Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 124 (2015) 200 – 212

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Boolean operations on arbitrary polyhedral meshes

Sâm Landier[a,]*

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

Abstract

A floating-point arithmetic algorithm designed for solving usual boolean operations (intersection, union, and difference) on arbitrary polyhedral meshes is described in this paper. It can be used in many pre- and post-processing applications in computational physics (e.g. cut-cell volume mesh generation or high order conservative remapping). The method provides conformal polyhedral meshes upon exit. The core idea is to triangulate the polygons, solve the intersections at the triangular level, reconstruct the polyhedra from the cloud of conformal triangles and then re-aggregate their triangular faces 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 preliminary results are shown.

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24)

Keywords: Boolean Operations; Polyhedral Meshes; Cell Reconstruction; Conformity, Conservative Remapping; Cut-cell Meshing; Constrained Delaunay Triangulation.

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) 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) depicted on fig. 1.a. A typical target application is related to

* Corresponding author. Tel.: +33-1-46- 73-46-24; fax: +33-1-46- 73-41-66.
E-mail address: sam.landier@onera.fr

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24)
doi:10.1016/j.proeng.2015.10.133

conservative remapping of some solution fields from one mesh to another [1,2,3,5,6,7,12,13,14]. Some of these works resolve completely the intersections between the source and target meshes [1,2,3,5,12]. In this case the computational cost is potentially important but necessary because all the intersections between the polygons must be resolved in the intersecting zone (in green fig. 1).

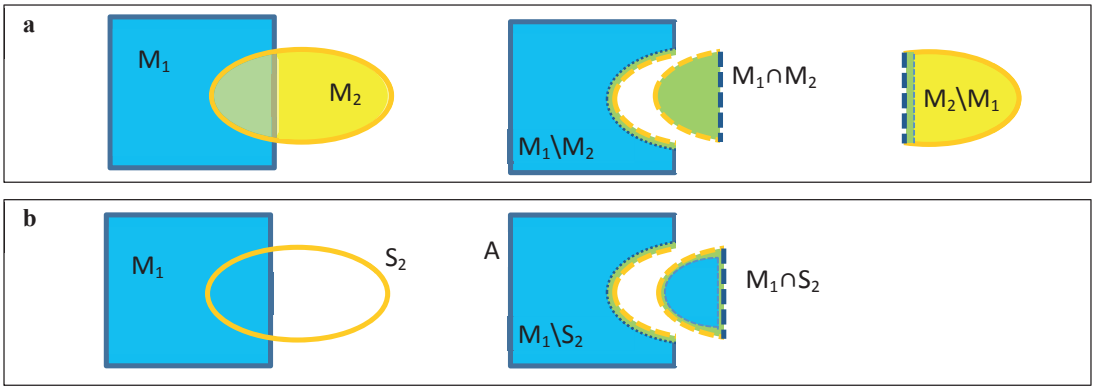

Fig. 1: boolean operations: (a) for volume meshes; (b) for a volume mesh vs a surface mesh

Some other applications like Chimera blanking [15] or any overlapping grid technique privileging some mesh over other ones [6,7], don't need to fully solve the intersection zone: these applications require solving a boolean operation on mixed-type operands (a volume mesh versus a surface mesh, cf. fig. 1.b) or a pure surface problem (two polyhedra).

The previously cited works dealing with intersections propose geometric algorithms specifically designed to solve conservative remapping applications and the resulting meshes (mostly tetrahedral) are generally an intermediary result that is not given upon exit. Our initial goal is to design a geometric kernel that can handle multiple applications giving an explicit true polyhedral mesh representation.

Among the more advanced 3D algorithms [1,2,6,7], Farrell et al. [1,2] have named *supermesh* the mesh S where all intersections have been solved. Its construction is a necessary step for conservative remapping (or for any underlying boolean operation on meshes) and it is very similar to the approach described in this paper. Their 2D version [1] considers the intersection solving as a *global* Constrained Delaunay Triangulation (CDT) problem: all the edge intersections are resolved and the whole set of split edges is given to a CDT engine. The elements mapping $M_1 \leftrightarrow S$ and $M_2 \leftrightarrow S$ are constructed upon exit to identify the cells decomposition for conservative remapping in both directions. First, although this point of view is applicable in 3D as we will see later on, the 3D version [2] adopts a *local* approach to compute the supermesh which induces redundant intersection computations since they are applied individually per target polyhedron. So for instance two adjacent elements in M_1 will cut some same M_2 elements more than once. Second, the clipping approach inspired by Eberly [4] used as the atomic intersection operation induces that the 3D version only deals with convex elements. 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]. Again dealing with atomic intersections at a volume element level is more costly than doing it at the polygon level because of redundant calculations induced on the shared polygons. Dealing with intersections at the face level avoids the requirement of such volume decomposition.

