Computer-Aided Design 45 (2013) 529–534

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

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

Technical note

Fast and robust Booleans on polyhedra

Songgang Xu[*], John Keyser

Department of Computer Science and Engineering, Texas A&M University, United States

ARTICLE INFO

Keywords:
Solid modeling
Spatial hashing
Robust computation

ABSTRACT

This paper presents efficient methods for supporting robust Boolean operations on pairs of polyhedral manifolds. A new spatial hashing technique is proposed for detecting intersections of pairs of polygon faces quickly and robustly. Robust predicates are used to compute the exact topology of the resulting polyhedron. Degenerate intersections are identified explicitly but handled in the topological routines. Geometric positions of new vertices are then approximated using finite-precision computations. Although vertex locations are rounded off, the exact connectivity graph is kept. The faces of the rounded-off triangulated final model thus could intersect each other and have the potential for an invalid embedding.

1. Introduction

Robustness is a long-recognized concern in implementing geometric algorithms [1–3]. Boolean operations on B-rep models, introduced in the 1980s [4,5], are a geometric problem well-known for robustness issues [3]. Computing Boolean operations robustly and efficiently (i.e. *robust Boolean operations*) has proven to be a challenging and complicated task. In this paper, we combine a new intersection detection technique and a robust Boolean technique, allowing us to compute Boolean operations on polyhedra that meet a standard definition of robustness [6].

For intersection detection, hierarchical structures [7,8] and spatial hashing [9,10] are widely used for tracking geometric information and reporting (self-)intersections efficiently. We follow the two-phase idea as outlined by Mirtich [11] and propose a new spatial hashing technique for the broad phase detection. Our new hashing technique improves on earlier hashing by guaranteeing that one primitive (model face) is uniquely mapped into one hash slot. This gives better performance and requires fewer hashing slots. For the narrow phase, intersection detection between primitives generalizes point-penetrating detection [9,10] to a robust pairwise detection on primitives that is derived from that in Guigue et al. [12]. It reports the type of intersection and associates the geometric elements to yield new vertices deterministically. We use the associated vertices to achieve topological consistency in the Boolean operations.

For robust Booleans a number of approaches have been presented in academia, including several focused on robustness for objects with curved surfaces. The plane-based polyhedral representation, introduced by Sugihara and Iri [13] and improved by Fortune [14,15], provided a robust solution for many rudimentary modeling operations. Naylor and Thibault [16,17] converted Boolean operations to a binary space partition (BSP) merging operation that has been the basis for recent work on robust Booleans [18,19]. Sugihara et al. [6] divided traditional geometric computations into topological computations and arithmetic computations. While keeping the topology consistent, they tried to achieve arithmetic exactness as far as possible. Following that approach, Smith and Dodgson have proposed Boolean operators that ensure topological validity of the result [20]. Our work is inspired by that approach.

For topology computations, we detect face–face intersections using our improved spatial hashing technique. For each face intersected, we record the intersection information using a topology graph that is guaranteed to be homeomorphic to the one obtained by exact computations. For arithmetic computations, we approximate new vertices using finite-precision arithmetic. Then we create new faces from each old face directed by the topology graph, which allows us to partition the input polyhedra. Finally, different patches of models are merged via vertex pairing. Faces that are not triangles are triangulated. Using this two-pass method, we obtain topological correctness for pairwise Booleans using finite-precision arithmetic.

The main contributions of our work are:

- *Low cost in time complexity.* Our new spatial hashing method allows us to find primitive intersections much more quickly. Also, we build robust results on finite-precision computations instead of arbitrary-precision exact computations, which achieves a faster performance.
- *Topological consistency of Boolean operations.* We propose Boolean operations utilizing intersection information on intersected faces. Our method generates a topologically consistent B-rep result [6] while guaranteeing that arithmetic errors only

[*] Corresponding author.
E-mail addresses: sxu@tamu.edu (S. Xu), keyser@cse.tamu.edu (J. Keyser).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.10.036

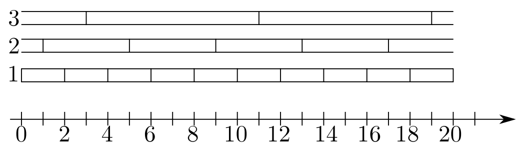

Fig. 1. Translated hierarchical space partition (1D).

happen at newly created vertices. This allows us to have robust pairwise operations, computed more efficiently than with exact computation.

