Computers and Mathematics with Applications 70 (2015) 1235–1254

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Exact, robust, and efficient regularized Booleans on general 3D meshes

Hichem Barki [a,*], Gaël Guennebaud [b], Sebti Foufou [a]

[a] CSE Department, College of Engineering, Qatar University, PO BOX 2713, Doha, Qatar

[b] Inria, Bordeaux University, France

ARTICLE INFO

Article history:
Received 8 September 2014
Received in revised form 5 May 2015
Accepted 20 June 2015
Available online 18 August 2015

Keywords:
Boolean operations
3D meshes
Computational geometry
Robust geometric computation
Solid modeling

ABSTRACT

Computing Boolean operations (Booleans) of 3D polyhedra/meshes is a basic and essential task in many domains, such as computational geometry, computer-aided design, and constructive solid geometry. Besides their utility and importance, Booleans are challenging to compute when dealing with meshes, because of topological changes, geometric degeneracies, etc. Most prior art techniques either suffer from robustness issues, deal with a restricted class of input/output meshes, or provide only approximate results. We overcome these limitations and present an exact and robust approach performing on general meshes, required to be only closed and orientable. Our method is based on a few geometric and topological predicates that allow to handle all input/output cases considered as degenerate in existing solutions, such as voids, non-manifold, disconnected, and unbounded meshes, and to robustly deal with special input configurations. Our experimentation showed that our more general approach is also more robust and more efficient than Maya's implementation (×3), CGAL's robust Nef polyhedra (×5), and recent plane-based approaches. Finally, we also present a complete benchmark intended to validate Boolean algorithms under relevant and challenging scenarios, and we successfully ascertain both our algorithm and implementation with it.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction and main contributions

The computation of Boolean operations for 3D meshes is an old problem which has been investigated several decades ago [1–4]. These operations constitute a very powerful tool for solid modeling by means of Constructive Solid Geometry (CSG) [5,6], and they are central to many application domains including CAD/CAM, simulations, and computer graphics. Nevertheless, their practical computation through a robust implementation is still challenging, especially when dealing with 3D boundary representation (B-rep) such as polygonal meshes. For any B-rep, the general approach to compute Booleans is conceptually simple: one has to compute all intersections between boundary primitives producing a new set of trimmed primitives, and then retain only parts belonging to the expected Boolean result. Although this direct approach has been employed frequently [7,8], it is subject to robustness issues in the form of *numerical and geometric degeneracies*.

Numerical degeneracies come from round-off errors inherent to floating-point arithmetic. They affect geometric predicates and constructions [9], leading to catastrophic failures in methods supposed to target exact Booleans.

Geometric degeneracies arise when Boolean algorithms rely on the so called *general position assumption* [10]. This implies that geometric configurations like (near) coplanar or tangentially touching facets, non-manifold geometries, unbounded

* Corresponding author.
E-mail addresses: hbarki@qu.edu.qa (H. Barki), gael.guennebaud@inria.fr (G. Guennebaud), soufou@qu.edu.qa (S. Foufou).

http://dx.doi.org/10.1016/j.camwa.2015.06.016
0898-1221/© 2015 Elsevier Ltd. All rights reserved.

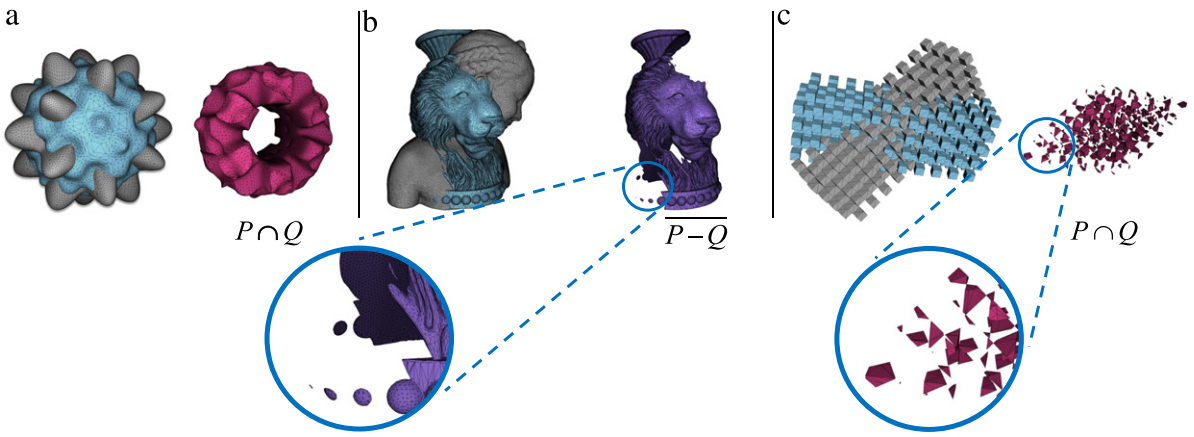

Fig. 1. Examples of challenging Boolean operations computed by our approach (operands depicted at the left). The result in (b) presents 8 unbounded components, with a genus 6, and (c) is a non-manifold intersection of non-manifold input models. Both (b) and (c) failed with Maya, CGAL's Nef [11], and plane-based technique [12].

objects, disconnected meshes, and arbitrary number of intersections are simply ignored and not handled by such approaches, thus yielding to failures in such cases. Considering all geometric degeneracies, i.e., enumerating and unambiguously handling all of them is not feasible for many approaches, because of the explosion of the number of degenerate cases to account for.

Added to numerical and geometric degeneracies, one has to consider topological issues. Most of the previous techniques restrict their input to the set of polyhedra (closed, manifold, and bounded meshes). However, even if Booleans are applied to polyhedra, the result is not necessarily manifold, nor connected, preventing the successive applications of Booleans. Moreover, many use cases involve unbounded meshes, which enclose an unbounded subset of the Euclidean space $\mathbb{E}^3$, as input and/or output, thus considerably limiting the application range of most existing approaches that appear to deal with polyhedra only.

Efficiency and memory consumption are also of main concern when targeting interactive applications, or real-world models.

Contributions. In this paper, we propose a novel approach addressing the main concerns of Booleans altogether: robustness, input/output topology generalization, efficiency, and exactness (Fig. 1).

To this end, instead of attacking this problem as a whole, our approach relies on a subtle sequential combination of different components, each being devoted to handle a special subset of configurations. We first apply a pre-processing dedicated to the handling of degenerate and non-planar polygonal facets. This step considerably simplifies subsequent treatments, and is essential for both **robustness** and efficiency purposes. Then, exact triangle intersections are computed and the result is efficiently encoded into 2D triangulated arrangements. Based on this lightweight representation, we show how to robustly and **efficiently** perform the primitive classification by exploiting the regularization, closedness, and orientability properties of our meshes, reducing by the way the number of intersection configurations to only two ones handled unambiguously. This classification relies on simple but exact predicates and avoids a case by case enumeration. Then our algorithm extends the result of this local classification by browsing both the union and intersection meshes at once in linear time. At this stage all intersecting components, and thus tricky configurations, have been classified. As a last resort, we show that disconnected or tangentially connected components can be reliably classified through **a robust** ray-shooting procedure. Our algorithm offers several benefits. **Input/output generalization** is accomplished by relying on simple but exact predicates and by considering the class of regular, closed and orientable meshes, which are able to better reflect the nature of physical solids, and which have a wider representation power than polyhedra and manifold surfaces restricting most prior art methods. **Special input configurations** (equivalent and complementary meshes) which are very problematic to prior art, are efficiently handled by our approach, thanks to two very fast predicates allowing to stop the algorithm in early stages. We emphasize that our algorithm directly works on the operands without any approximation, thus producing **exact results** in contrast to volumetric or other kinds of approximations. Finally, our approach handles Boolean computations on meshes that are geometrically closed, i.e., a viewer outside (resp. inside) the mesh cannot see the inside (resp. outside) of the mesh, but whose combinatorial structure is open. A typical example is a halfedge data structure containing border edges. In other words, our approach performs an **implicit mesh repair** on these combinatorially open meshes which are often encountered in practice, while computing Booleans.

In this paper, we also propose new guidelines to benchmark Boolean operations. Through an extensive and rich set of experiments, we consolidate our statements about exactness and robustness of our method, and demonstrate its generality in terms of correct handling of a rich set of input/output topologies (i.e., manifold/non-manifold, bounded/unbounded, connected/disconnected) that are not handled in most prior art. In particular, we compare our approach to Maya [13], CGAL's Nef [11], and recent plane-based techniques [10,12], and show its superiority in terms of robustness and efficiency.

2. Previous work

Several works have been devoted to the computation of Booleans on meshes. We distinguish volume-based approaches that perform the Booleans on an intermediate volumetric representation, and Boundary-based ones that directly deal with the input meshes.

2.1. Volume-based approaches

In order to avoid the aforementioned difficulties related to B-rep, several approaches first convert the operands into some volume representations through sampling, from which it is straightforward to compute the Booleans through distance fields [6] or level-sets methods [14]. These steps can be GPU accelerated using Layered Depth Images (LDI) for the discretization [15,16]. In both cases, the resulting volume is then converted back to a B-rep through marching cubes or some more advanced contouring techniques [17,18]. Some hybrid approaches aim to limit the effect of the discretization to the vicinity of mesh intersections [19,20]. However, as stated in [20], the surface–volume–surface conversion steps are still approximate and such approaches exhibit several limitations: non-preservation of sharp features and geometric details during sampling, manifoldness restriction, topological inconsistencies, stitching problems, etc.

2.2. Boundary-based approaches

Boundary representations include B-splines, Bézier surfaces, piece-wise linear surfaces (meshes), point sets, and many others. Here, we focus only on polygonal meshes, for which the most natural way to compute Booleans is to directly act on them as already described in the introduction. Unfortunately, a direct implementation of this algorithm is subject to numerical degeneracies, and many researchers thus focused on the pure numerical aspects of Boolean computations.

For instance, *epsilon tweaking* techniques combined to CSG and BSP trees [21,2,22] have been employed in some CAD/CAM packages. These methods do not give any guarantee about their success, and some tolerance parameters must be empirically adjusted on a case by case basis. Likewise, *symbolic perturbation* of the mesh coordinates [23,24] affects the correctness of Booleans while still being subject to numerical issues. *Interval arithmetic* techniques [25,26] track error bounds on the underlying floating-point arithmetic. Approaches based on them are quite efficient but non-robustness issues are still present: for instance, intervals can become arbitrarily large and lead to geometry collapse, or decisions cannot be taken without relying on other strategies.

Instead of focusing on numerical issues, alternative topology-oriented methods attempt to make guarantees on the correct connectivity of the produced Boolean results. Smith and Dodgson [27] achieved such topological robustness by relying on a series of interrelated operations, regardless of the used arithmetic. The produced geometric artifacts, such as gaps, slivers, zero-length edges, or zero-area facets, are smoothed-out in a post-processing step.

Hachenberger et al. [11] used *exact arithmetic* and proposed the first public exact and robust Boolean computation technique. They build upon the powerful Nef polyhedra structure [28] that is able to model non-manifoldness, unboundedness, and parts of different dimensions as well as to handle topological operations (boundary, interior, exterior, and closure) that change between open and closed half-spaces. On the other hand, this approach is not amenable to large size meshes, because of the huge computational resources both in space and time required by arbitrary precision arithmetic and the greedy Nef structure.