Another mature and efficient algorithm developed in FLUSEPA3D CFD code by Brenner [6,7] deals with polyhedral mesh intersections over overlapping grids. It is done at a face level (between polygons broken down into triangles) but 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.

Previous efforts directly related to boolean operations have been investigated but designed to handle polyhedra operands [8,9] or mixed type polygons/polyhedra [10] operands. The latter handles a larger range of non-manifold topologies (including holes on polygons) than the proposed algorithm but they all do the classification of entities (i.e. selecting the appropriate entities to answer the boolean operation) in a geometric and individual way using an octree decomposition to create the "in or out" information. This approach cannot apply efficiently when operands are meshes with a large number of polyhedra. A topological approach using the neighboring information is simpler and more efficient.

2. Admissible topologies

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

2.1. 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 Π 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)

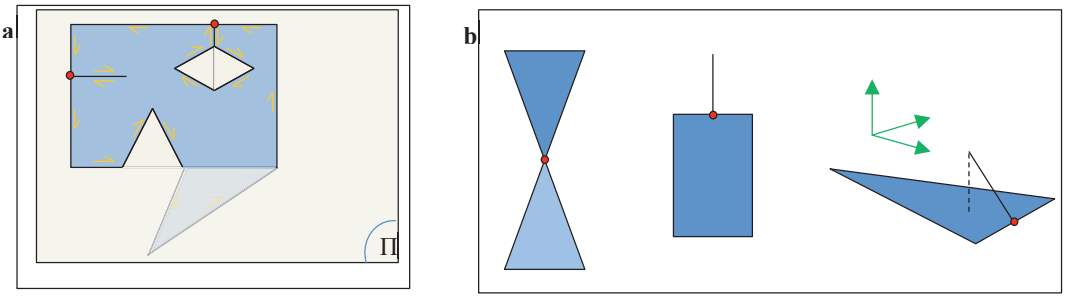

Fig. 2: (a) aPG; (b) three non-admissible polygons

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 vertex list of size q $\{S_i\}$ 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 figure 2.b are not admissible.

2.2. 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)

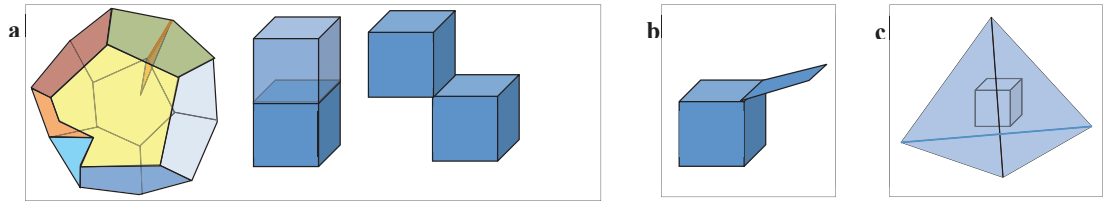

Fig. 3: (a) and (b) some aPHs; (c) non-admissible case: a tetrahedron with cubic hole

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. 3.a are four separate maPH.

2.3. 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 3.a) because any outside one (quadrangular face on the cube fig. 3.b) 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.

3. 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:

- Intersection: $M_1 \cap M_2 =$ common elements to both M_1 and M_2
- Difference: $M_1 \setminus M_2 =$ elements inside M_1 and outside M_2
- Union: $M_1 \cup M_2 =$ 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.

