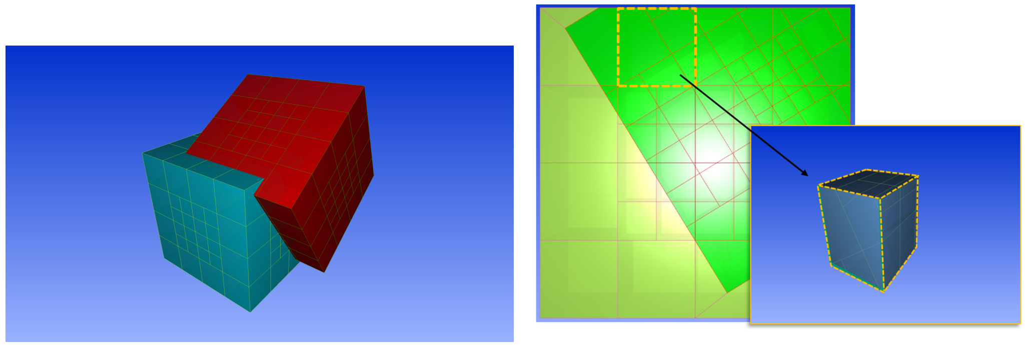

FIGURE 20: (a) Two intersecting octrees M_1 and M_2 ; (b)M_1' ; (c) one M_1 cell's break down.

6. Applications and results

The method has been implemented in templatized C++ in the Cassiopee Software [17]. This section gives some test results for several ascending size problems covering the different boolean operations on different type of operands. All the test have been done using 1.e-13 as absolut tolerance and all the polygonal aggregations are convex.
All the tests and performance measures have been done on a single Intel Xeon core @ 2.8 GHz.

6.1. Prioritized union of two volume meshes

We consider a spherical boundary layer mesh immersed in a octal background mesh. This is the simplified typical configuration for a CFD viscous external flow around a body (cf. Fig. 21a). Before processing it, the octree has been decimated by discarding any cell having a node inside the inner sphere because the interior of the boby is irrelevant for that kind of applications. The conformal result is depicted in Fig. 21b and the performances are available in table 1.

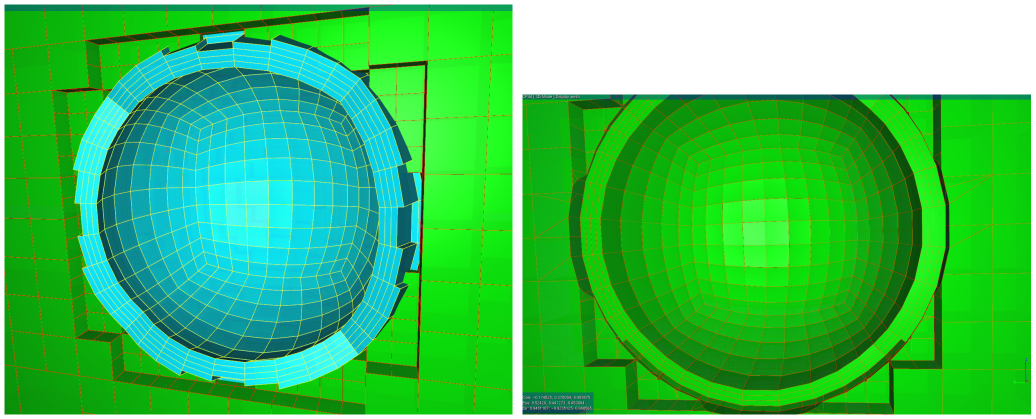

FIGURE 21: (a) a spherical boundary layer mesh appended into an octree ; (b) conformal union.

Step	CPU time (s)
1) Init (16,182 polygons)	0.11
2) Triangulation (6,174 polygons)	0.09
3) Intersections(13,164 triangles)	0.30
4) Triangle split(39,208 triangles)	1.05
5) Cell construct (2,150 polyhedra)	1.07
6) Polygons Aggregation	0.64
7) Classification	0.27
Total	**3.53**

TABLE 1: Performances for the 6.1 case.

6.2. Intersection, two volume meshes

Let′s consider now two identical conformal polyhedral mesh operands M_1 and M_2 obtained from an octree having a refinement at its centroid (cf. Fig. 22) in an intersecting position. They both have 1,576 polyhedra.

The figures 23 shows the resulting volume meshes of the difference and the non-prioritized intersection of them. It contains 4,821 polyhedra which is less than the double of the initial number of elements. With a tetrahedral approach (i.e. splitting all the polyhedra into tetraherdons first) we would en up with an increase of at least one order of magnitude. The polyhedral approach si therefore better fitted to ouput the resulting mesh for boolean operations. The performance are shown in the table 2.