Sugihara and Iri [29] observed that one can advantageously perform Booleans on *plane-based representations* of meshes without requiring any geometric constructions (only predicates): the set of planes of the Boolean result is a subset of the sets of planes of the operand meshes. In practice, this means that the precision of the result can be bounded according to that of the operands. Moreover, robustness of predicates is easier to achieve than that of constructions, and several static filters have been proposed to ensure their correctness, even with fixed precision arithmetic [30,31]. Based on such planar representations, Bernstein and Fussell [10] proposed an exact method that has been recently improved by Campen and Kobbelt [12]. An octree subdivision is used to localize the intersection computations that are performed through nested BSP trees, and quantization is applied to guarantee correctness of the computations with fixed precision arithmetic. Even though in theory plane-based geometry does not require constructions, both methods actually employ geometric constructions. Moreover, the point–plane–point conversions that are necessary to handle meshes and the quantization are subject to robustness issues in case of non-planar, near degenerate, or near microscopic polygons.

Our algorithm follows the direct approach with exact arithmetic, and we show how to make it feasible and robust by adopting an adequate pre-processing and a novel classification strategy.

3. Definitions and preliminaries

Before describing our method in detail, we briefly introduce some definitions necessary to identify its scope and to justify our choices.

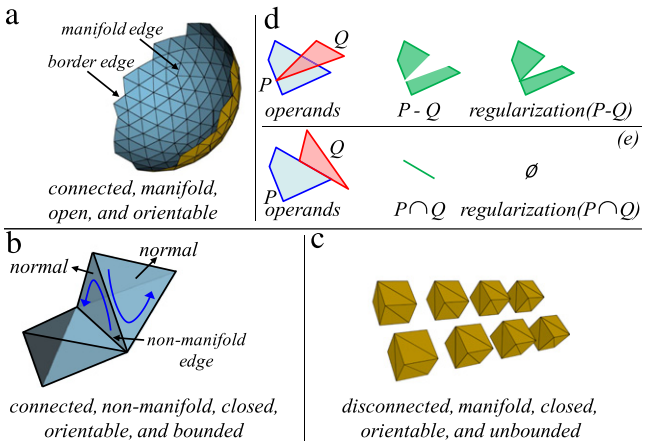

Fig. 2. (a)–(c) Illustration of mesh properties. Blue (front-facing) and orange (back-facing) indicate boundedness/unboundedness. (d)–(e) Regularized Booleans example in 2D. The resulting mesh in (d) is non-manifold even if the operands are manifold, and the result in (e) is the empty set. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Definition 1. A mesh is said: **connected** if the corresponding graph, whose nodes/arcs are facets/edges, is connected (Fig. 2(a), (b)); **manifold** if the neighborhood of each point on its surface is homeomorphic to an open disk (Fig. 2(a), (b)); **closed** if there is no border edge (Fig. 2(b), (c)); **orientable** if its facets are oriented in a consistent way such that the mesh unambiguously partitions space into interior/exterior sub-spaces (Fig. 2(b)); **bounded** if it encloses a bounded set in $\mathbb{E}^3$ (Fig. 2(b)).

In terms of combinatorial mesh data structures, the manifoldness property implies that the edges of a manifold mesh are either manifold (two incident facets) or border ones. The orientability property implies that each edge of an orientable mesh is traversed in opposite orders with respect to pairs of cyclically ordered facets incident to this edge (Fig. 2(b)). This holds for both manifold edges and non-manifold ones. The orientability property also implies that disjoint components of a mesh must be oriented consistently. By convention, and in addition to orientability, mesh facets are oriented so that the edges of each facet are ordered in a counterclockwise manner with respect to this facet, and the normal of each facet is oriented locally outwards that mesh. Finally, a valid mesh must be non-self-intersecting. This geometric property states that all the intersections among its faces (vertices, edges, and facets) must be encoded in its underlying combinatorial structure.

The orientability and closedness properties are essential because they avoid the ambiguity when identifying the region of space enclosed by a mesh (its interior) and the region outside of it (its exterior). An open and/or non-orientable mesh is not able to define such regions (e.g., Fig. 2(a)). In our work, we consider the general sets of closed and orientable meshes, which correspond to the surfaces of any physically realizable objects.

Definition 2. Two closed and orientable meshes are said: **equivalent** if they represent the boundary of the same subset of $\mathbb{E}^3$; **complementary** if one of them is equivalent to the complement of the other.

The complement of a mesh is obtained by flipping that mesh, i.e., by reversing the orientations of its facets. The equivalence and complementarity relations hold for any closed and orientable meshes, including unbounded, disconnected, and non-manifold ones. We emphasize that two triangulations of the same mesh having different facets, edges, and vertices, can still be equivalent if they enclose the same subset of space. A similar discussion applies to complementary meshes.

Definition 3. A **polyhedron** is a mesh that is connected, manifold, closed, orientable, and bounded.

It is clear that polyhedra represent only compact and bounded regions in $\mathbb{E}^3$ (e.g., model *Cow76kf* in Fig. 8). Unlike closed and orientable meshes, polyhedra cannot deal with non-manifold sets (Fig. 2(b)) or disconnected ones (Fig. 2(c)) that often occur in practice. Moreover, the set of polyhedra is not closed under Booleans: a Boolean applied to polyhedra does not necessarily give a polyhedron (Fig. 2(d)). This is a major issue shared by most previous work whenever one wants to apply successive Boolean operations, or simply deal with general meshes (Section 8).

Definition 4. A set is **regular** if it is equal to the closure of its interior [1]. The closure of a set consists of all its elements including boundary ones. The interior of a set consists of all its elements not belonging to its boundary.

Definition 5. A **regularized operation** is defined as the operation followed by a regularization of its result (Fig. 2(d), (e)).

Requicha [1] stated that regular sets are closed under regularized Booleans. This implies that closed and orientable meshes (which are regular by definition) are closed under regularized Booleans. This regularization is crucial since it

excludes lower dimensionality features (such as isolated edges and vertices) and enforces the resulting boundary parts to belong to the Boolean mesh.

In summary, our generalization of regularized Booleans to closed and orientable meshes is motivated by: (1) the unambiguous definition of the represented sets in $\mathbb{E}^3$, (2) the representation power of this mesh class compared to polyhedra, and (3) the closedness of the set of considered meshes under Boolean operations. From now on, mesh will designate a non-self-intersecting, regular, closed and orientable mesh, and Booleans will refer to regularized Booleans.

4. Our Boolean approach

Given two meshes P and Q, our method computes their union $U = P \cup Q$ and intersection $I = P \cap Q$ using the following three steps:

1. Input mesh pre-processing (Section 5).
2. Triangle intersections, special configurations handling, and unified representation (Section 6).
3. Triangle classification and union/intersection mesh browse (Section 7).

Our algorithm computes both union $U = P \cup Q$ and intersection $I = P \cap Q$ meshes in one pass. Any other Boolean result can be obtained by expressing it in terms of union/intersection and complement operations. As an example, the difference $D = P - Q$ is equivalent to $D = (P^c \cup Q)^c$, where $(.)^c$ denotes the complement.

As detailed in the following sections, our algorithm has been designed to involve only basic arithmetic operations, for which the field of rational numbers is closed. Therefore, numeric robustness of our algorithm is guaranteed by using rational exact arithmetic.

5. Input mesh pre-processing

The purpose of the mesh pre-processing step is to simplify further computations by triangulating the input meshes and eliminating degenerate facets.

By working exclusively on triangles, the triangulation step permits to simplify subsequent processing and to robustly handle the "non-planarity" issue related to polygonal facets and round-off errors. This issue causes serious robustness problems to approaches relying on plane-based geometry. In our approach, it is safe to simply ignore the degenerate triangles that may already exist, or be produced by the triangulation. The reasons are twofold: (1) since degenerate triangles have null area, removing them will not introduce holes in the meshes, i.e., they do not affect the geometric closedness of the considered meshes; and (2) since the triangle intersections will be recomputed by our algorithm (Section 6), we do not rely on the topology or connectivity of P and Q that may be affected by such a removal. Indeed, our algorithm handles geometrically closed operands, whose combinatorial structure (e.g., the underlying edge structure) is open, which is not the case of other approaches (Section 8).

6. Intersections and representation

The second main step of our algorithm consists in the five following sub-steps:

1. Computation of the intersections among all triangles coming from the pre-processing step.
2. Handling special input configurations, i.e., equivalent and complementary operands.
3. Subdivision of each triangle into sub-triangles, according to the geometric primitives resulting from its intersection with other triangles.
4. Triangulation of the subdivisions if necessary.

The outcome of these sub-steps is a unified representation of the two operands, which is ready for the fast and robust union/intersection classification algorithm of Section 7. All these sub-steps are detailed below.

6.1. Triangle–triangle intersections computation

In this sub-step, we compute all the intersections among triangles coming from the pre-processing step. Triangle–triangle intersections are computed using the robust "interval overlap method" [32], and accelerated by the "iso-oriented bounding boxes" algorithm [33]. This algorithm reduces the 3D intersection predicates into more efficient 1D interval intersection predicates. The computation of a triangle–triangle intersection is done only when the corresponding bounding boxes intersect. Thanks to the pre-processing step, all triangles are non-degenerate, so their supporting planes equations are also non-degenerate. Consequently, our triangle–triangle intersection computations are completely degeneracy-free. The computed intersections may be points, line segments, triangles, or convex polygons. All these configurations are explicitly accounted for during the upcoming subdivision and re-triangulation, i.e., all of them will be encoded in the computed 2D arrangements (Section 6.3).

6.2. Handling special configurations

Special configurations denote Boolean computations where the operands P and Q are either equivalent or complementary (Definition 2). They are considered special because the expected Boolean result is trivial to guess, but it is surprisingly difficult to compute for most existing implementations, especially due to numerical and geometric degeneracies. As the experiments of Section 8 show, even approaches relying on exact computations fail on such operands.

In order to detect special input configurations, we proceed as dictated by the following proposition and the deduced corollary.

Proposition 1. *Given two triangular meshes P and Q whose triangles are all non-degenerate, if every triangle of P overlaps at least one coplanar triangle of Q and vice versa, then P and Q are equivalent.*

We emphasize that throughout the paper, *coplanar* (resp. *opposite*) overlapping triangles denote a pair of triangles that both lie on the same supporting plane with the same (resp. *opposite*) orientation, and whose intersection is a polygon. In contrast, *non-overlapping* triangles refer to a pair of triangles that are neither coplanar nor opposite overlapping.

Proof. Let us prove Proposition 1 by contradiction. So equivalently, let us show that the opposite proposition being true leads to contradiction. The opposite proposition states that "P and Q are not equivalent **and** every triangle of P overlaps at least one coplanar triangle of Q and vice versa". Since P and Q are not equivalent, then there exists at least one triangle of one of them which does not entirely lie on the boundary of the second mesh, i.e., triangle t either lies completely outside the second mesh, lies completely inside it, crosses it (lies partly inside and partly outside the second mesh), or partly overlaps one or more triangles of the second mesh. The first and second cases imply that there exists a triangle t of one of the two meshes, which does not overlap any coplanar triangle of the other mesh, contradicting by the way the second statement of the opposite proposition. Therefore, for these two cases, Proposition 1 is true. Concerning the third and fourth cases, it is always possible to find a triangulation (even if not explicitly performed) of the crossing or partly overlapping triangle of one of the meshes, such that at least one generated sub-triangle falls into one of the two previous cases (either entirely outside or inside the second mesh), while the resulting mesh, say R is still equivalent to the re-triangulated mesh. In consequence, since mesh R and the second mesh are neither equivalent nor complementary, it follows that Proposition 1 is also true for the third and fourth cases. In conclusion, Proposition 1 is true.