4. The algorithm

4.1. 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 giving a consistent orientation of the operands' skins and reduces the zone of interest to the intersecting zone. 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. So we can sum up the whole process as depicted in the fig. 4:

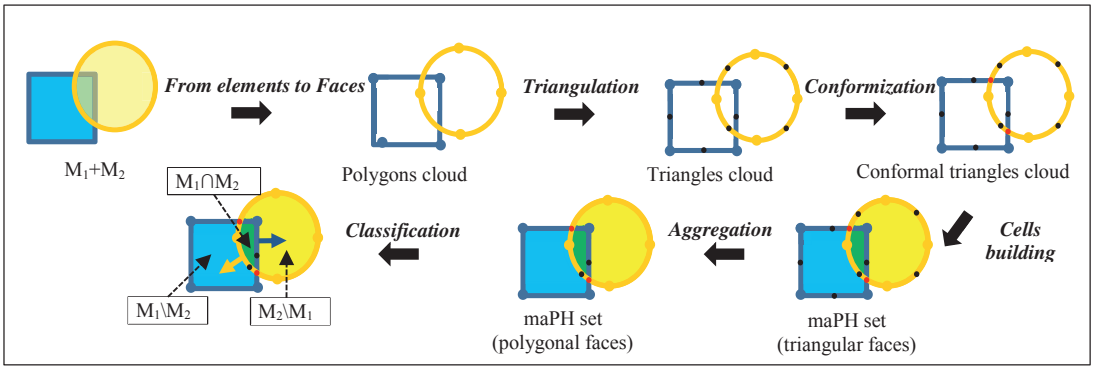

Fig. 4: Algorithm's scheme

4.2. Initializing and Pre-conditioning

Initializing starts by flagging the outer polygons as "pure skin" of M_1 or M_2. This is simply achieved traversing them through all the polyhedra and counting the number of times they are reached: any polygon with this counter equal to one is a skin polygon (remember non-manifold faces are duplicated).

In order to reduce the overall computation cost which is mainly due to the intersections processing, we focus on the intersection zone and keep apart the non-involved elements that we will only consider at the classification step. 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 fine refinement of the obtained set is then done using two bounding box trees, one for each mesh. Any element of mesh M_1 and M_2 having their boxes intersecting are kept, discarded otherwise. We get a set of intersecting zones; one of them is depicted on fig. 5.b.

The new outer polygons (those attached to discarded elements) are flagged as "connection skin" polygons. Then 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 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 pre-conditioning. They will be considered in future work.

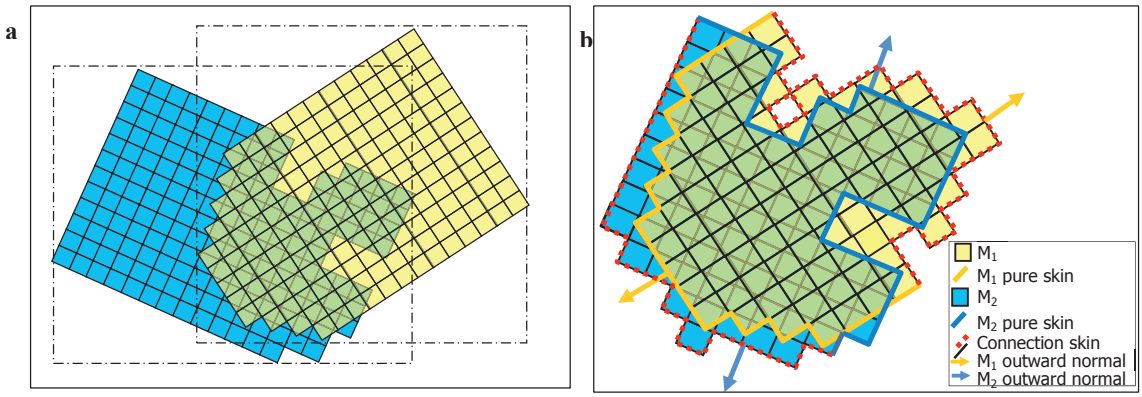

Fig. 5: (a) initial aMHs M_1 and M_2; (b) working set for intersection after initializing a pre-conditioning.

4.3. From the polygons to a conformal triangles cloud

A 2D-Constrained Delaunay Triangulation operator [11] is applied to the polygons cloud to get a triangular representation. This can always be achieved with admissible polygons as defined. During this step a mapping table *ancPG* gives for each triangle the polygon it belongs to. A Triangle intersection solving operator called *Conformizer* and described in the next paragraph, is then applied to this triangles cloud in order to get a conformal triangles cloud. The Conformizer provides also for any created conformal triangle its triangle ancestor as a mapping. Combining ancPG and this latter mapping, we can build a new mapping *ancPG** associating the split triangles to the original polygons that will be used to aggregate back the triangles to make aPGs from them as described in §4.5.