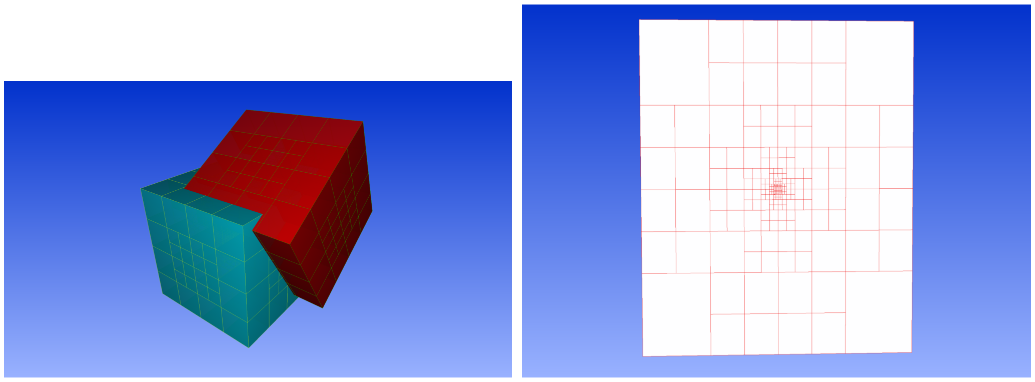

FIGURE 22: (a) Two intersecting octrees M_1 (blue) and M_2 (red) ;(b) octree operand′s inner structure.

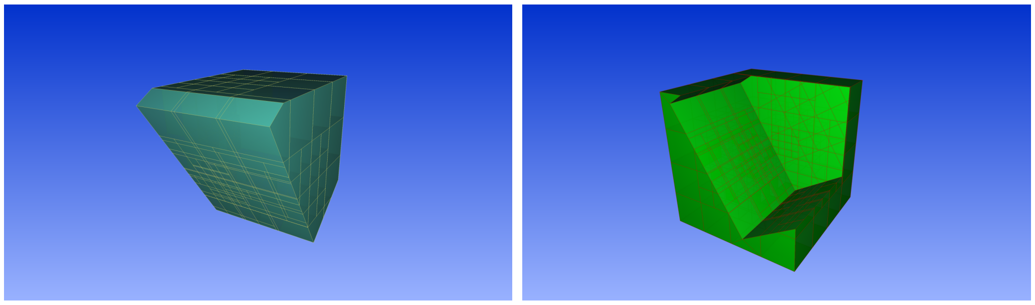

FIGURE 23: (a) $M_1 \cap M_2$; (b) $M_1 \setminus M_2$.

6.3. Union of a surface mesh with a volume mesh

We consider a light triangulated cat surface with 740 triangles (Fig. 24a) fully immersed in a Cartesian grid (29,440 hexs, 91,300 quads). The algorithm generates here the conformal union, i.e. the surface is integrated into the mesh. A view of the clipped mesh is depicted in Fig. 24b and the performance details are given in table 3. A 5.8 speed-up factor has been obtained since [26].

6.4. Union of two surface meshes

Even though this algorithm is more dedicated for cases with at least one volume mesh input, it is interesting to compare it with a pure surface algorithm [21]. We consider here a case that has been treated in the paper,

Step	CPU time (s)
1) Init (11,784 polygons)	0.03
2) Triangulation (10,322 polygons)	0.21
3) Intersections(22,757 triangles)	0.67
4) Triangle split(58,207 triangles)	1.30
5) Cell construct (5,116 polyhedra)	1.35
6) Polygons Aggregation	0.6
7) Classification	0.18
Total	**4.34**

Table 2: Performances for the 6.2 case.

Step	CPU time (s)
1) Init (92,000 polygons)	0.37
2) Triangulation (25,460 polygons)	0.02
3) Intersections(50,182 triangles)	1.29
4) Triangle split(111,098 triangles)	2.72
5) Cell construct (10,593 polyhedra)	2.48
6) Polygons Aggregation	1.4
7) Classification	1.03
Total	**9.31**

Table 3: Performances for the 6.3 case.

the union of a Buddha and a lion shown on Fig. 25a. The case is approximatively 1.48 million triangles. In order to make a fair comparison, polygon aggregation has been disabled as the the output mesh is triangular. The time reported by the author for this case is 1.027 sec. for a multithreaded run (4 threads) on a slightly slower processor (@2.7 GHz). Considering that our implementation is sequential, there code is around eight times faster (cf. table 4) than our current implementation. This is mainly due to the cell construct and classification steps that are currently not optimized for surface meshes. But one order of magnitude is acceptable regarding the versatility of our algorithm and the remaining ways of improvement given in the conclusion. The algorithm shows a good robustness with this case having triangles of very different size at the bottom of the lion model (as depicted in 25b).