Corollary 1. *Given two triangular meshes P and Q whose triangles are all non-degenerate, if every triangle of P overlaps at least one triangle of Q lying on an opposite plane, then P and Q are complementary.*

Corollary 1 is a special case of Proposition 1 where the latter is used to show the equivalence of P and Q^c.

Proposition 1 and Corollary 1 allow to efficiently identify special input configurations using only two predicates that, given a triangle of one mesh, determine the existence of at least one coplanar (resp. opposite) overlapping triangle of the second mesh. The operands P and Q are said equivalent (resp. complementary) if the first (resp. the second) predicate is satisfied for all triangles of both operands.

Our detection method is much more efficient than explicitly checking that the mesh triangles span the same or opposite areas, since the latter procedure requires costly geometric constructions. Moreover, its overhead is negligible because the two involved predicates are already partly evaluated during the previous triangle–triangle intersections sub-step, and it only remains to compare the (already available) relative plane orientations. Indeed, given a triangle of one mesh, our intersection algorithm easily informs us about the existence of at least one overlapping triangle coming from the second mesh, as this reduces to verifying that an already computed intersection of type polygon or triangle is detected for this triangle. Second, given that all triangles of both operands have been checked as overlapping other triangles, we only have to check whether two overlapping triangles have either coincident or opposite planes to conclude that P and Q are either equivalent or complementary.

This sub-step enables early termination of the algorithm thus preventing further useless and costly processing in such trivial cases: union and intersection meshes are then equivalent to either P or Q for equivalent operands, while for complementary operands the union is the whole space $\mathbb{E}^3$ and the intersection is the empty set $\emptyset$.

6.3. Triangles subdivision

Once all intersections have been computed and special configurations checked for, a main difficulty arises: how to subdivide the input triangles according to the computed intersections and how to encode the subdivided triangles? For this purpose, we propose the use of 2D arrangements [34,35] to both subdivide and encode input triangles. Each triangle is associated to a 2D arrangement constructed by projecting into a 2D space all the segments and points resulting from the intersection of this triangle with the others (Fig. 3). When the intersection result is a triangle or a polygon, its segments are also considered for the arrangement construction. This contrasts to some prior work that considered as degenerate such overlapping triangle configurations because they violate the *general position assumption*. In Section 7, we will also show that coplanar and opposite overlapping triangles (whose intersections are either triangles or polygons) are robustly handled by the classification and union/intersection browse step.

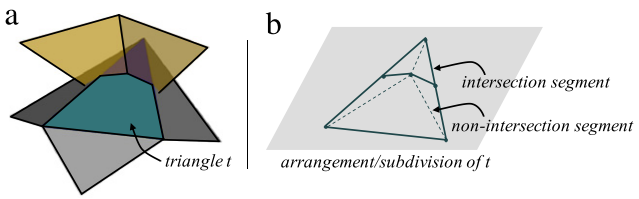

Fig. 3. Illustration of the subdivision process, with (a) a given triangle t intersecting other triangles in 3D, and (b) the corresponding 2D arrangement and its triangulation.

The passage between 3D triangles and the corresponding 2D arrangements must be handled through a bijective (degeneracy-free) and ideally fast projection. Given a triangle t with a normal $\mathbf{n} = (x_n, y_n, z_n)$, a projection that fulfills our requirements consists in ignoring, among x_n, y_n, and z_n, the coordinate having the largest absolute value (say y_n), and considering the plane spanning the values of the two others (say the z–x plane) for the embedding of the corresponding 2D arrangement. The inverse 2D to 3D mapping is deduced from the equation of the supporting plane of t.

The use of such 2D arrangements embedded in the x–y, y–z, or z–x planes exhibits several advantages. The reduction of the problem dimensionality from 3D to 2D yields more efficiency and easiness, compared to directly dealing with arrangements embedded in the triangle supporting planes. The associated bijective projection is straightforward and preserves the input numerical precision. Moreover, this eases the handling of the combinatorial structure of the intersecting 3D triangles (non-manifold topology, disconnected parts, overlapping triangles, etc.) that is prone to fail when handled with classical 3D mesh data structures.

6.4. Triangulation of the subdivisions

The result of the last sub-step consists in 2D arrangements encoding the subdivision of the mesh triangles. However, the faces of these 2D arrangements are most probably not triangular or convex. Added to that, depending on the intersection configuration, some arrangements may contain faces with holes inside them. Robustly dealing with all these geometries in subsequent steps is a big challenge. This is why in this sub-step, each arrangement is triangulated by considering its edges and vertices as the constraints of a constrained Delaunay triangulation. This permits to exclusively deal with triangles, which is much easier than dealing with all aforementioned geometries. Most importantly, using constrained Delaunay triangulations ensures the preservation of the input intersection segments and vertices in the resulting triangular 2D arrangements, and avoids the introduction of degenerate triangles as 2D arrangement faces. Therefore, the consistency of the next steps of our algorithm is guaranteed.

As explained in the next section, opposite overlapping triangles can be safely removed. Then an important property of each sub-triangle resulting from our arrangement triangulations is its unique membership state: each sub-triangle either belongs to the union mesh or to the intersection mesh, but not to both meshes. A proof of this statement can be easily conducted, for instance based on an equivalent result in [36]. The resulting sub-triangles will be denoted as triangles in the rest of the paper.

Even though we avoid the use of a global mesh data structure for encoding the subdivided mesh triangles, it is still important to keep track of all the necessary geometric and topological information linking our triangles within the constructed 2D arrangements. When constructing an arrangement a_i, for each of its edges, we keep track of all triangles that generated this edge by intersecting the corresponding triangle t_i. All the possible intersection configurations are accounted for. As explained in the next section, this edge–triangles incidence relation is used for the union/intersection classification.

7. Classification and union/intersection browse

The last main step of our algorithm is to classify the resulting triangles into union and intersection ones. As dictated by Algorithm 1, we first locally classify triangles around each intersection segment, then propagate this information through the mesh, and employ ray-shooting to resolve the remaining ambiguities if any. This process is done without making any general position assumptions, without any kind of perturbation, and with robust handling of overlapping triangles.

7.1. Radial sort of triangles

The radial sort is done by considering an arbitrary Cartesian coordinate system u–v–w whose w-axis corresponds to the segment s (Fig. 4(a)). The opposite triangle vertices, i.e., triangle vertices not shared with segment s are then orthogonally projected onto the u–v plane, and then sorted by increasing radial angles (Fig. 4(b)). Note that the triangles are sorted cyclically and the u and v axes are thus arbitrarily chosen. To avoid transcendental trigonometric functions and thus guarantee the robustness of the sort in the context of rational exact arithmetic, the signs of the projected vertices onto the frame u–v are used to classify them into one of its four quadrants. Within each quadrant, pairs of vectors are compared through the sign of their 2D cross product.

Algorithm 1: Union/Intersection classification

Input: Triangle intersections and their arrangements
Output: The union U and intersection I meshes

```
for each intersection segment s do
    Sort the triangles incident to s radially and remove the opposite overlapping ones;
    for each triangle t_i in the sorted list around s do
        if t_i-t_{i+1} define union/intersection (∪/∩) config. then
            Classify all triangles around s into U or I;
            break;
        else
            Tag t_i as undefined;
Browse the union and intersection meshes starting from the known ∪/∩ configurations;
while there exist non-classified triangles do
    Perform a ray-shooting to classify them;
```

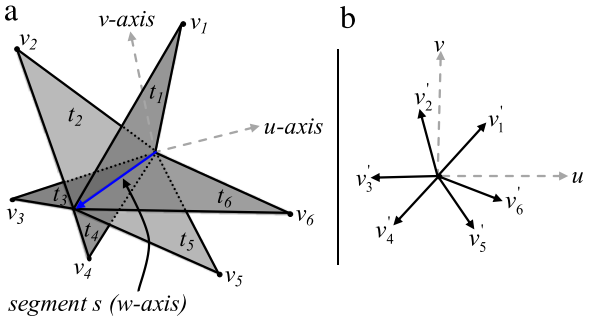

Fig. 4. Principle of the radial sort of triangles sharing a segment s. The vectors v'_i in (b) that correspond to the orthogonal projections of the vertices v_i in (a) on the frame u–v determine the radial (ascending) order $\ldots, t_1, t_2, t_3, t_4, t_5, t_6, t_1, t_2, \ldots$..

Overlapping triangles lying on the same plane or on two opposite planes (Fig. 5(a)) are sorted at the same location. While the first configuration will require a special care in the upcoming steps, the second one of two overlapping and opposite triangles is handled by safely discarding them. Indeed, such triangles cannot belong to U or I, because we are dealing with a B-rep (i.e., meshes) and we are considering regularized Boolean operations. In other words, when considering union, such triangles are surrounded by the volumes enclosed by P and Q (Fig. 5(a)) and thus do not belong to the B-rep of the union mesh. When dealing with intersection, the regularization operation removes them (Fig. 2(e)).

7.2. Local classification of the sorted triangles

Given a segment s, let $t_1, \ldots, t_{m_s}$ be the m_s triangles incident to s and radially sorted around it. The goal of this stage is to classify each triangle as part of the union or intersection solely based on orientability and relative radial positions.

7.2.1. Prerequisites

In order to perform a correct classification, let us first introduce two propositions that will allow us to considerably reduce the number of possible configurations that have to be considered to only two ones. Our propositions are based on the following set theory result and 3D mesh-related convention:

1. **Existence of meets and joins**. For every two sets $P, Q \subset \mathbb{E}^3$: $(P \cap Q) \subseteq (P \cup Q)$. In particular, $(P \cap Q) = (P \cup Q) \Leftrightarrow P = Q$.
2. **Mesh normal convention**. The normal of any facet of a mesh is oriented locally outwards that mesh.

Proposition 2. *Consider a segment s and the m_s incident triangles $t_1, \ldots, t_{m_s}$ radially sorted around it. If there exist two successive non-overlapping triangles t_i and t_{i+1} defining a non-orientable surface, then:*

1. *Either t_i belongs to the union mesh U and t_{i+1} belongs to the intersection mesh I, or vice versa.*
2. *The triangle whose normal points outwards the sub-space defined by the two triangles is the union one, while the other is the intersection one.*

Proof. The proof of the first part which states that t_i and t_{i+1} do not both belong to the same Boolean result is straightforward, because if t_i and t_{i+1} both belong to U (resp. I), then this contradicts the orientability of U (resp. I).