4.3.1. Conformizer Operator

If we consider a cloud of intersecting triangles in arbitrary positions (cf. fig. 6.a), 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 6.b). An usual split sequence looks like the one depicted fig. 6.c and this algorithm do it this way but 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 three-step algorithm: 1) Computing and storing the traces for each triangles, 2) Splitting each intersected triangles and 3) Swapping some edges to recover the intersection traces. The last two steps are equivalent to do a constrained Delaunay triangulation for each intersected triangle [11]. The drawback of this approach is the storage of the traces and the extra computation cost due to edge swapping but the benefit is a very high rate of convergence to reach the resolved state compared to the former approach: only very few global iterations of the Conformizer are required (typically two or three) to resolve all the intersections whatever the size of the cloud. The first reason is that all the intersections are processed at once for each intersected triangle t, whereas t will undergo n iterations with a intersection solver processing one intersection at a time if n is the number of triangles intersecting t. For that kind of solvers, t and its descendants (split triangles) will need to recheck at each iteration the potential intersection candidates by making some redundant queries to the search structure (e.g. a bounding box tree). The second reason is that adding a swap procedure in order to recover the correct traces prevents to introduce parasite intersection computations for the next iteration. Swapping the orange edge depicted on fig. 6.d will prevent to get to the situation on fig. 6.e. Some potential parasite edges (like the yellow one on fig. 6.d) being too close to a vertex of one of its triangles (i.e. if distance d is smaller than the given intersection tolerance on fig. 6.d) might be swapped as well to anticipate a future split. This is implemented as swapping any swappable edge that improves the quality of the triangles sharing them. As a last remark on this topic, taking care of processing the splitting points by triangle provenance will reduce the amount of required swapping.

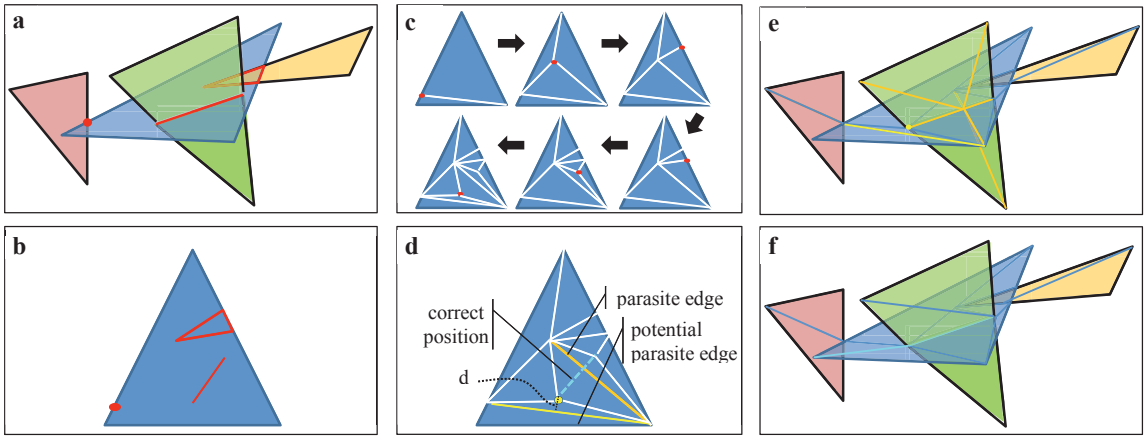

Fig. 6: (a) Triangles cloud; (b) Intersections traces; (c) split sequence; (d) parasite splits; (e) Over split cloud; (f) Correct split cloud

A last step is to clean the obtained cloud by collapsing any triangles with an edge smaller than the given tolerance and discard any degenerated triangle. Intersecting points are privileged (attractors) compared to existing vertices when merging in order to not disturb the intersection profile (cf. fig. 7). The Conformizer operator requires more than an iteration to converge because of this merging step: moving the points might induce new triangles intersections to solve. Its algorithmic complexity is also O(nlog(n)) where n is the number of triangles because of the bounding box localizer.