- *A uniform way to treat degenerate intersections.* Although degenerate cases are detected explicitly by geometric predicates, once they are converted into the topology graph, they are treated uniformly when determining the final topology. While BSP relation approaches also handle degeneracies in a uniform way [16, 17], they rely on the exactness of arithmetic computations.

This paper deals with polyhedral models only. In the remainder of the paper, our B-reps are assumed to be triangulated manifolds in the form of a triangulation $\mathcal{G} = \{T, (\mathbf{p}_j)_{1\le j\le n}\}$, where $n \ge 3$ are the vertex indices, T is a set of triangles represented by triples of vertices, and $\mathbf{p}_j \in \mathbb{R}^3$ is the geometric position of vertex j. For space reasons, we have omitted proofs of the theorems that we present here, but they can be found in a separate technical report [21].

2. Intersection detection

For finding intersections, we follow the two-phase idea as outlined by Mirtich [11] and propose an efficient spatial hashing technique, translated hierarchical spatial hashing (THSH), for the broad phase detection. For those primitives passing the broad phase, the narrow phase will check for collisions robustly using geometric predicates.

2.1. Spatial hashing

Space partition. We partition space into a hierarchical structure of cubical tilings with different subdivision levels. For each level i, we partition the space into a set of axis-aligned cubes with an edge length of 2^i. Those partitions start from the origin point. For level i, we add a translation $2^{i-1} - 1$ to the coordinates of each cube on each dimension. Fig. 1 illustrates the translated space partitions in 1D.

Primitives hashing. We construct the hash table H to store primitives and report potential space collisions.

- Construct the axis-aligned bounding box (AABB) B_i for triangle $t_i \in T$.
- Let d be the length of the longest edge of B_i. Associate B_i with the minimum subdivision level $l + 1$, where $l = \lceil \log_2 d \rceil$ [9].
- Find the first bounding box C_i that fully encloses B_i. This will be on subdivision level $l + 1$ to $l + 4$. Denote the size of C_i as $s_i = 2^{l+k}$, where $1 \le k \le 4$.
- Associate t_i with C_i and hash $C_i(\mathbf{p}_i, s_i)$ to H using the key $k_i = \langle \mathbf{p}_i, s_i \rangle$. In 3D space, k_i is $\langle x_i, y_i, z_i, s_i \rangle$. In our implementation, we choose DJB2 [22] as the default hash function.

Theorem 1. *In d-dimensional space, provided the initial minimum level for t_i is $l + 1$, we can find a box C_i from level $l + 1$ to $l + 1 + d$ to enclose it.*

Note that objects are typically hashed closer to the minimum level $l + 1$ than the worst case $l + 1 + d$. We briefly analyze the time complexity of THSH construction. For each primitive t_i, we can obtain box B_i, region cube C_i, and key k_i in constant time. In total, given n primitives, the complexity for this part is $O(n)$. When hashing primitives, we could assume that level l is not vary large, and so each primitive is hashed in constant time. The complexity for this part is also $O(n)$. Therefore, we provide a linear hash construction algorithm. Unlike some other spatial hashing techniques [9,10], our method hashes each primitive into one hash slot. In [10], this number is not guaranteed, while in [9], this number is guaranteed to be less than 8 in 3D space. Our method reduces the number of elements in the hash table dramatically. In the following, we show that this hash structure can report intersections in an efficient way.

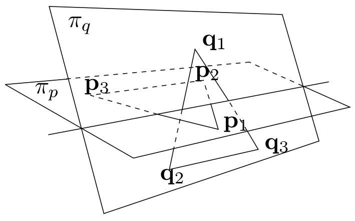

Fig. 2. General 3D intersection.

2.2. Broad phase intersection detection

The potential intersections/collisions between primitive pairs are reported in the broad phase. The general procedure is: for each subdivision level, we hash the region (query region) containing the primitive; if there is a collision in the hash table, we continue to check whether there is a geometric overlap between the query region and regions in the hash slot; if so, we report that there is a potential collision.

For example, given a point $\mathbf{p}$ in one dimension, for each level l, the region containing $\mathbf{p}$ is $[a, a + 2^l]$, where $a = \lfloor \frac{p-(2^{l-1}-1)}{2^l} \rfloor \cdot 2^l + (2^{l-1} - 1)$. We traverse each subdivision level and check for potential collisions. Given a region $[\mathbf{m}, \mathbf{n}]$ in one dimension for query, we find the bounding region $[a_m, a_m + 2^l]$ for $\mathbf{m}$ and the bounding region $[a_n, a_n + 2^l]$ for $\mathbf{n}$. We query each region from $[a_m, a_m + 2^l]$ to $[a_n, a_n + 2^l]$ and report potential collisions in the same way as we query a point.