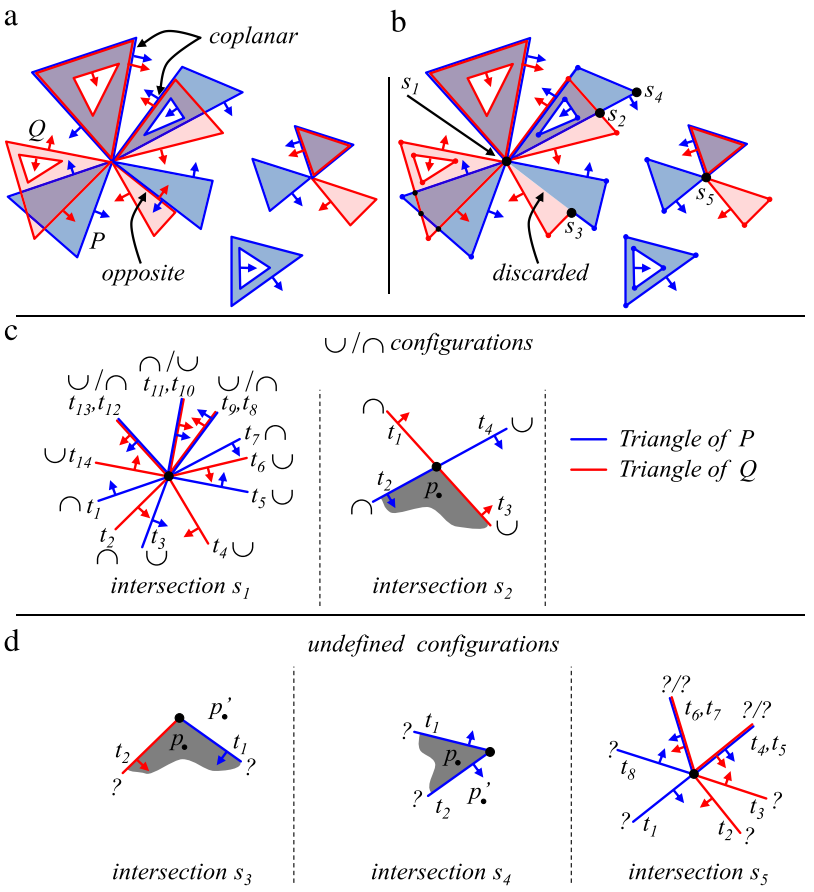

Fig. 5. A 2D illustration of the classification process. In this representation, 3D triangles correspond to 2D segments, and 3D segments to 2D points. (a) Two arbitrary meshes P (blue) and Q (red) with voids, disconnected and non-manifold configurations. (b) The triangles obtained after the pre-processing and triangle subdivision steps. Black disks correspond to 3D intersection segments. (c) $\cup/\cap$ configurations corresponding to segments s_1 and s_2. (d) Undefined configurations corresponding to segments s_3, s_4, and s_5. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For the second part, consider locally around segment s, the open subspace defined by t_i and t_{i+1} (gray-shaded in Fig. 5(c) middle) and a point p arbitrarily close to segment s and lying within this subspace. Remember that t_i and t_{i+1} define a non-orientable surface, so one of them belongs to U while the other belongs to I (first statement of our proposition). To fix ideas, let us suppose that $t_i = t_2$ while $t_{i+1} = t_3$ in Fig. 5(c) middle. The classification of point p with respect to union U and intersection I implies four exhaustive cases:

1. p lies inside both U and I.
2. p lies outside both U and I.
3. p lies outside U and inside I.
4. p lies inside U and outside I.

Consider the first case. If we suppose that $t_i = t_2$ belongs to I while $t_{i+1} = t_3$ belongs to U, then this contradicts case (1) statement that p lies inside both U and I, because the mesh normal convention implies that $p \notin I$. Now, if we suppose that $t_i = t_2$ belongs to U while $t_{i+1} = t_3$ belongs to I, then the mesh normal convention implies that $(P \cap Q) \not\subseteq (P \cup Q)$ because p is inside I ($p \in P \cap Q$) and outside U ($p \notin P \cup Q$), meaning that this configuration is also contradictory because it violates the existence of meets and joins. Therefore case (1) is impossible. The impossibility of case (2) can also be proven analogously to case (1). Case (3) is also impossible because it implies that p is inside I ($p \in P \cap Q$) and outside U ($p \notin P \cup Q$), meaning that it violates the existence of meets and joins. The fourth case is the only valid one because it verifies the existence of meets and joins. Since p lies inside U and outside I (case (4)), and according to the mesh normal convention, it follows that among t_i and t_{i+1}, the triangle that belongs to union U is the one whose normal is oriented outwards the subspace defined by the two triangles (t_3 in Fig. 5(c) middle), the other one being part of the intersection (t_2 in Fig. 5(c) middle). For completeness, a similar proof can be conducted for all cases when $t_i = t_3$ while $t_{i+1} = t_2$ in Fig. 5(c) middle.

Proposition 3. *Consider a segment s and the m_s incident triangles $t_1, \ldots, t_{m_s}$ radially sorted around it. If there exist two successive triangles t_i and t_{i+1} defining an orientable surface, then both t_i and t_{i+1} belong to either union U or intersection I.*

Proof. Let us proceed by contradiction, i.e., let us suppose that t_i and t_{i+1} do not belong to the same Boolean result. We arbitrarily assume that t_i belongs to I while t_{i+1} belongs to U. To fix ideas, we consider Fig. 5(d) middle and suppose that $t_i = t_1$ while $t_{i+1} = t_2$. This configuration implies that the normals of t_1 and t_2 are oriented outwards the open subspace defined by these triangles. From our assumptions, it follows that p lies inside both I and U. If we move the point p radially around the segment s so that it crosses the union triangle t_2 to produce a point p' as in Fig. 5(d) middle, then the membership of p' with respect to the intersection mesh I is the same as that of point p because the boundary of the intersection mesh I was not crossed. In contrast, crossing the boundary of the union mesh U implies that p' lies outside U ($p' \notin P \cup Q$ because $p \in P \cup Q$), which contradicts the existence of meets and joins since p' lies inside I ($p' \in P \cap Q$). The symmetric case assuming that t_i belongs to U while t_{i+1} belongs to I can be proved similarly. Therefore, both t_i and t_{i+1} must belong to the same Boolean result. For completeness, the only remaining case where the normals of t_i and t_{i+1} are oriented towards the defined open subspace (Fig. 5(d) left) can be analogously proved, by moving p so that it crosses the triangle belonging to the intersection mesh I.

Remark that t_i and t_{i+1} in Proposition 3 cannot be opposite overlapping because such triangles are discarded by regularization during radial sort.

7.2.2. Local classification algorithm

Based on the two aforementioned propositions, for each segment s, the classification of its incident triangles is performed by analyzing each pair of non-overlapping triangles t_i and t_{i+1} incident to it. Triangles will be classified as being part of either the union ($\cup$-tagged) or intersection ($\cap$-tagged). However, as suggested in the general overview of the method, for particular segments, not all incident triangles can be classified using such purely local analysis and the remaining unclassified triangles will thus be ?-tagged. This local classification procedure is briefly presented below and further detailed and justified in the next sub-section.

Given the cyclic sequence of m_s triangles $t_1, \ldots, t_{m_s}$ incident to segment s and radially sorted around it, our algorithm starts by searching the first pair of non-overlapping successive triangles t_i and t_{i+1} defining a non-orientable surface. Two situations might happen:

If such a pair cannot be found, then only coplanar overlapping triangles, if any, can be classified. For each overlapping pair, one triangle is arbitrarily $\cup$-tagged, while the other is $\cap$-tagged. All other triangles remain ?-tagged. We refer to this situation as the *undefined configuration*.

Otherwise, all incident triangles will be properly classified, and we refer to this situation as the $\cup/\cap$ *configuration*. Given the non-overlapping and non-orientable pair t_i, t_{i+1}, the triangle whose normal is oriented outwards the sub-space defined by these two triangles is $\cup$-tagged, while the other is $\cap$-tagged (Proposition 2). Then, the next triangle t_{i+2} in the cyclic sequence is classified by considering the pair of triangles t_{i+1} and t_{i+2}. If this pair defines an orientable surface, then the tag of t_{i+1} is copied to t_{i+2} (in this case t_{i+1} and t_{i+2} are non-overlapping). Otherwise, t_{i+2} receives the opposite tag of t_{i+1} (in this case t_{i+1} and t_{i+2} might overlap each other). This classification step is repeated by considering next triangle t_{i+3} until processing triangle t_{i-1} in the cyclic sequence, i.e., until all triangles incident to segment s are classified.

7.2.3. Local classification details

The aforementioned steps are further explained and illustrated in the next paragraphs, which correspond to the only two possible configurations for any segment s: the $\cup/\cap$ *configuration* and the *undefined configuration*.

THE $\cup/\cap$ CONFIGURATION

For a particular segment s, if and only if a pair of non-overlapping successive triangles t_i and t_{i+1} incident to it and defining a non-orientable surface is detected (e.g., t_2 and t_3 in Fig. 5(c) left and middle), then the segment s is said to exhibit a $\cup/\cap$ configuration. According to Proposition 2, t_i and t_{i+1} do not both belong to the same Boolean because this Boolean (either union or intersection) is orientable. So among t_i and t_{i+1}, the triangle whose normal is oriented outwards the sub-space defined by the two triangles is classified as a union triangle by being $\cup$-tagged (t_3 in Fig. 5(c)), while the other belongs to the intersection and is $\cap$-tagged (t_2 in Fig. 5(c)).

Once a $\cup/\cap$ configuration has been determined for a segment s by finding a first pair of non-overlapping successive triangles t_i and t_{i+1} defining a non-orientable surface, all other triangles surrounding the segment s can be unambiguously classified as well. Without loss of generality, suppose t_{i+1} is $\cup$-tagged. If the next triangle t_{i+2} defines an orientable surface with t_{i+1}, then according to Proposition 3, it necessarily belongs to union U and will be $\cup$-tagged (e.g., t_4, t_5, and t_6 in Fig. 5(c) left and t_4 in Fig. 5(c) middle). Otherwise, t_{i+2} necessarily belongs to I and will be $\cap$-tagged (e.g., t_7 and t_1 in Fig. 5(c) left and t_1 in Fig. 5(c) middle). An analogous discussion applies when t_{i+1} is $\cap$-tagged.

Notice that if two coplanar overlapping triangles around s are detected (e.g., t_8 and t_9 in Fig. 5(c) left), then one of them will be tagged similarly to the triangle preceding it because of orientability (Proposition 3, e.g., t_8 is $\cap$-tagged as t_7 in Fig. 5(c) left). The other one will be tagged differently with respect to union and intersection (e.g., t_9 is $\cup$-tagged in Fig. 5(c) left) because of non-orientability of the coplanar overlapping triangles (Proposition 2). Even if the classification of coplanar overlapping triangles seems arbitrary, it is robust and has no effect on the propagation process, because both union and intersection

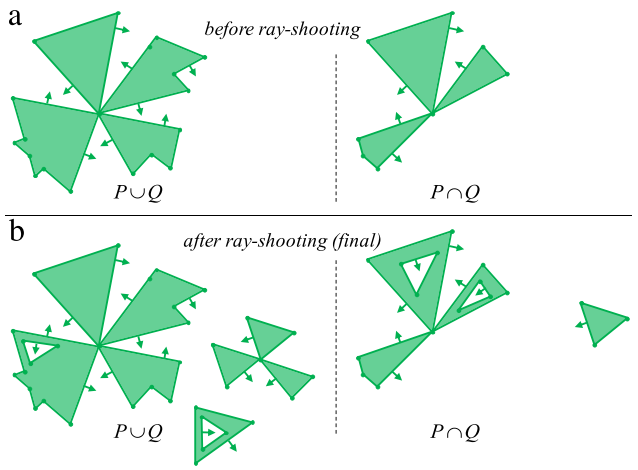

Fig. 6. Ray-shooting principle applied to more complex meshes. (a) temporary union and intersection meshes obtained after radial sort, classification, and browse sub-steps but before ray-shooting sub-step. Consider Fig. 5(a) for the operand meshes P and Q. (b) Final union and intersection meshes after performing ray-shooting.

meshes share the same coplanar overlapping parts, so it does not matter which triangle is classified to union and which one is classified to intersection.