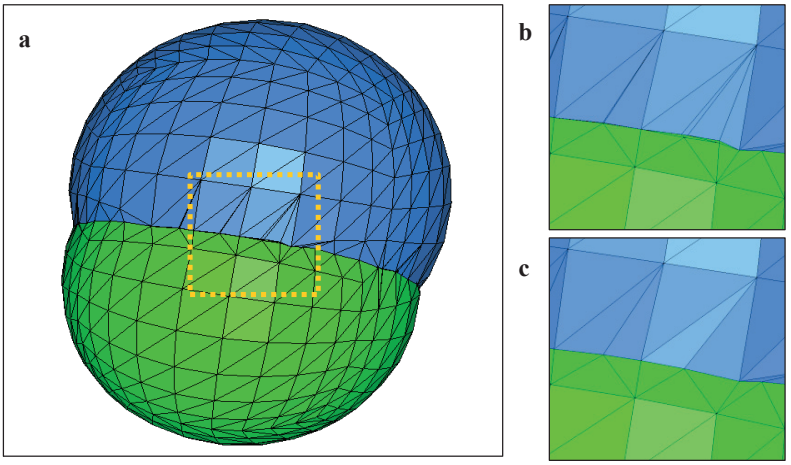

Fig. 7: (a) two intersecting aPH; (b) before merging; (c) after merging

4.4. 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. 8.a).

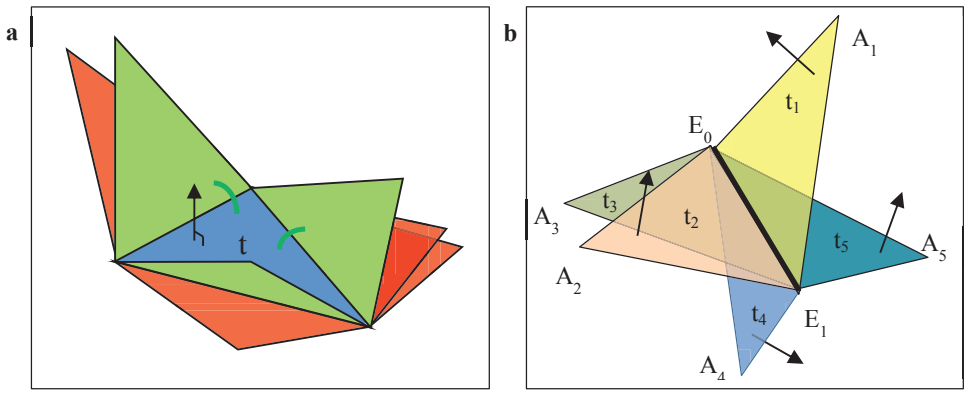

Fig. 8: (a) t's right neighbors (green); (b) sorting the triangle sharing an edge

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. 8.b) 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 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.8.b 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 build a monotonically increasing function q for any triangle pair (t_i, t_j) which writes:

$$q(t_i, t_j) = -sign(\alpha) \times atan2(s, c)$$

Where s and c are sine and cosine of the angle between t_i and t_j normals, and α writes:

$$\alpha = [E_0, E_1, A_i, A_j] = \begin{vmatrix} E_{0x} & E_{0y} & E_{0z} & 1 \\ E_{1x} & E_{1y} & E_{1z} & 1 \\ A_{ix} & A_{iy} & A_{iz} & 1 \\ A_{jx} & A_{jy} & A_{jz} & 1 \end{vmatrix} = \begin{vmatrix} E_{0x} - A_{jx} & E_{0y} - A_{jy} & E_{0z} - A_{jz} \\ E_{1x} - A_{jx} & E_{1y} - A_{jy} & E_{1z} - A_{jz} \\ A_{ix} - A_{jx} & A_{iy} - A_{jy} & A_{iz} - A_{jz} \end{vmatrix}$$

An interpretation of α is to give A_j position in regard with $(E_0E_1A_i)$ plane: a negative value means it is under the plane, a positive value implies it is above and zero means the four points are coplanar.

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 coloring algorithm to gather the triangles by element: any triangle set with the same color is a maPH.