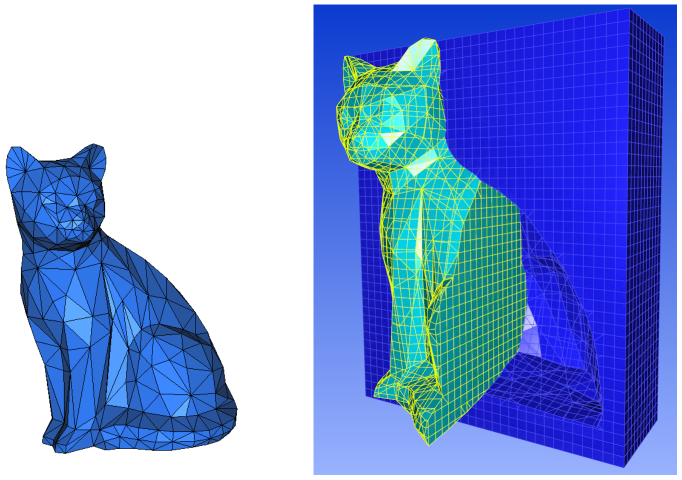

FIGURE 24: (a) triangular surface mesh ; (b) conformal union with a cartesian grid.

Step	CPU time (s)
1) Init (1,487,474 triangles)	6.89
2) Triangulation (584,426 polygons)	0.00
3) Intersections(1,051,966 triangles)	7.15
4) Triangle split(3,441,290 triangles)	4.53
5) Cell construct	12.28
6) Classification	2.19
Total	**32.04**

TABLE 4: Performances for the 6.4 case.

6.5. Difference between a volume mesh and a surface mesh : cut-cell meshing No 1

Here we present the bottom part of a landgear (cf. Fig. 26a), having 116,000 triangles immersed into an octal mesh with 303,000 polyhedra (1,067,178 polygons). The resulting cut-cell mesh is shown depicted in Fig. 27a. The performances (cf. table 5 have been significantly improved (speed up factor 20) since [26]. A lot of ways of improvement remains though as expressed in the conclusion.

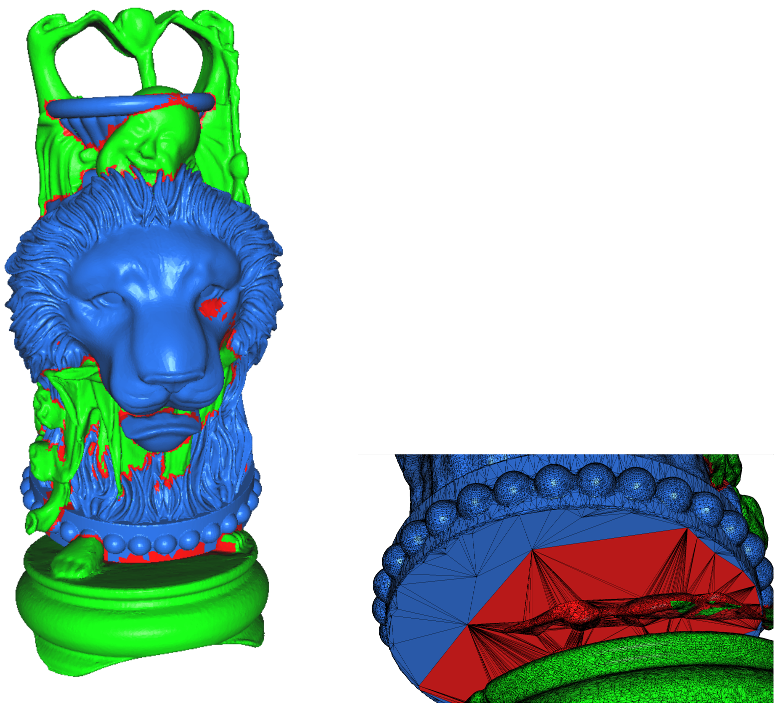

FIGURE 25: (a) Union of a Buddha and a lion (b) zoom on a critical region.

6.6. Difference between a volume mesh and a surface mesh : cut-cell meshing No 2

This case is a more complex landgear model : 831,000 triangles (cf. Fig. 26b). The immersing octree has 127,000 polyhedra (450,000 polygons). The resulting cut-cell mesh is shown depicted in Fig. 27b. It took 353 sec. to compute this cut-cell mesh as shown in table 6.