2.3. Narrow phase intersection detection

Given four points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ in 3D, the orientation predicate is defined to be $[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] = (\mathbf{d} - \mathbf{a}) \cdot ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}))$. Robust computation of the orientation predicate has been extensively studied, and we use a robust implementation [23] for this fundamental operation, ensuring overall robustness. As in Guigue and Devillers [12], by testing 3D orientation predicates, we can detect the intersection between a pair of primitives.

As an example consider two triangles $t_p = \triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and $t_q = \triangle \mathbf{q}_1\mathbf{q}_2\mathbf{q}_3$ in Fig. 2. In [12], the essential idea is: for plane π_q, find oriented edges $\overrightarrow{\mathbf{p}_1\mathbf{p}_2}$ and $\overrightarrow{\mathbf{p}_3\mathbf{p}_1}$ going across it. For plane π_p, find oriented edges $\overrightarrow{\mathbf{q}_1\mathbf{q}_2}$ and $\overrightarrow{\mathbf{q}_3\mathbf{q}_1}$ going across it. Then for this example the predicate for intersection of two triangles is composed by testing the signs of two orientations: $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{q}_1, \mathbf{q}_2] \le 0$ and $[\mathbf{p}_1, \mathbf{p}_3, \mathbf{q}_3, \mathbf{q}_1] \le 0$.

We augment this method to handle degenerate intersection cases. For the plane of one triangle, we have to find a pair of edges from the other triangle that go across it. There are three degenerate cases, as shown in Fig. 3. Assume the red line shows the plane π_q, extending perpendicular to the paper. We will find two edges in t_p to go across it. For each edge $(\mathbf{p}_i, \mathbf{p}_j)$, we test whether two points are on different sides of plane π_q. We ignore the case where both edge ends are on the plane. This can be decided by testing the signs of orientation predicates $b_0 = [\mathbf{p}_i, \mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2]$ and $b_1 = [\mathbf{p}_j, \mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2]$. If $b_0 \cdot b_1$ is less than zero, it is a nondegenerate

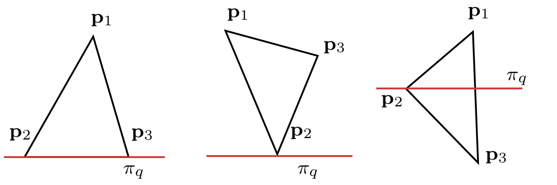

Fig. 3. Degenerate 3D intersections. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

edge intersection, and the endpoints are on opposite sides. If the product is zero, and either b_0 or b_1 is not zero, we also find the needed edge. Otherwise, we ignore it. Let the endpoint indices of one intersecting edge be u_0, u_1 and use u_2, u_3 for the other edge. Similarly, we use the indices $v_i, 0 \le i \le 4$ for the other triangle. Let $w_0 = [\mathbf{p}_{u_0}, \mathbf{p}_{u_1}, \mathbf{q}_{v_0}, \mathbf{q}_{v_1}]$ and $w_1 = [\mathbf{p}_{u_2}, \mathbf{p}_{u_3}, \mathbf{q}_{v_3}, \mathbf{q}_{v_2}]$. To determine whether two triangles intersect, we can test whether or not $w_0 \le 0$ and $w_1 \le 0$.

3. Boolean operations

In this section, we propose details of Booleans on polyhedra. Our method has three passes. In the first pass, topology analysis, we use the method in Section 2 to detect intersections, including degenerate cases. We translate collected information into a topology-based graph. In the second pass, model changing, we evaluate coordinates of new vertices. By analyzing topology information in the topology graph, we connect those vertices in a correct way to generate new faces. The third pass, vertex pairing, pairs vertices between two polyhedra. Vertices here include those in original polyhedra and those yielded by intersections. This helps to merge partitioned polyhedra to generate the final result. Since Boolean operations work on two models, for simplicity, we name them triangulations $\mathcal{G}_p$ and $\mathcal{G}_q$. The corresponding triangle sets are T_p and T_q.

3.1. Topology analysis

Under the intersection configuration, given one triangle $t_p \in T_p$, define its *face graph* as follows:

Definition 2. $G_{t_p} = \{V, E\}$ is a graph, where V is the original triangle vertices and vertices created by intersecting with a set of triangles in T_q, and where E is the segments connecting those vertices.