4.5. 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:

- They have the same polygon ancestor (using ancPG* mapping, cf. 4.3)
- They form a connected set with a single boundary loop
- They are shared by the same two maPHs (for preserving conformity)

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 onto 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. 9: 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: 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.

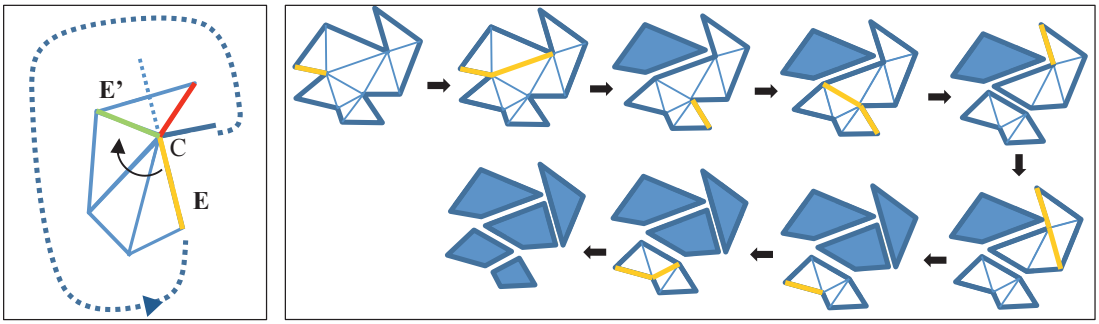

Fig. 9: (a) getting the best cutting edge; (b) split process

4.6. 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. 4.2) are considered and classified and second the other elements are straightforwardly classified by a coloring algorithm for which the delimiting boundaries are pure skin polygons (fig. 10).

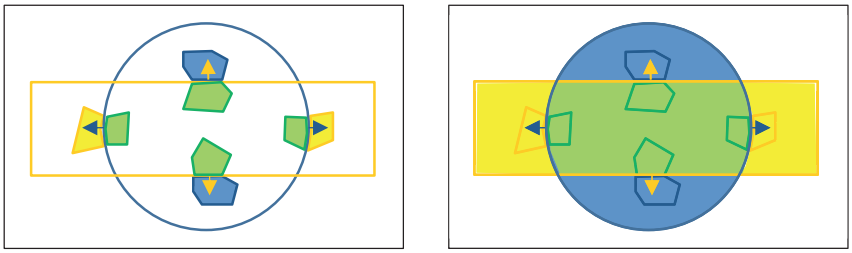

Fig. 10: (a) skin elements classification; (b) filling by coloring

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. 10), 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).

4.7. Conservative remapping

Let's consider the mesh $M_1' = (M_1 \backslash M_2) \cup (M_1 \cap M_2)$. M_1' represents the new mesh of the domain occupied by M_1. Having the mapping ancPGa (cf. 4.5), 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 coloring algorithm for which the delimiting boundaries are M_1 polygon bits. Swapping indices in the preceding gives the $M_2 \leftrightarrow M_2'$ mapping.

5. Applications and results

The method has been implemented in templatized C++ in the Cassiopee Software [17]. As preliminary results let's start by illustrate a cut-cell meshing application. We consider a triangulated surface (cat skin) fully immersed in a Cartesian grid. The algorithm succeeded in building the conformal union (cf. fig.11 for a view of the clipped mesh and Table 1 for the performance details).

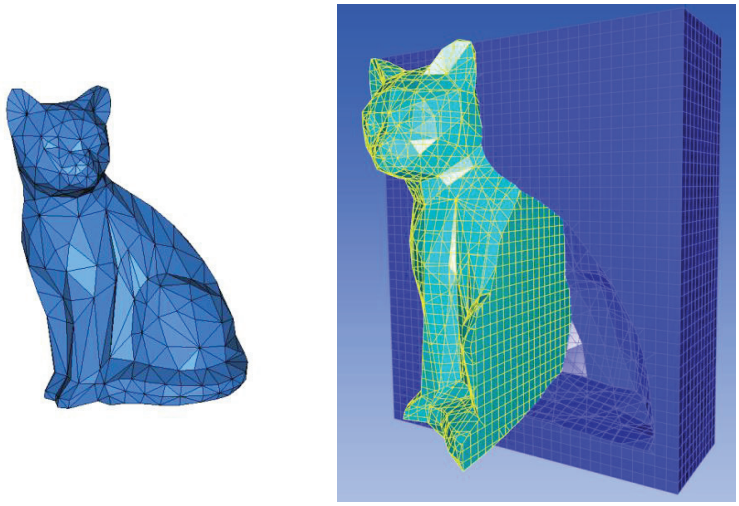

Fig. 11: (a) cat skin; (b) polyhedral and conformal mesh of the union with a Cartesian grid.