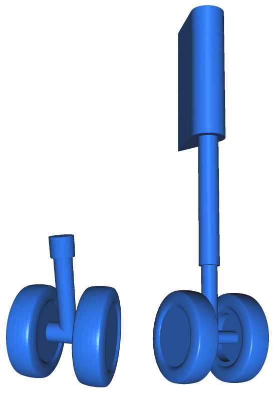
FIGURE 26: (a) bottom part of a landgear (b) complete model.

Step	CPU time (s)
1) Init (1,124,000 polygons)	3.11
2) Triangulation (584,426 polygons)	0.32
3) Intersections(1,051,966 triangles)	38.90
4) Triangle split(3,441,290 triangles)	101.84
5) Cell construct (218,951 polyhedra)	83.77
6) Polygons Aggregation	45.56
7) Classification	21.66
Total	**295.16**

TABLE 5: Performances for the 6.5 case.

Step	CPU time (s)
1) Init (1,282,282 polygons)	4.3
2) Triangulation (1,223,998 polygons)	6.27
3) Intersections(1,664,962 triangles)	48.98
4) Triangle split(4,186,360 triangles)	113.98
5) Cell construct (161,192 polyhedra)	100.8
6) Polygons Aggregation	55.67
7) Classification	23.35
Total	**353.35**

TABLE 6: Performances for the 6.6 case.

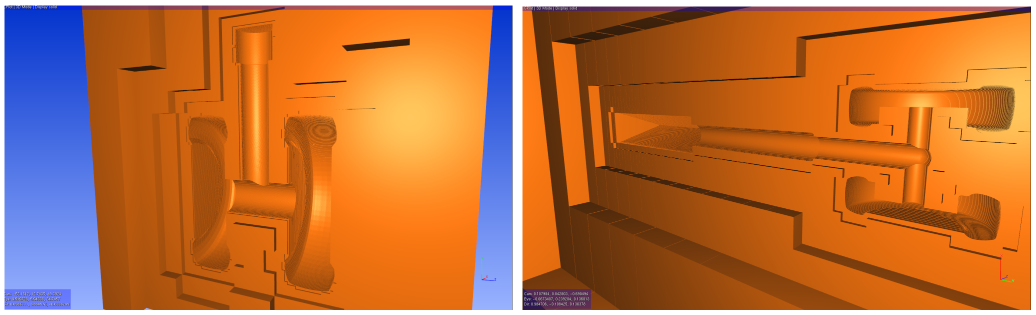

FIGURE 27: clipped view of the cut-cell meshes

7. Conclusion and perspectives

An algorithm with a O(nlog(n)) algorithmic complexity has been described to resolve boolean operations for arbitrary polyhedral meshes and has been implemented successfully. The current tests have shown that even if some good progresses have been made since the first implementation tests, the current performances are roughly around 1,000 polyhedra constructed per second which is not good enough to deal with transient industrial CFD cases. As a Pre or Post tool, the performance can already be considered as satisfactory though. But there are plenty of ways of improvements.

The first issue of this algorithm comes from the fact that the triangular intersection kernel has been designed to handle any kind of configuration. So it could handle upon entry some non-conformal meshes as long as they are consistently oriented. This is because there are no assumption on the input connectivity, the set of input triangles (once the polygons have been triangulated) is treated as a soup. This is powerful and allows covering a wider range of cases but efficiency would be significantly improved by using this connectivity information for instance for the localization as described in [12] or [2]. Almost all targeted 3D applications deal with conformal meshes upon entry.

The second issue comes from the triangles split stage. Using a constrained-Delaunay algorithm is obviously too heavy for the purpose and is the most sensitive part to robustness issues. Using ear-clipping techniques could be an improvement [ebe98, hel98], but the true enhancement will be to find a solution that does not generate triangles at all. The polygons could still be triangulated to compute the intersections in order to facilitate the intersection kernel design, but we could imagine other ways of cutting the polygons along the intersection traces to work on a polygon cloud rather than a heavy triangular cloud. This would remove the reaggregation stage, woud light the polygon sorting and the cell construction stage would deliver directly the maPHs.

Another way of improvement would be to somehow cut and classify simultaneously the polygon bits. This would avoid to create useless elements, meaning elements rejected at the classification stage.

When all these algorithm changes will be implemented and validated, the code will be multithreaded and specialized to focus on the conservative problem specifically, keeping as an option to output or not the geometry upon exit.