This classification is repeated until all the triangles incident to segment s are appropriately tagged.

THE UNDEFINED CONFIGURATION

This configuration happens when all pairs of successive triangles t_i and t_{i+1} either define an orientable surface (Fig. 5(d) left and middle) or are coplanar overlapping (e.g., t_4 and t_5 in Fig. 5(d) right). Indeed, if all pairs t_i and t_{i+1} are orientable, then the whole local surface around s is orientable, and according to Proposition 3, it follows that all triangles around s belong to the same Boolean result, but we cannot decide more about them. The segments resulting from the triangulation of arrangements fall into this case because they are incident to exactly two triangles defining an orientable surface. Even though the presence of coplanar overlapping triangles incident to s introduces non-orientable pairs, for which one triangle is $\cup$-tagged while the other one is $\cap$-tagged (both union and intersection meshes share the same coplanar parts), they do not help here because we cannot decide about the classification of the remaining triangles incident to segment s by using Proposition 2. At this point, the remaining triangles incident to segment s are just ?-tagged. Their respective memberships will be retrieved during the mesh browse or, at a last resort, during ray-shooting as detailed in the next subsections.

It is worth noting that the number of intersection configurations, whose enumeration and correct handling were considered unfeasible has been drastically reduced to only two which are unambiguously handled. It also becomes clear why the case of equivalent operands must be handled during the preceding step (Section 6). Indeed, in such a case, all segments s would be surrounded exclusively by pairs of coplanar overlapping triangles and no $\cup/\cap$ configuration could be detected.

7.3. Union/Intersection mesh browse

This step permits to classify ?-tagged triangles that belong to a connected component for which at least one triangle has been classified. Given a not yet visited $\cup$-tagged triangle, all neighboring ?-tagged triangles that do not violate the orientability property are $\cup$-tagged and inserted into the union mesh U. This procedure is repeated until all $\cup$-tagged triangles have been visited and their neighborhood classified. A similar processing is done for each $\cap$-tagged triangle to form I. This browsing process is linear in terms of the triangles count, and an illustration of its outcome when applied to the meshes of Fig. 5(a) is given in Fig. 6(a).

7.4. Ray-shooting of non-classified triangles

The browse procedure does not permit to classify the components for which no $\cup/\cap$ configuration has been found. This is the case of disjoint components, or components that only touch tangentially. We propose to solve these last ambiguities through a ray-shooting approach which is illustrated in Fig. 7 and detailed as follows. As before, P and Q refer to the complete operand meshes.

1. Pick an arbitrary ?-tagged triangle t. By construction, all ?-tagged triangles (and thus t) do not overlap any other coplanar triangles of the second mesh (cf. Section 7.2). To ease the explanation, let us suppose that t belongs to the mesh P (Fig. 7(a)).

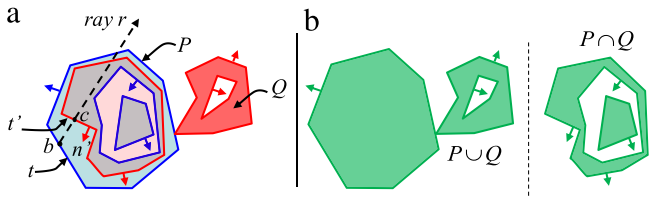

Fig. 7. Ray-shooting principle. (a) No ∪/∩ configuration is detected for two meshes P and Q that are touching tangentially and composed of disjoint surfaces. (b) Final union and intersection meshes after performing ray-shooting.

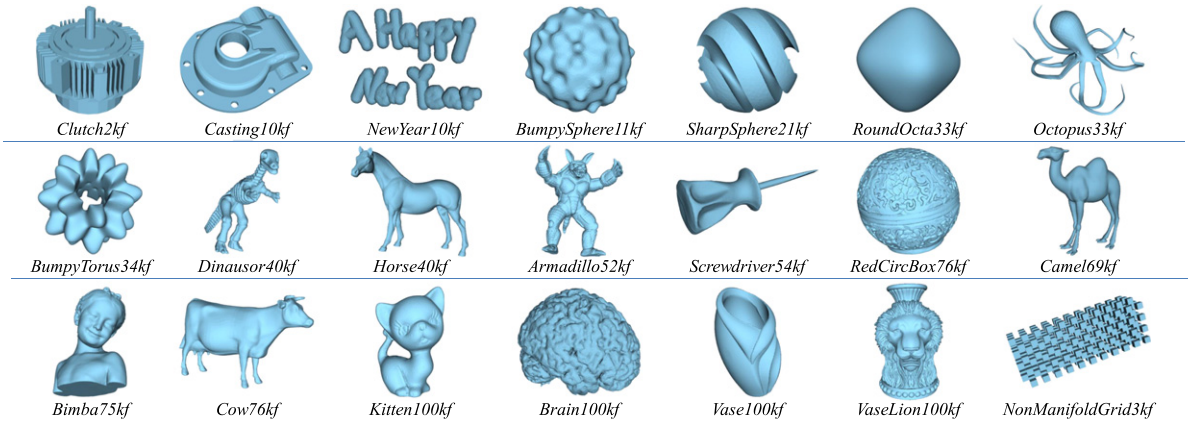

Fig. 8. Benchmark models. The name postfix 'nkf' indicates the number of kilo triangles. All models actually represent polyhedra except NewYear10kf (disconnected) and NonManifoldGrid3kf (non-manifold and disconnected). Models are provided courtesy of the AIM@SHAPE Shape Repository.

2. Shoot a random ray starting from an arbitrary point **b** of t towards Q triangles, and find the closest intersection point **c**, say on triangle t'. The random ray is chosen to be non-parallel to both supporting planes of t and t' (Fig. 7(a)).
3. If **b** is on the negative side of the plane supporting t', then the triangle t lies inside the region enclosed by Q, and thus it does not belong to the union $P \cup Q$. Moreover, since t is part of the boundary of P and is inside Q, it follows that it is part of the boundary of the intersection $P \cap Q$.
4. Otherwise, **b** is on the positive side of t', and t' is inside the region enclosed by P: t belongs to the union $P \cup Q$ (Fig. 7(a)).
5. Browse the corresponding connected surface, starting from t and classify all reachable triangles accordingly (Fig. 7(b)).

This process is repeated until all triangles are appropriately classified.

In the literature, ray-shooting has been recognized as an unstable process due to numerical issues arising for some geometric configurations, like nearly coplanar mesh triangles. However we argue that, in our context, the employed ray-shooting technique is both robust and degeneracy free for the following reasons. Firstly, by construction the ray shot from triangle t is ensured not to miss Q triangles because we shoot towards Q triangles bounding boxes, until hitting a triangle t'. Secondly, numerical issues are avoided by choosing non-overlapping and so non-coplanar triangles t and t'. The existence of a ray non-parallel to both supporting planes of t and t' is guaranteed by the existence of non-overlapping triangles, i.e., triangles not lying on the same plane or on two opposite planes. These non-overlapping triangles do exist at this stage because all overlapping triangles have already been classified in the previous step, or in the extreme case of equivalent or complementary operands, these triangles have already been classified earlier in the algorithm (see *special configurations handling* in Section 6) so local classification and ray-shooting would not be performed. In the same vein, **b** is guaranteed not to belong to the supporting plane of t' because the existence of non-overlapping triangles t and t' is ensured.

Since we do not rely on any visibility criteria to determine union and intersection during local classification, our approach deals with meshes regardless of their topology ((un)boundedness, (dis)connectedness, (non-)manifoldness). This also holds for ray-shooting whose result is guaranteed for closed and orientable meshes. We also emphasize that this ray-shooting process does not require that some triangles have already been classified, thus allowing to handle completely disjoint meshes. Fig. 6 gives an illustration of the ray-shooting outcome when performed on the complex shapes of Fig. 5(a). Experimental results ascertaining our approach under all mentioned configurations can be found in the next section.

8. Implementation and results

We implemented our algorithm with the help of the CGAL library [32]. In particular, we used the `Arrangement_2` and `Constrained_Delaunay_triangulation_2` data structures for 2D Arrangements construction and their triangulation (Section 6). In order to handle non-manifold, disconnected, and unbounded input/output meshes, we employed our own

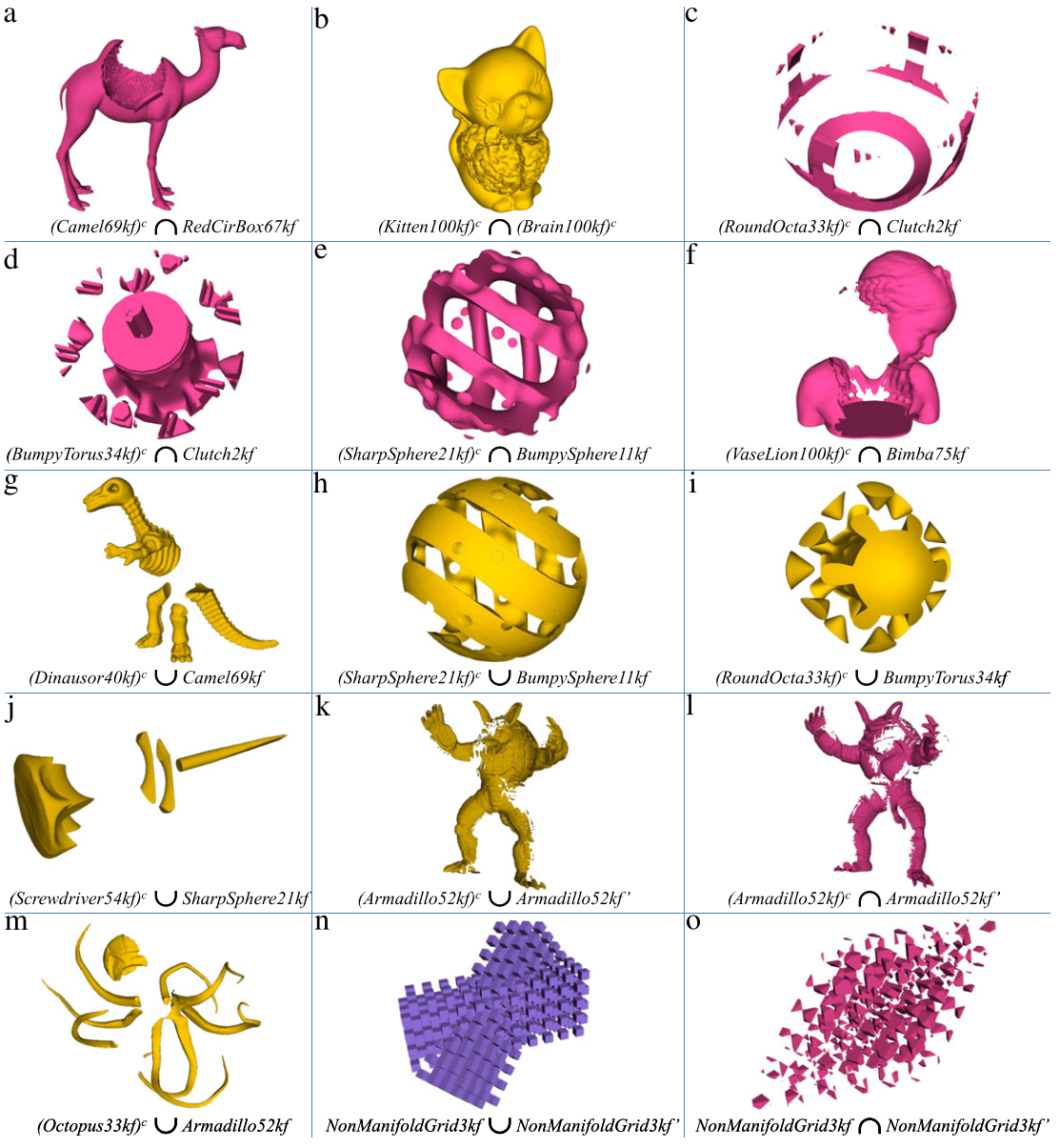

Fig. 9. Some results of our algorithm applied to closed and orientable meshes depicted in Fig. 8. Mixtures of purple (front-facing) and orange (back-facing) indicate boundedness/unboundedness. Note the rich topology (unboundedness, disconnectedness, non-manifoldness) of the handled input/output meshes. Models Armadillo52kf' and NonManifoldGrid3kf' are rotated version of the original ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