Since both polyhedra are manifolds, we have two properties: For each triangle t, the corresponding face graph G_t is planar. A face graph G_t can be disconnected since manifolds are not necessarily convex.

Our goal is generating robust Boolean results without using heavy-duty exact computations. Determining vertices on the face graph requires exact computation. We instead compute approximate vertices V' and use these to obtain a pseudo-face graph $G'_t = \{V', E\}$ that is homeomorphic to G_t in topology. Note that V' should have a correspondence to V and E is the same as that in G_t. Numerical error is introduced when creating new vertices.

In detail, we give a partial order to geometric elements as follows: vertex $\prec$ edge $\prec$ face. We associate each vertex with a corresponding geometric computation tuple:

Definition 3. Vertex $\mathbf{v}$ is associated with a tuple consisting of a pair of minimum elements as $v = (E_p, E_q)$, where E_p is from one model and E_q is from the other model, between which the intersection yields vertex $\mathbf{v}$.

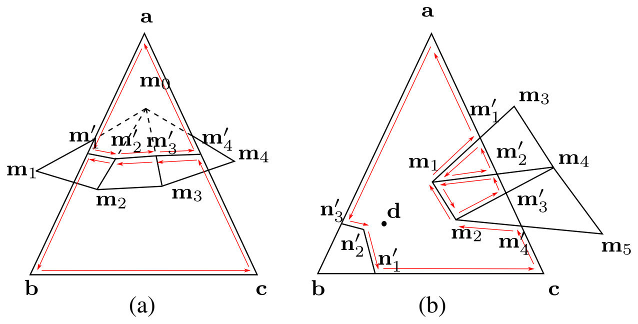

Fig. 4. Creating new faces.

Remark 4. Vertex $\mathbf{v}$ is given deterministically by the associated tuple. If the tuple is not minimal, we can use the boundary of E_p or E_q as a replacement to obtain a smaller tuple. We continue do this, until we reach the minimum.

Theorem 5. *Given an intersection vertex* $\mathbf{v}$*, any pair of primitives* E_p *and* E_q *from different manifolds, of which the intersection yields* $\mathbf{v}$*, generates the same associated topology tuple for* $\mathbf{v}$*.*

The basic requirement for creating a face graph is creating new vertices without duplication and connecting them in a correct topology. The duplication of vertices can be avoided by checking whether it already exists in the current graph using the associated topology tuple. For example, consider segment intersections in Fig. 4a. Assume we have created segment $\mathbf{m}'_1\mathbf{m}'_2$. Next, we try to create $\mathbf{m}'_2\mathbf{m}'_3$. We find that vertex $\mathbf{m}'_2$ already exists since in both triangles $\triangle\mathbf{m}_0\mathbf{m}_1\mathbf{m}_2$ and $\triangle\mathbf{m}_0\mathbf{m}_2\mathbf{m}_3$, $\mathbf{m}'_2$ is induced by $(f_{\mathbf{abc}}, e_{\mathbf{m}_0\mathbf{m}_2})$. So we do not create a new vertex for $\mathbf{m}'_2$.

Notice that some new vertices are created on original edges. We have to sort those intersections (new vertices) along a segment (edge). Formally, given $\mathbf{v}_1 = (E_1, e)$ and $\mathbf{v}_2 = (E_2, e)$, we want to sort $\mathbf{v}_1$ and $\mathbf{v}_2$ along edge e, where E_1 and E_2 can be vertices, edges or faces. We give the solution when E_1 and E_2 are both faces. Other combinations can be solved similarly. Since models are manifolds, E_1 and E_2 share one vertex, share one edge, or do not intersect.

Theorem 6. *There is a plane* π *for which* E_1 *is in the left half-space and* E_2 *is in the right half-space. We allow vertices of* E_1 *and* E_2 *to fall on* π*, but not to cross it.*

We can build plane π in Theorem 6 robustly as follows. Two triangles have six vertices. We select every triple of vertices to define a plane and test it until we find a satisfactory one. After placing two triangles in two half-spaces, we simply check where the two ends of edge e fall. This will tell us the order of intersections along edge e. The goal of generating this "pseudo-face graph" is to direct generation of new faces, since it has the correct topology intersection information on the face. Theorem 7 tells us the essential idea of how to build new faces directed by the pseudo-face graph.