8. References

[1] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, C.R. Wilson, Conservative interpolation between unstructured meshes via supermesh construction, Comput. Methods Appl. Mech. Engrg., Vol. 198, pp. 2632-2642 (2009).

[2] P.E. Farrell, J.R. Maddison, Conservative interpolation between volume meshes by local Galerkin projection, Comput. Methods Appl. Mech. Engrg. (2010).

[3] S. Menon, D.P. Schmidt, Conservative interpolation on unstructured polyhedral meshes : An extension of the supermesh approach to cell-centered finite-volume variables, Comput. Methods Appl. Mech. Engrg., Vol. 200, pp. 2797-2804 (2011).

[4] D.H. Eberly, 3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics, fist ed., Morgan Kaufmann, 2000, ISBN 1558605932.

[5] J. Grandy, Conservative Remapping and Region Overlays by Intersecting Arbitrary Polyhedra, JCP, Vol. 148, pp. 433-466 (1999).

[6] P. Brenner, Three Dimensional Aerodynamics with Moving bodies applied to solid propellant, AIAA, 27th Joint Propulsion Conf. (1991).

[7] P. Brenner, Simulation du mouvement relatif de corps soumis un coulement instationnaire par une mthode de chevauchement de maillages, AGARD FDP Symposium on Progress and Challenges in CFD Methods and Algorithms (1995).

[8] D. Badouel, G. Hgron, Opérations booléennes sur polyèdres : évaluation d′arbres CGS. [Research Report] RR-0839 (1988).

[9] M.O. Benouamer. Opérations booléennes sur les polyèdres reprsentés par leurs frontières et imprécisions numériques, Ph.D. Dissertation, Ecole Nationale Supérieure des Mines de Saint-Etienne ; Université Jean Monnet - Saint-Etienne (1993).

[10] E.Levent Gursoz, Y. Choi, F.B. Prinz, Boolean set operations on non-manifold boundary representation objects, Computer-Aided Design, Vol. 23, Issue 1, pp 33-39 (1991).

[11] H. Borouchaki, Paul L. George, Aspects of 2-D Delaunay mesh generation, IJNME, Vol. 40, pp. 1957-1975 (1997).

[12] F. Alauzet, M. Mehrenberger, P1-conservative solution interpolation on unstructured triangular meshes, IJNME, Vol. 84, pp. 1552-1588, 10.1002/nme.2951 (2010).

[13] L.G. Margolin, Mikhail Shashkov, Second-order sign-preserving conservative interpolation (remapping) on general grids, JCP, Vol. 184, pp. 266-298 (2003).

[14] R. Garimella, M. Kucharik, M. Shashkov, An efficient linearity and bound preserving conservative interpolation (remapping) on polyhedral meshes, Computer and Fluids, Vol. 36, pp. 224-237 (2007).

[15] Z.J. Wang, V. Parthasarathy, and N. Hariharan, A fully automated Chimera methodology for multiple moving body problems, 36th AIAA Aerospace Sciences Metting and Exhibit (1997).

[16] O. Devillers ; P. Guigue, Faster Triangle-Triangle Intersection Tests, RR-4488 INRIA (2002).

[17] C. Benoît, S. Peron, S. Landier, Cassiopee : a CFD pre- and post-processing tool, Aerospace Science and Technology, Vol. pp. 45, 272-243 (2015).

[18] J.R. Schewchuk, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates, Discrete and Computational Geometry, Vol. 18, pp. 303-363 (1997).

[19] F. Alauzet, A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes, Comput. Methods Appl. Mech. Engrg. (2015).

[20] H. Edelsbrunner, E.P. Mcke, Simulation of simplicity : a technique to cope with degenerate cases in geometric algorithms, ACM Transactions on Graphics (TOG), Vol. 9, no 1, pp. 66-104 (1990).

[21] M. Douze, J-S. Franco, B. Raffin, QuickCSG : Arbitrary and Faster Boolean Combinations of N Solids, Diss. Inria-Research Centre GrenobleRhne-Alpes (2015).

[22] D. Pavi, M. Campen, L. Kobbelt, Hybrid booleans, Computer Graphics Forum, Vol. 29, No. 1, Blackwell Publishing Ltd (2010).

[23] D. Eberly, Triangulation by ear clipping, Geometric Tools, LLC (1998).

[24] M. Held, Efficient and reliable triangulation of polygons, Computer Graphics International, Proceedings IEEE (1998).

[25] R. Goldman, Graphics Gems, A. S. Glassner Ed., pp. 304 (1990).

[26] S. Landier, Boolean Operations on Arbitrary Polyhedral Meshes, Procedia Engineering, Vol. 124, pp. 200-212 (2015).