general simplicial-complexes data structure, ensuring a correct handling of the rich topology of closed and orientable meshes, and avoiding verification issues related to polygon soups. We guaranteed numerical robustness by using exact rational number types [37] filtered by the lazy kernel adapter of Fabri and Pion [38]. In order to gain additional performance, we employed the OpenMP parallel loop construct for the parallelization of the sub-steps of our algorithm that involve independent iterations. These sub-steps include triangle–triangle intersections (parallel loop on the pairs of triangles whose bounding boxes intersect), 2D arrangement constructions and triangulations (parallel loop on the list of 2D arrangements), and per-segment radial sorts and local classifications (parallel loop on the segments). Finally, since the browses of union and intersection meshes are two independent processes, we also parallelized them by relying on the OpenMP parallel sections construct.

Our results have been obtained on a 3.4 GHz Intel Core i7 computer equipped with 16 GB of memory. Fig. 8 depicts a small selection of the models we used in our evaluations. They come from different sources (scan, CAD) and vary in their topology. Some Boolean results obtained with our approach are given in Fig. 9. A much more extensive set of sample models and results is provided in the supplemental materials (see Appendix A).

Table 1
Detailed running times for our approach and comparison with prior art.

Operands		Steps time (%)			Result (# kilo triangles)		Running times (s)			
P	*Q*	Prep.	Tri. Sp. Arr.	Cls. Br.	Union	Inters.	Ours	Maya	Nef	PGBT
Casting10kf	Clutch2kf	17.48	56.51	26.01	12	6	**3.14**	1.44	9.49	YES
BumpySphere11kf	BumpyTorus34kf	4.58	62.74	32.68	32	31	**10.28**	17.04	52.54	YES
RoundOcta33kf	SharpSphere21kf	5.48	63.42	31.10	34	36	**10.96**	24.41	61.80	YES
Bimba75kf	NewYear10kf	6.09	59.19	34.72	79	14	**16.21**	NO	NO	YES
Armadillo52kf	Dinausor40kf	6.56	57.18	36.26	74	25	**16.59**	55.49	95.28	YES
Horse40kf	Cow76kf	9.24	51.02	39.74	96	24	**20.96**	NO	120.08	NO*
Camel69kf	Armadillo52kf	4.95	40.12	54.93	111	16	**29.22**	58.11	121.97	NO**
Bimba75kf	Vase100kf	7.35	57.31	35.34	139	54	**32.01**	NO	NO	NO**
Kitten100kf	RedCircBox67kf	6.11	49.96	43.93	119	58	**33.80**	192.22	NO	NO**
Kitten100kf	Brain100kf	5.79	52.49	41.72	160	81	**45.68**	NO	NO	NO**
Average timings and speed-up		**7.36**	**55.00**	**37.64**	**86**	**35**	**21.89**	**×3.35**	**×5.06**	–

Prep.—Pre-processing, Tri. Sp. Arr.—Triangle intersections, Special configurations handling, Arrangement computations, and segment–triangles relation reconstruction, Cls. Br.—local Classification and mesh Browse, YES—successful union/intersection computations by PGBT (no running time provided), NO—failure of both union and intersection computations with NO* for "boundaries or degenerate elements", and NO** for "memory or processing time limits, inconsistent, complex, or self-intersecting input" as reported by Campen and Kobbelt [39].

In the following, we evaluate and compare our algorithm by considering efficiency, robustness, exactness, and input/output generalization. We conducted our comparisons to Autodesk Maya [13] (that implements a non-robust technique using floating-point arithmetic), the Nef polyhedra-based approach of CGAL [11] (exact method using exact arithmetic), and the online implementation [39] of the plane geometry-based technique (PGBT) of Campen and Kobbelt [12] (exact approach using fixed precision arithmetic). Except PGBT (whose implementation is not available), Maya and Nef are implemented on the same hardware as our algorithm, and the provided timings and comparisons have been performed under the same environment (same hardware, same OS, etc.).

8.1. Performance evaluation

The detailed running times for our algorithm are reported in Table 1. The overall timings correspond to the computation of both union and intersection in a single pass. Not surprisingly, the most time consuming step is the construction of the compatible representation, which comprises the computation of triangle intersections and 2D arrangements. On the other hand, the pre-processing step represents only a small fraction.

Table 1 also includes comparisons to Maya, Nef and PGBT. If we put aside their numerous failure cases that will be discussed in Section 8.3, we observe that, even though our approach relies on greedy exact arithmetic, it is more than 3 times faster than the non-robust Maya method based on floating-point arithmetic, and about 5 times faster than Nef. Since the online implementation of PGBT does not report running times, we can only guess that our approach is as efficient as PGBT since it has been reported to be 2.5–13 times faster than Nef [12]. The technique of Bernstein and Fussell [10] that inspired PGBT was reported to be twice slower than Maya, and so it is much slower than our approach. We believe even higher performance could be achieved for our approach by adopting fixed precision arithmetic as in PGBT.

In contrast to standard tests like those of Table 1, special input configurations are challenging for prior methods and are rarely mentioned. We recall that in step 2 of our algorithm (Section 6), the cost of detection of special configurations is negligible compared to the cost of the triangle–triangle intersections and 2D arrangement computations. The case of equivalent operands is reported in Table 2, and we note that for complementary operands our algorithm performs equally well. These results clearly demonstrate that our approach handles robustly such tricky configurations since it succeeded for all reported ones, in addition to being more efficient than Maya and Nef which failed for larger models.

8.2. Ascertaining benchmark

In this section we focus on the evaluation of the robustness, exactness, and generality of our approach. To this end, we performed several kinds of tests that are described below.

Randomized benchmark. A classical approach to evaluate Boolean operators consists in randomly choosing two models (not necessarily different) from a large set of samples, applying some random transformations, computing the union and intersection, and finally checking the results. In order to stress our algorithm we constructed our database with models exhibiting different topologies, and we also randomly flipped the input models to test unbounded sets that will appear not to be handled by most prior work. Figs. 1 and 9 present a very small subset of our results, and a larger selection is included in our supplemental materials (see Appendix A). Figs. 1(a) and 9(a)–(b) present single component outputs, Figs. 1(b) and 9(c)–(j) present disconnected bounded and unbounded outputs. Taking the same model for the operands with one slightly rotated is particularly challenging. For instance, the differences of *Armadillo52kf* and a slightly rotated version of it (Fig. 9(h) and (i)) are composed of more than one hundred disconnected pieces each. All our results have been automatically checked

Table 2
Special input configurations test and comparison with prior art for equivalent operands.

Operands P and Q	Ours (s)	Maya (s)	Nef (s)	PGBT
BumpySphere11kf	**20.53**	96.50	118.76	YES
RoundOcta33kf	**56.46**	900.20	341.81	YES
Dinausor40kf	**68.44**	1917.10	NO	YES
Armadillo52kf	**97.06**	2848.16	NO	YES
Bimba75kf	**116.66**	NO	NO	YES
Vase100kf	**171.10**	NO	NO	NO
Average speed-up		$\times$**19.50**	$\times$**5.92**	–

as combinatorially and geometrically closed. Whenever the Nef based technique succeeded, we checked their exactness by comparing both outputs in arbitrary precision.

Non-manifoldness. To evaluate the robustness to non-manifold inputs/outputs, we extended the previous benchmark with handcrafted models in the form of non-manifold grids of cubes. In $\mathbb{E}^3$, a mesh contains only two kinds of (intra-mesh) non-manifold primitives: non-manifold vertices (whose neighborhood is not homeomorphic to an open disk) and non-manifold edges (incident to more than two facets). Non-manifold facets do not exist because of regularization, e.g., if one wants to collate two meshes so that at least two of their triangles become opposite overlapping (hence non-manifold), then by regularization, the latter will be removed.

Based on these two exhaustive types of non-manifoldness, we performed two series of benchmarks. First, we generated several grids of cubes exhibiting non-manifold vertices (e.g., model *NonManifoldGrid3kf* of Fig. 8), and computed Boolean operations on them after being randomly flipped and transformed. See for instance Fig. 9(n)–(o). Thanks to our versatile mesh data structure and our robust classification, such cases are perfectly handled by our algorithm, regardless of their disconnectedness and (un)boundedness.

In the second series, we generated a grid of cubes exhibiting non-manifold edges (Fig. 10(a)) and computed Boolean operations. To ease comprehension and clarify illustrations, we deliberately chose a small size grid composed of four cubes (48 facets) and containing 6 non-manifold edges. Contrary to the first series of tests where we randomly flipped and transformed the operands, in this series we manually translated and flipped our operands in order to evaluate our algorithm on more challenging and extremely difficult configurations (Fig. 10(b)–(g)), by exercising several sorts of degeneracies: mixtures of inter-mesh vertex/edge/face non-manifoldness, partial equivalence/complementarity, partly coplanar/opposite overlapping triangles, full/partial edge–edge coincidence, etc. All operands P and Q reported in Fig. 10 present mixtures of inter-mesh vertex/edge/face non-manifoldness.

In Fig. 10(b), operands P and Q are translated so that they exhibit several tangentially touching facets, which were removed when computing $P^c \cap Q^c$. The corresponding union $P^c \cup Q^c$ is $\mathbb{E}^3$ (not shown in the figure). The configuration of Fig. 10(c) shows partly opposite overlapping triangles, while those of Fig. 10(d) middle and right exhibit both partly opposite and partly coplanar overlapping triangles, respectively. Fig. 10(d) also presents partial edge–edge coincidences. Cumulative Boolean operations are also robustly handled by our algorithm as shown in Fig. 10(e), where the result $P \cap Q^c$ of Fig. 10(c) has been used as the operand P for the computation of the mesh $P \cup Q$. Fig. 10(f) upper row and lower row present respective partly equivalence/complementarity configurations, where operands share a cube. The case of full equivalency/complementary, i.e. special configurations, was already discussed in Section 8.1.

Finally, in Fig. 10(g), we transformed operand Q so that one of its vertices touches tangentially a face of P and computed both union and intersection meshes. Because intersection configurations (including the current one) are exhaustively encoded in the computed 2D arrangements, such configurations are robustly handled. The union mesh shows that the face of P has been correctly triangulated according to the touching vertex of Q, while the complex intersection mesh has also been successfully computed. Again, thanks to our robust classification which deals with any closed and orientable regular meshes, regardless of their (non-)manifoldness, (dis)connectedness, (un)boundedness, or relative positions and orientations, all the aforementioned challenging configurations are perfectly handled by our algorithm. More results are provided in our supplemental material (see Appendix A).