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 (as illustrated on the left of fig. 12) in an intersection position.

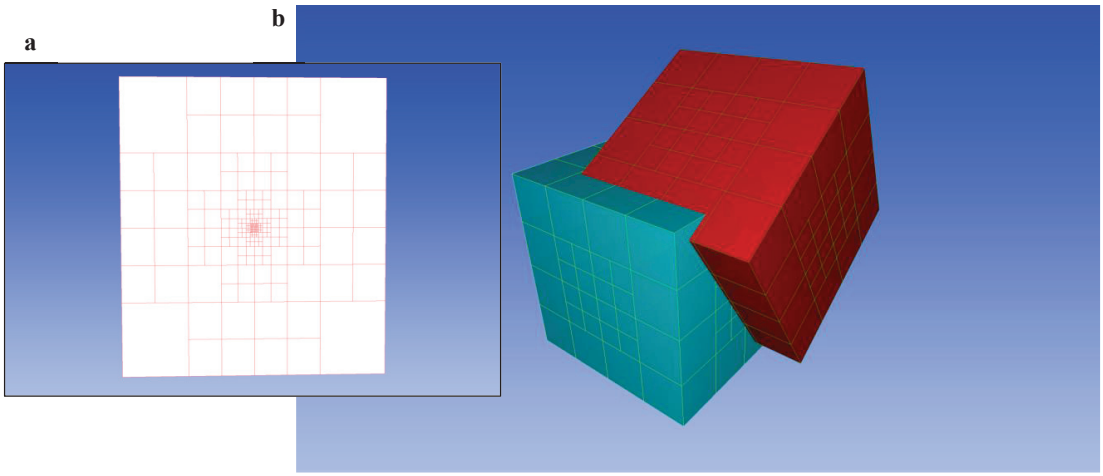

Fig. 12: (a) octree operand's inner structure; (b) Two intersecting octrees M_1 (blue) and M_2 (red)

Noting S_1 and S_2 the outer polygonal contours of respectively M_1 and M_2, figure 13 illustrates some mixed-type operands operations.

Fig. 13: (a) $M_1 \cap S_2$; (b) $M_1 \backslash S_2$

Figure 14 illustrates the 3D conservative remapping capability from the source mesh M_2 to the target mesh M_1 by showing the new mesh M_1' of the domain occupied by M_1. The boolean operation writes: $M_1' = (M_1 \backslash M_2) U (M_1 \cap M_2)$. The break down for a given M_1 cell is given on figure 14.b.