Theorem 7. *On the graph, each new face is enclosed by an oriented cycle that does not contain any cycle in the same orientation inside.*

3.2. Model changing

In this pass, we evaluate coordinates of vertices and create new faces on the basis of the pseudo-face graph. According to our topology information, only two kinds of vertices need evaluation: edge–face intersections (e.g. $\mathbf{m}'_2$ in Fig. 4a) and edge–edge intersections (e.g. $\mathbf{m}'_2$ in Fig. 4b). The second case can be reduced to a 2D intersection problem.

For the first case, assume the intersecting triangles are t_p and t_q. As in narrow phase intersection, we obtain the vertex indices u_k, v_k, where $0 \leq k \leq 3$. We compute another two orientation predicates after obtaining w_0 and w_1:

$$w_2 = [\mathbf{p}_{u_0}, \mathbf{p}_{u_1}, \mathbf{q}_{v_3}, \mathbf{q}_{v_2}] \quad \text{and} \quad w_3 = [\mathbf{p}_{u_2}, \mathbf{p}_{u_3}, \mathbf{q}_{v_0}, \mathbf{q}_{v_1}]. \qquad (1)$$

The signs of these predicates indicate the endpoints of the line segment of intersection between the two triangles. If $w_2 < 0$ then one end comes from intersecting segment $\mathbf{q}_{v_3}, \mathbf{q}_{v_2}$ with plane π_p; otherwise it is from intersecting segment $\mathbf{p}_{u_0}, \mathbf{p}_{u_1}$ with plane π_q. If $w_3 < 0$ then one end comes from intersecting segment $\mathbf{q}_{v_0}, \mathbf{q}_{v_1}$ with plane π_p, and otherwise from intersecting segment $\mathbf{p}_{u_2}, \mathbf{p}_{u_3}$ with plane π_q. For example, in Fig. 2, we find two edges $(\mathbf{q}_1, \mathbf{q}_2)$ and $(\mathbf{p}_1, \mathbf{p}_2)$, and their intersections with the other plane denote the beginning and end of the intersection segment.

Theorem 7 describes the property of cycles that enclose new faces. After obtaining the topology and arithmetic information of vertices, we can create new faces by traversing the pseudo-face graph.

We first explain how to construct faces around one vertex. We assume that for each vertex $\mathbf{v}$ on a graph, its neighbors neigh($\mathbf{v}$) are sorted in counterclockwise order. We define two operations. Assume $\mathbf{u}$ and $\mathbf{v}$ are two vertices and $\mathbf{v}$ is a neighbor of $\mathbf{u}$. idx(neigh($\mathbf{u}$), $\mathbf{v}$) returns the index of $\mathbf{v}$ in the sorted neighbors of $\mathbf{u}$. succ($\mathbf{u}$, id) returns the (id + 1)th neighbor of u. Then succ($\mathbf{u}$, idx(neigh($\mathbf{u}$), $\mathbf{v}$)) returns the successor vertex of $\mathbf{v}$ in the sorted neighbors of $\mathbf{u}$. Algorithm 1 creates faces sharing vertex $\mathbf{v}$. To create all the new faces, we simply access vertices which have not been accessed and create faces sharing them without duplication. Note that vertex coordinates are approximated. Thus, resulting faces that are not triangles might not be coplanar following vertex computation (we triangulate these patches later in the process). This does not affect vertex connectivity in the following steps.

Algorithm 1: Construct faces around vertex **v**

```
for each u ∈ neigh(v) do
    v′ ← v;
    Let S be an empty ordered set;
    Insert u into S;
    do
        find w ← succ (u, idx (neigh(u),v′))
        v′ ← u, u ← w;
        Insert w into S ;
    while (w ≠ v);
    Create a new faces using vertices of S in order.
```

After creating new faces, we have to classify faces with regard to the type of Booleans. If the face of $\mathcal{G}_p$ is inside of $\mathcal{G}_q$, we mark it **IN**. If it is outside, we mark it **OUT**. If it is on the model (when there is a coplanar intersection), we mark it **ON**. For faces in $\mathcal{G}_q$, we mark in a similar way. Then the Boolean operations are computed as follows:

- $\mathcal{G}_p \cap \mathcal{G}_q$ Merge all faces marked **IN** and **ON**.
- $\mathcal{G}_p \cup \mathcal{G}_q$ Merge all faces marked **OUT** and **ON**.
- $\mathcal{G}_p - \mathcal{G}_q$ Merge **OUT** faces in $\mathcal{G}_p$ with **IN** faces in $\mathcal{G}_q$.