Combinatorially opening. A variant of the previous tests consists in taking combinatorially closed models, randomly flipping them, generating some vertices along their edges to make them combinatorially open while remaining geometrically closed, computing Boolean operations, and comparing the results with those obtained by operating on the combinatorially closed versions. This benchmark checks the robustness of our approach with respect to the presence of degenerate facets and T-joints, and demonstrates its ability to implicitly perform the necessary mesh repair. Our approach successfully handled such configurations. Some results are given in the supplemental material (see Appendix A).

Cumulative Booleans. Another extension of the previous benchmarks consists in successively computing Booleans from previous Boolean results. This does not only permit to validate that our algorithm produces valid outputs, but also to check that our algorithm handles the many disjoint components produced by Booleans as input.

Boolean identities. In order to thoroughly check the exactness and robustness of our algorithm, we propose a new kind of benchmark exploiting the Boolean identities summarized in Table 3. For each pair of operands, we compute both sides of each Boolean identity, and check that the results are the same. As for the previous tests, we employed randomly flipped and

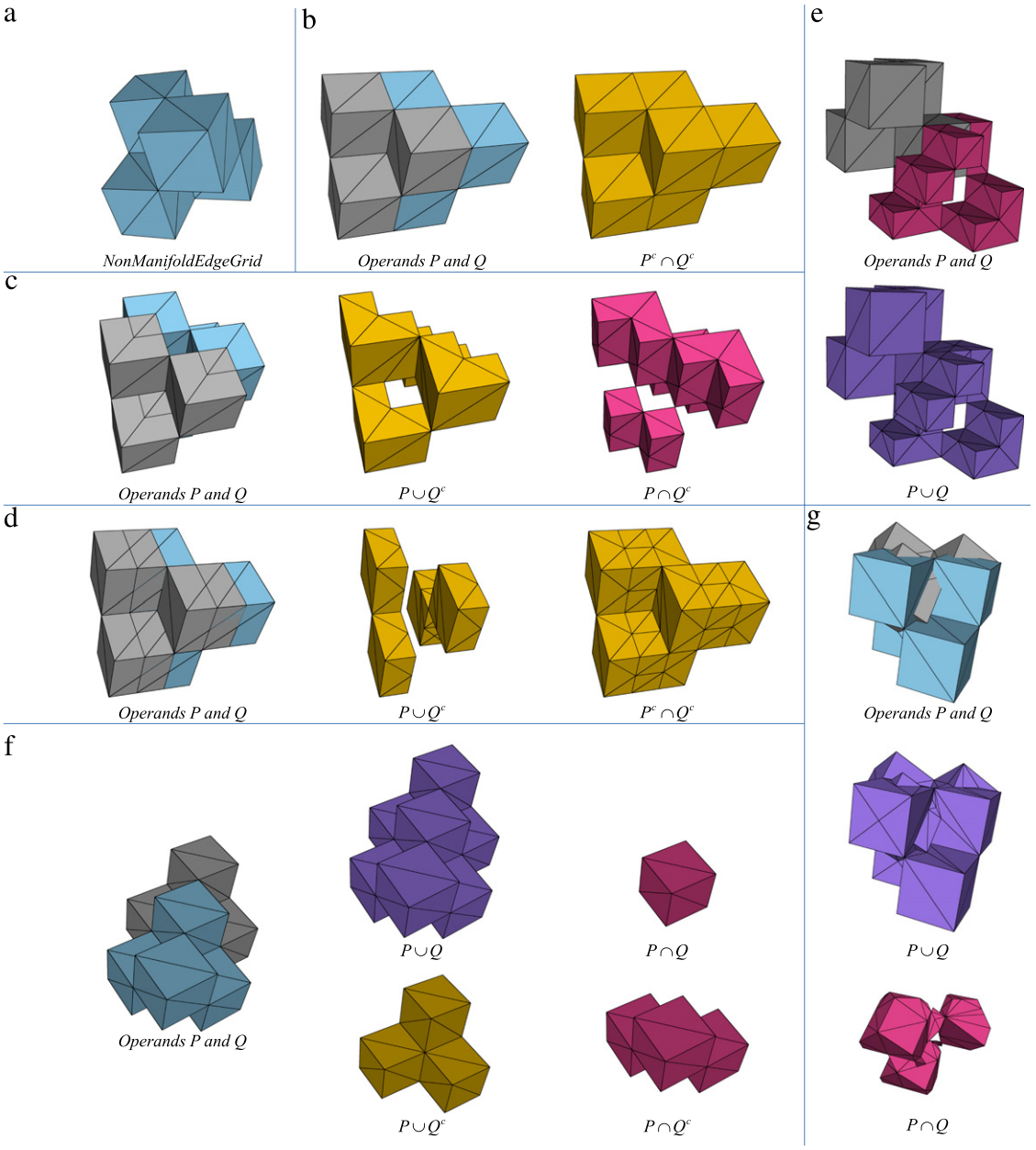

Fig. 10. Some results corresponding to extremely difficult configurations: mixtures of vertex/edge/face non-manifoldness, (un)boundedness, (dis)connectedness, partial equivalence/complementarity, partial coplanarity/opposite coplanarity, full/partial edge–edge coincidence, vertex–face intersections, etc. (a) Non-manifold grid (6 non-manifold edges) composed of 48 facets. (b)–(g) Different results obtained by computing different Booleans on the model in (a) and different transformed and complemented versions of it. Blue color is used for P, gray for Q, mixtures of purple for bounded results, and orange for unbounded ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

transformed operands from all the aforementioned kinds of input models (manifold/non-manifold, connected/disconnected, combinatorially closed/open, with degenerate facets). This fully automatic benchmark is thus very general with respect to inputs/outputs, and also naturally tests cumulative Booleans. Again, our approach successfully passed this benchmark.

Finally, we shall note that our results have been successfully double checked in arbitrary precision by: (1) comparing them with Nef results, and (2) using the Boolean identities benchmark.

8.3. Comparisons

In this section, we compare the robustness of our approach to Maya, Nef and PGBT. According to Bernstein and Fussell [10], commercial modelers like Maya are generally built on epsilon tweaking techniques. Nef polyhedra uses planes to partition

Table 3
Boolean identities.

Idempotent laws	$P \cup P = P; P \cap P = P$
Complement laws	$P \cup P^c = \mathbb{E}^3; P \cap P^c = \emptyset$
Commutative laws	$P \cup Q = Q \cup P; P \cap Q = Q \cap P$
Associative laws	$P \cup (Q \cup R) = (P \cup Q) \cup R$
	$P \cap (Q \cap R) = (P \cap Q) \cap R$
Distributive laws	$P \cup (Q \cap R) = (P \cup Q) \cap (P \cup R)$
	$P \cap (Q \cup R) = (P \cap Q) \cup (P \cap R)$
De Morgan laws	$(P \cup Q)^c = P^c \cap Q^c; (P \cap Q)^c = P^c \cup Q^c$
Absorption laws	$P \cup (P \cap Q) = P; P \cap (P \cup Q) = P$
Simplification laws	$P \cup (P^c \cap Q) = P \cup Q;$
	$P \cap (P^c \cup Q) = P \cap Q$

Table 4
Comparison of Hausdorff distances to Nef method.

Operands		Ours		Maya		PGBT	
P	Q	$\cup$	$\cap$	$\cup$	$\cap$	$\cup$	$\cap$
Casting10kf	Clutch2kf	**8e**$^{-6}$	**9e**$^{-6}$	117.38	89.90	8e^{-4}	9e^{-4}
BumpySphere11kf	BumpyTorus34kf	**0**	**0**	117.10	98.24	5e^{-4}	5e^{-4}
RoundOcta33kf	SharpSphere21kf	**0**	**0**	96.13	92.66	5e^{-4}	5e^{-4}
Armadillo52kf	Dinausor40kf	**0**	**0**	1286.60	1013.09	4e^{-3}	4e^{-3}

the Euclidean space $\mathbb{E}^3$ into cells with a *in, out* membership labeling [11]. In theory, Nef polyhedra can represent non-manifold and unbounded sets. As stated by Campen and Kobbelt [12], in order to obtain successful computations, the input meshes of PGBT must be closed, manifold, correctly oriented, free of degenerate polygons, and intersecting (neither boundary-disjoint nor nested).

A subset of our comparisons for polyhedral operands is summarized in Table 1-right. As already observed, our approach successfully computed all Booleans while the other methods exhibit several failure cases. For Nef, we observed that the process runs out of memory for large meshes because of the very greedy Nef data structure. PGBT reported the two following failure reasons: (1) "presence of boundaries and degenerate facets" (NO* entries in Table 1), and (2) "memory/time processing limits owing to complex, inconsistent, or self-intersecting input" (NO** entries in Table 1). These failures highlight deficiencies of the PGBT approach since our inputs do not present boundaries, degenerate facets, inconsistency or self-intersections, and some failing configurations of PGBT were properly handled by Maya and Nef.

A careful analysis revealed that CGAL [32] was unable to load any of the results of PGBT meaning that they are not valid polyhedra. Regarding Maya, both MeshLab and CGAL reported invalid results for four of the ten configurations of Table 1. On the contrary, both ours and Nef results passed these tests. We emphasize that all these tests have been performed under the same conditions for all the compared methods, by first rounding all the results to standard floating-point-arithmetic, before checking them with MeshLab and CGAL. The detailed verification instructions can be found in the supplemental material (see Appendix A). In addition, even if the rounding of these results may cause self-intersections, this does not explain the inconsistency of PGBT and some Maya results, since self-intersecting meshes are still successfully loaded by both MeshLab and CGAL, contrary to topologically inconsistent meshes.

In contrast, the extended fixed-precision of PGBT was supposed to improve the rounding aspect but PGBT had the worst results among compared implementations. This means that the problem is inherent to PGBT, not to rounding. We also note that all our claims are based on fair tests that have been performed on the `.off` files (after rounding to standard arithmetic) output by the different implementations.

In Section 8.2, we have already double checked the exactness of our results by comparing them to Nef results in arbitrary precision and by fulfilling the Boolean identities benchmark. Table 4 numerically compares the accuracy of our, Maya, and PGBT results, by reporting the Hausdorff distances between them and the reference results obtained with the Nef method. These numbers has been obtained through surface sampling using the Metro tool of Cignoni et al. [40]. As already confirmed in Section 8.2, our approach exhibits exact results: the small error observed in the first row is explained by numerical round-off errors inherent to the sampling/distance computations of the Metro tool itself. When they succeed, PGBT performs rather well, whereas Maya's results exhibit a large error.

We have performed other benchmarks to check the behavior of the compared techniques under topological considerations of input/output meshes. These results are summarized in Table 5 and discussed below.

We have already shown that our algorithm handles orientable and closed meshes, a class which covers all the aforementioned topologies, either separately or in combination. Non-manifold meshes were not handled by Maya although loaded successfully, not feasible with Nef because no "non-manifold mesh to Nef polyhedron" conversion mechanism is provided, and not accepted by PGBT. Theoretically, Nef polyhedra can be obtained from non-manifold meshes by half-space intersections and complements. However, this implies using extended kernels which are even slower and greedy than those currently used in CGAL's Nef [11]. Thus, such a strategy is definitely not practical.

Table 5
Topological comparison with prior art.