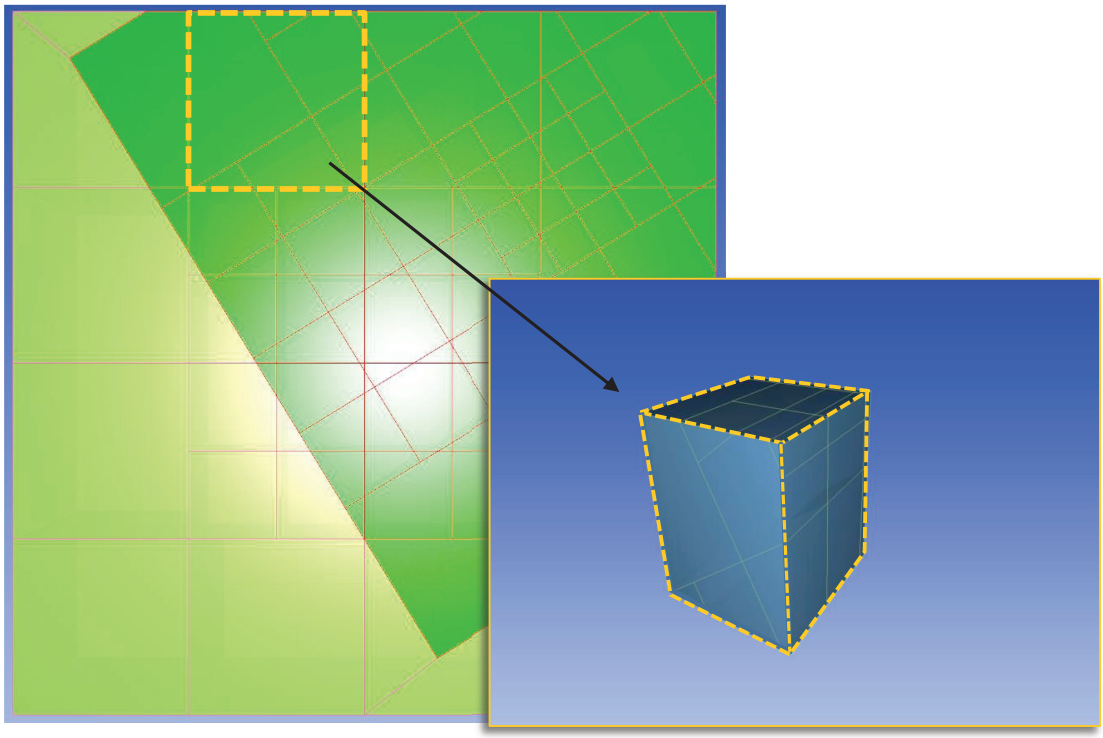

Fig. 14: (a) M'_1; (b) one M_1 cell's break down

Table 1. CPU times for the cut-cell meshing of the cat skin (758 triangles) immersed in a Cartesian grid (29,440 cells)

Step	CPU time (*s*)	Step	CPU time (*s*)
1) Initialization (92,000 polygons)	3.57	5) Polygons aggregation (110,972 triangles)	1.55
2) Triangulation (25,460 polygons)	0.09	6) Classification (32,866 cells)	0.14
3) Conformization (50,182 triangles)	41.13	**Total**	**51.07**
4) Cell construct (10,589 cells)	4.59	(performances measured on a single Intel Xeon core @ 2.8 GHz)	

6. Conclusion and perspectives

An algorithm with a O(nlog(n)) algorithmic complexity has been described to resolve the boolean operations for arbitrary polyhedral meshes and has been implemented successfully. The preliminary tests have shown that the current implementation performs well on simple configurations but it has not yet a sufficient robustness and performance efficiency to handle real industrial cases especially for moving bodies' applications where the whole boolean process needs to be computed at each time iteration.

Robustness issues that are mostly due to an improper degeneracies handling will be addressed in priority. Previous works on adaptive floating-point geometric predicates [18] and arithmetic filters [19] will certainly be considered to get a robust computation of intersections keeping in mind that this is the most time consuming step (approximatively 80% of the overall CPU time in the current implementation) so the retained solution will have to be very efficient.

Performances improvement will then be the main effort and the current sequential version will be converted into a multi-threaded one. The computation of intersections with this algorithm is precisely well fitted for multi-threading because the traces can be computed and stored concurrently and the splitting/swapping of each triangle doesn't engender any interdependence. Improving the localization of intersections by using a technique as described in [12] or [2] could bring a potential benefit as well. Another potential improvement would be to consider the robust approach proposed by Devillers and Guigue [16] to implement a Triangle-Triangle intersection predicate. This algorithm will need to be adapted to handle a tolerance and returns the intersections traces.

After the overall performance and improvement steps achieved, some design tuning and extensions will be done to handle efficiently the conservative remapping applications.

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 à un écoulement instationnaire par une méthode de chevauchement de maillages, AGARD FDP Symposium on « Progress and Challenges in CFD Methods and Algorithms » (1995).
[8] D. Badouel, G. Hégron, Opérations booléennes sur polyèdres : évaluation d'arbres CGS. [Research Report] RR-0839 (1988).
[9] M.O. Benouamer. Opérations booléennes sur les polyèdres représentés 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 & 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 & Computational Geometry, Vol. 18, pp. 303-363 (1997).
[19] P. Guigue, Geometric constructions with fixed precision, Computer Science, Université Nice Sophia Antipolis, <tel-00471447> (2003).