3.3. Vertex pairing and face triangulation

At this point, polyhedra have been separated into several patches; we must now merge them to produce the final result. The basic idea is to find a correspondence between vertices on partitioned boundaries of the two models. It is very natural to record this correspondence when splitting old faces. For example, in Fig. 4a, when creating vertex $\mathbf{v}_1$ for $\triangle\mathbf{abc}$, we also create the corresponding vertex $\mathbf{v}_2'$ for $\triangle\mathbf{m}_0\mathbf{m}_1\mathbf{m}_2$. This will build up the correspondence relationship between the boundary vertices from different models. When merging patches, we know which two vertices are related and use only one instance in the final result.

Faces having more than three vertices are triangulated. For one face, we first map it onto a 2D axis-aligned plane, which is a robust operation. We use a robust approach of Shewchuk's [24] to triangulate the 2D polygon and map the result back into 3D space.

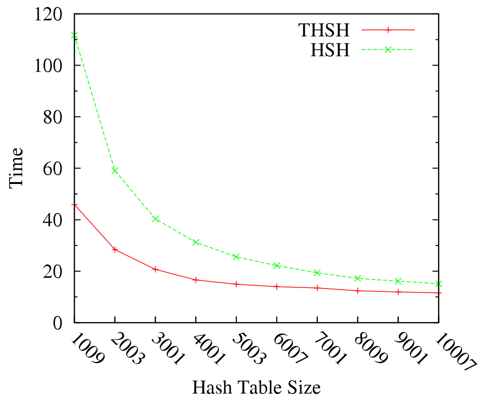

Fig. 5. Hash performance.

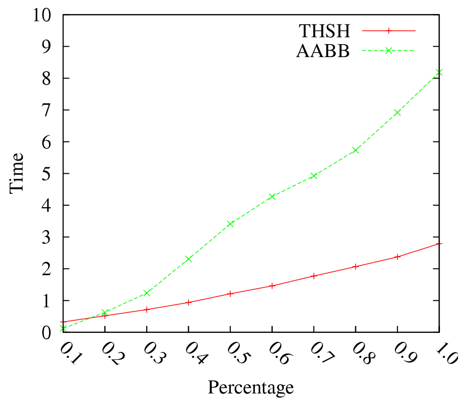

Fig. 6. Boolean performance.

4. Experiments

Our experiments are run on a notebook computer with a Core2 2.4 GHz CPU and 2 GB main memory.

First, we show a performance comparison between THSH and HSH. A model with around 100K faces is used to test performance with different hash table sizes. For each size, we query around 1.037 million intersections. Results are shown in Fig. 5. The x-axis shows hash table sizes and the y-axis time in seconds. With the same performance, THSH uses around 1/3 to 1/2 the hash slots of HSH, which means THSH uses less memory than HSH by avoiding primitive croppings. With the same number of hash slots, THSH also uses less time in querying, which means the average bucket size of THSH is less than that of HSH.

Second, we show the Boolean performance comparison between THSH and AABB tree-based detections. The Armadilloman and cat models are used for this test. We fix the cat model (with 5K faces) and change the number of faces in the Armadilloman (which has at most 60K faces). For THSH, we set the size of the hash table to be 10 007. Performance results are given in Fig. 6. The x-axis is the fraction of the maximum number of faces used in the Armadilloman model and the y-axis is time cost in seconds. Our method's performance is linear, and the tree-based method grows much faster than ours. Model results are given in Fig. 7.

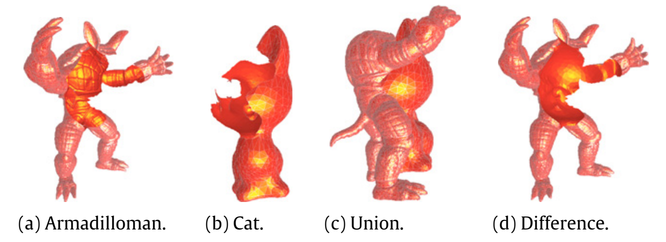
(a) Armadilloman. (b) Cat. (c) Union. (d) Difference.

Fig. 7. **OUT** patches ((a), (b)) and Boolean results ((c), (d)).