Topology	Approaches			
	Ours	Maya	Nef	PGBT
Non-manifoldness	**YES**	NO[1]	NO[2]	NO[2]
Unboundedness	**YES**	YES	NO[1]	NO[2]
Intra-mesh disjointness[a]	**YES**	NO[2]	YES	NO[2]
Inter-mesh disjointness[b]	**YES**	YES	YES	NO[2]
Equivalence	**YES**	YES	YES	YES
Complementarity	**YES**	NO[1]	NO[1]	NO[2]
Combinatorial opening	**YES**	NO[2]	NO[2]	NO[2]
Degenerate facets	**YES**	NO[1]	NO[2]	NO[2]

NO[1] Operands were accepted, but the computation failed.
NO[2] Operands were rejected, so the computation was not possible.
[a] One of the operands is composed of components whose boundaries do not intersect (disjoint or nested).
[b] The boundaries of the operands do not intersect each other (disjoint or nested).

For unbounded meshes, Maya succeeded but only in the case of intersecting ones, while Nef and PGBT failed in all cases. According to our experiments, bounded and unbounded meshes were both considered bounded by Nef, hence the failure to compute correct Booleans. For PGBT, the provided web service clearly identifies the restriction to bounded meshes.

Meshes composed of disjoint and closed surfaces were not managed by Maya nor by PGBT. While Maya was simply not able to load them (even if manifold), PGBT failure is due to its inability to handle unbounded (i.e., flipped) and non-intersecting meshes. In the presence of intra-disjoint meshes for which all surfaces enclose a bounded region in space and for which all pairs of surfaces coming from the two operands are intersecting, PGBT computation was successful (e.g., *Bimba75kf* and *NewYear10kf* test). In the general case, however, PGBT failed.

Inter-disjoint meshes were correctly handled by Maya and Nef but only when they are bounded. PGBT failed in such a configuration.

As we already saw in Table 2, equivalent meshes are supported by all the aforementioned approaches, provided that the models are small enough. Complementary meshes are not handled by Maya, Nef, and PGBT. For Nef and PGBT, the reason is their inability to handle unbounded objects, while Maya which is supposed to handle unbounded meshes, crashed on complementary ones.

Finally, combinatorially open meshes were not accepted by any of the three approaches because of closedness precondition violation, and the presence of degenerate facets prevented the success of Maya, Nef, and PGBT.

8.4. Benchmark summary

To summarize our experiments, the performance benchmarks of Section 8.1 have demonstrated that for general position or standard configurations, our algorithm is more efficient than Maya's implementation ($\times 3$), CGAL's robust Nef polyhedra ($\times 5$), and recent plane-based approaches. The compared approaches failed to handle larger operands as shown in Table 1. For special configurations, our algorithm outperformed both Nef and Maya, which also failed for larger models. The extensive ascertaining benchmarks of Section 8.2 revealed that our approach successfully handled all the tested challenging configurations, regardless of their (non-)manifoldness, (dis)connectedness, (un)boundedness, or relative positions and orientations, thus confirming both its exactness and robustness. Finally, the comparative benchmarks of Section 8.3 confirmed the robustness of our technique under many topological considerations, compared to Maya, Nef, and PGBT which exhibit several failure cases as summarized in Table 5.

8.5. Extensions

Even though our work focuses on closed meshes, open meshes are sometimes used as operands in Boolean computations, for instance, for trimming a closed mesh by an open mesh defining a portion of a half-space. Assuming that the output of the Boolean operation is a well defined closed mesh and that ray-shooting is not needed, i.e., the boundaries of the operand meshes intersect each other, then it is possible to handle such a configuration by ignoring border edges in the classification step. Incident triangles will be tagged as union/intersection ones by considering other incident non-border edges, or by the mesh browse.

Finally, we emphasize that our approach strives to compute *exact* Boolean results. Therefore, the resulting meshes might contain very small components and/or extremely elongated triangles nearby surface intersections. In some applications, however, it might be preferable to trade exactness for mesh quality. Since our approach produces regular, closed, and orientable meshes, such a behavior can be easily accomplished as a post-processing by, for instance, filtering out small components, and re-meshing the neighborhoods of edges coming from triangle intersections. Such tools are part of most of mesh processing and authoring software.

9. Conclusion and future work

We have presented a new approach generalizing the computation of Booleans from polyhedra to the richer set of closed and orientable meshes. By means of an extensive evaluation, we demonstrated its exactness, efficiency, and robustness to both numerical degeneracies and all imaginable topological configurations. In particular we showed that our approach seamlessly handles non-manifoldness, unbounded sets, degenerate facets, combinatorial openings, multiple component inputs, disjoint operands, as well as equivalent or complementary ones. Our benchmark comparisons revealed that any of these configurations caused failures in standard commercial solutions and state-of-the-art research implementations. We achieved such a high robustness and generality by showing how to elegantly address the *explosion of degenerate cases* faced by previous approaches.

Even though high-end performance was not our primary goal in this research, we showed that our current prototype achieved a performance similar to the fastest approach presented so far, while being much more general. The speed of our implementation is currently limited by the use of greedy exact arithmetic: the lazy kernel adapter we are using yields to predicates that are 4–5 times slower than their inexact counterparts [38]. We would like to investigate the use of more recent exact filtering predicates [41] that are almost as efficient as floating-point-based ones. Another interesting avenue for future research would be to generalize our approach to higher order B-rep, such as NURBS and subdivision surfaces that become massively used nowadays.

Acknowledgment

This publication was made possible by NPRP grant #09-906-1-137 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.camwa.2015.06.016.

References

[1] A. Requicha, Representations for rigid solids: Theory, methods, and systems, ACM Comput. Surv. 12 (1980) 437–464.
[2] D. Laidlaw, W. Trumbore, J. Hughes, Constructive solid geometry for polyhedral objects, SIGGRAPH Comput. Graph. 20 (1986) 161–170.
[3] M. Mäntylä, An Introduction to Solid Modeling, Computer Science Press, 1988.
[4] C. Hoffmann, Geometric and Solid Modeling: An Introduction, Morgan Kaufmann, 1989.
[5] E. Ferley, M.-P. Cani, J.-D. Gascuel, Practical volumetric sculpting, Vis. Comput. 16 (2000) 469–480.
[6] S. Frisken, R. Perry, A. Rockwood, T. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, in: ACM SIGGRAPH'00, 2000, pp. 249–254.
[7] A. Requicha, H. Voelcker, Boolean operations in solid modeling: Boundary evaluation and merging algorithms, Proc. IEEE 73 (1) (1985) 30–44.
[8] C. Hoffmann, J. Hopcroft, M. Karasick, Robust set operations on polyhedral solids, IEEE Comput. Graph. Appl. 9 (1989) 50–59.
[9] J. Shewchuk, Lecture notes on geometric robustness. Tech. rep., University of California at Berkeley, 2009.
[10] G. Bernstein, D. Fussell, Fast, exact, linear booleans, Comput. Graph. Forum 28 (5) (2009) 1269–1278.
[11] P. Hachenberger, L. Kettner, K. Mehlhorn, Boolean operations on 3D selective nef complexes: data structure, algorithms, optimized implementation and experiments, Comput. Geom., Theory Appl. 38 (1–2) (2007) 64–99.
[12] M. Campen, L. Kobbelt, Exact and robust (self-)intersections for polygonal meshes, Comput. Graph. Forum 29 (2) (2010) 397–406.
[13] AUTODESK Maya.
[14] K. Museth, D. Breen, R. Whitaker, A. Barr, Level set surface editing operators, ACM Trans. Graph. 21 (2002).
[15] F. Romeiro, L. Velho, L.H. de Figueiredo, Scalable GPU rendering of CSG models, Comput. Graph. 32 (5) (2008) 526–539.
[16] H. Zhao, C. Wang, Y. Chen, X. Jin, Parallel and efficient boolean on polygonal solids, Vis. Comput. 27 (2011) 507–517.
[17] L. Kobbelt, M. Botsch, U. Schwanecke, H.-P. Seidel, Feature sensitive surface extraction from volume data, in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'01, 2001, pp. 57–66.
[18] T. Ju, F. Losasso, S. Schaefer, J. Warren, Dual contouring of hermite data, ACM Trans. Graph. 21 (2002).
[19] D. Pavić, M. Campen, L. Kobbelt, Hybrid booleans, Comput. Graph. Forum 29 (1) (2010) 75–87.
[20] C. Wang, Approximate boolean operations on large polyhedral solids with partial mesh reconstruction, IEEE Trans. Vis. Comput. Graphics 17 (6) (2011) 836–849.
[21] W. Thibault, B. Naylor, Set operations on polyhedra using binary space partitioning trees, SIGGRAPH Comput. Graph. 21 (1987) 153–162.
[22] B. Naylor, J. Amanatides, W. Thibault, Merging BSP trees yields polyhedral set operations, SIGGRAPH Comput. Graph. 24 (1990) 115–124.
[23] S. Fortune, Polyhedral modelling with multiprecision integer arithmetic, Comput.-Aided Des. 29 (2) (1997) 123–133.
[24] R. Seidel, The nature and meaning of perturbations in geometric computing, Discrete Comput. Geom. 19 (1998).
[25] M. Segal, Using tolerances to guarantee valid polyhedral modeling results, SIGGRAPH Comput. Graph. 24 (1990) 105–114.
[26] B. Bruderlin, Robust regularized set operations on polyhedra. In: System Sciences, 1991. Proc. of the 24th Annual Hawaii International Conference, 1991, pp. 691–700.
[27] J. Smith, N. Dodgson, A topologically robust algorithm for boolean operations on polyhedral shapes using approximate arithmetic, Comput.-Aided Des. 39 (2) (2007) 149–163.
[28] W. Nef, Beiträge zur Theorie der Polyeder, 1978.
[29] K. Sugihara, M. Iri, A solid modelling system free from topological inconsistency, J. Inf. Process. 12 (4) (1990) 380–393.
[30] S. Fortune, C. Van Wyk, Static analysis yields efficient exact integer arithmetic for computational geometry, ACM Trans. Graph. 15 (1996) 223–248.
[31] J. Shewchuk, Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete Comput. Geom. 18 (1997) 305–363.
[32] CGAL, 2014. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org/.
[33] A. Zomorodian, H. Edelsbrunner, Fast software for box intersections, Internat. J. Comput. Geom. Appl. 12 (1–2) (2002) 143–172.
[34] R. Wein, E. Fogel, B. Zukerman, D. Halperin, 2D arrangements, in: CGAL User Manual, third ed., 2008.
[35] P. Agarwal, M. Sharir, Arrangements and their applications, in: Handbook of Computational Geometry, 1998, pp. 49–119.

[36] V. Shapiro, Solid modeling, in: G. Farin, J. Hoschek, M.-S. Kim, J. Hoschek, M.-S. Kim (Eds.), Handbook of Computer Aided Geometric Design, 2002, pp. 473–518. Chapter 20.
[37] GMP, 2014. GMP, the GNU MP Bignum Library. http://gmplib.org/.
[38] A. Fabri, S. Pion, A generic lazy evaluation scheme for exact geometric computations, in: Proc. 2nd Library-Centric Software Design, 2006.
[39] M. Campen, L. Kobbelt, WebBSP. 2010. www.graphics.rwth-aachen.de/webbsp/.
[40] P. Cignoni, C. Rocchini, R. Scopigno, Metro, 2002. http://vcg.isti.cnr.it/activities/surfacegrevis/simplification/metro.html.
[41] A. Meyer, S. Pion, FPG: A code generator for fast and certified geometric predicates, in: Real Numbers and Computers, 2008, pp. 47–60.