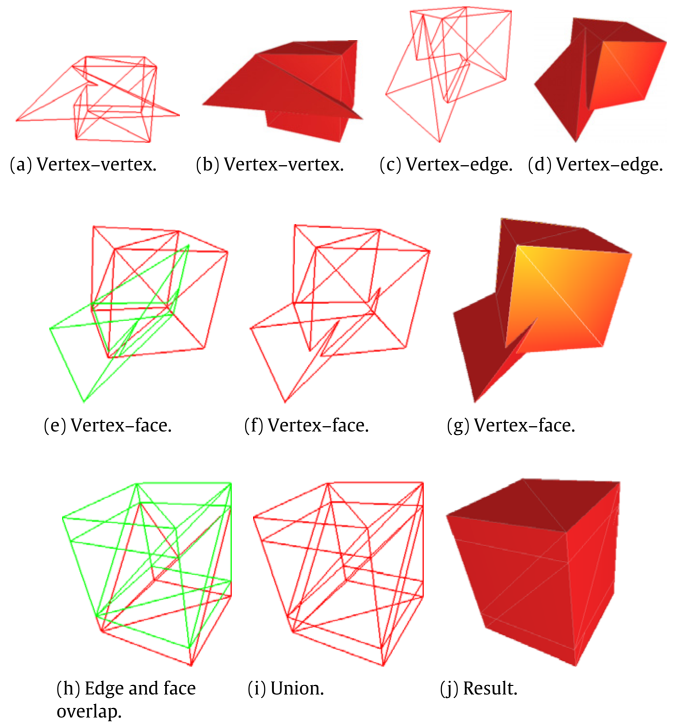
(a) Vertex–vertex. (b) Vertex–vertex. (c) Vertex–edge. (d) Vertex–edge.
(e) Vertex–face. (f) Vertex–face. (g) Vertex–face.
(h) Edge and face overlap. (i) Union. (j) Result.

Fig. 8. Degenerate Booleans.

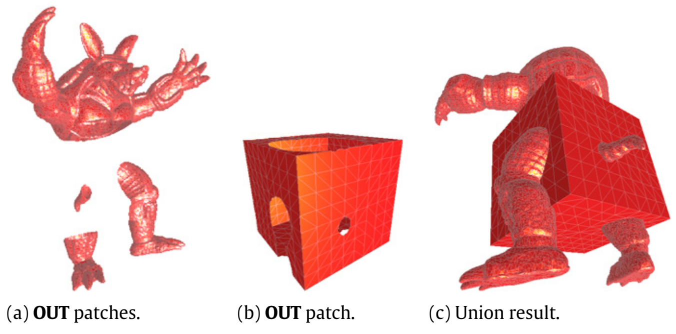
(a) **OUT** patches. (b) **OUT** patch. (c) Union result.

Fig. 9. Boolean on discontinuous patches.

Robustness results. Fig. 8 demonstrates the handling of degenerate configurations.

General result. The Armadilloman model is cut into four pieces by the cube in Fig. 9. This demonstrates that our method can handle separate model patches.

Degenerate result. In this result, we compute the union of two cube models of different mesh densities. So, the intersection of the two models has many degenerate cases, including co-planar faces, vertex–edge, edge–edge, edge–face etc. We set the splitting curve on model 1. Fig. 10(a) shows the cube 1 **OUT** patch and Fig. 10(b) the cube 2 **ON–OUT** patch. The union result is shown in Fig. 10(c).

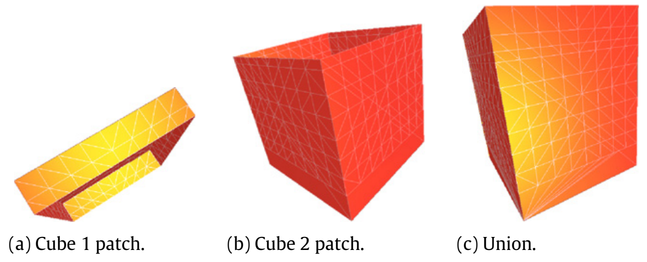
(a) Cube 1 patch. (b) Cube 2 patch. (c) Union.

Fig. 10. Degenerate case.

5. Discussion and future work

A motivation of our work was to improve the performance and robustness of Boolean computations without explicitly relying on higher-precision computation (though fixed higher-precision computation is "hidden" in the orientation predicate). THSH guarantees that one primitive is associated with at most one hash slot, so computing complexity is decided solely by the hash table design. It can be implemented robustly with standard arithmetic operations. Our method guarantees that the topology graph is homeomorphic to that obtained by exact geometric calculations.

Since our final geometric computation of vertex positions is not exact, the final output model will have round-off error that could lead to self-intersections or degeneracies. Despite this, our system meets the definitions of being both numerically robust and topologically consistent [6]. Note that the output (like the input) must be a triangular manifold, and that we are restricting ourselves to pairwise Booleans. Certain degenerate input configurations can produce non-manifold results, though other degenerate conditions are handled cleanly (see Fig. 8). Chains of Boolean operations can present other challenges not seen in pairwise intersections, such as ensuring relative ordering among different objects. Maintaining data that could be used to guarantee consistency across a tree of CSG operations would require specialized data storage from the output of one stage to the next; this is done in prior exact computation work, but goes against our goal of providing a simple tool that is usable as a stand-alone function whose input and output follow traditional model representations.

Acknowledgment

The second author's work was supported in part by NSF Grant IIS-0917286.

References

[1] Hoffmann C. Robustness in geometric computations. Journal of Computing and Information Science in Engineering 2001;1:143.
[2] De Berg M, Cheong O, Van Kreveld M. Computational geometry: algorithms and applications. New York Inc: Springer-Verlag; 2008.
[3] Hoffmann C, Hopcroft J, Karasick M. Robust set operations on polyhedral solids. Computer Graphics and Applications, IEEE 1989;9(6):50–9.
[4] Requicha A, Voelcker H. Boolean operations in solid modeling: boundary evaluation and merging algorithms. Proceedings of the IEEE 1985;73(1): 30–44.
[5] Laidlaw D, Trumbore W, Hughes J. Constructive solid geometry for polyhedral objects. ACM SIGGRAPH computer graphics, vol. 20. ACM; 1986. p. 161–70.
[6] Sugihara K, Iri M, Inagaki H, Imai T. Topology-oriented implementation—an approach to robust geometric algorithms. Algorithmica 2000;27(1):5–20.
[7] Akenine-Moller T, Haines E, Hoffman N. Real-time rendering, AK. 2002.
[8] Jiménez P, Thomas F, Torras C. 3d collision detection: a survey. Computers & Graphics 2001;25(2):269–85.
[9] Eitz M, Lixu G. Hierarchical spatial hashing for real-time collision detection. In: IEEE international conference on shape modeling and applications, 2007. IEEE; 2007. p. 61–70.
[10] Teschner M, Heidelberger B, Müller M, Pomeranets D, Gross M. Optimized spatial hashing for collision detection of deformable objects. In: Proceedings of vision, modeling, visualization VMV'03. 2003. p. 47–54.
[11] Mirtich B. Efficient algorithms for two-phase collision detection. In: Practical motion planning in robotics: current approaches and future directions. 1997. p. 203–23.

[12] Guigue P, Devillers O. Fast and robust triangle–triangle overlap test using orientation predicates. Journal of graphics tools 2003;8(1):25–32.

[13] Sugihara K, Iri M. A solid modelling system free from topological inconsistency. Journal of Information Processing 1990;12((4):380–93.

[14] Fortune S. Polyhedral modelling with multiprecision integer arithmetic. Computer-Aided Design 1997;29(2):123–33.

[15] Fortune S, Van Wyk C. Efficient exact arithmetic for computational geometry. In: Proceedings of the 9th annual symposium on computational geometry. 1993. p. 163–72.

[16] Naylor B, Amanatides J, Thibault W. Merging bsp trees yields polyhedral set operations. ACM SIGGRAPH Computer Graphics 1990;24(4):115–24.

[17] Thibault W, Naylor B. Set operations on polyhedra using binary space partitioning trees. ACM SIGGRAPH computer graphics, vol. 21. ACM; 1987. p. 153–62.

[18] Bernstein G, Fussell D. Fast, exact, linear booleans. Computer Graphics Forum, vol. 28, Wiley Online Library. 2009. p. 1269–78.

[19] Campen M, Kobbelt L. Exact and robust (self-) intersections for polygonal meshes. Computer Graphics Forum, vol. 29. Wiley Online Library, 2010. p. 397–406.

[20] Smith J, Dodgson N. A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic. Computer-Aided Design 2007;39(2):149–63.

[21] Xu S, Keyser J. Topology-oriented Boolean operations. Technical Report 2012-6-1, Texas A&M University Department of Computer Science and Engineering, 2012.

[22] McKenzie B, Harries R, Bell T. Selecting a hashing algorithm. Software: Practice and Experience 1990;20(2):209–24.

[23] Shewchuk J. Robust adaptive floating-point geometric predicates. In: Proceedings of the twelfth annual symposium on Computational geometry. ACM; 1996. p. 141–50.

[24] Shewchuk J. Triangle: engineering a 2d quality mesh generator and Delaunay triangulator. In: Applied computational geometry towards geometric engineering. 1996. p. 203–22.