Computer Aided Geometric Design ••• (••••) •••–•••

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Accelerated robust Boolean operations based on hybrid representations

Bin Sheng [a,*,1], Bowen Liu [b,1], Ping Li [c,1], Hongbo Fu [d], Lizhuang Ma [a], Enhua Wu [e,f]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[b] Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong
[c] Faculty of Information Technology, Macau University of Science and Technology, Macau
[d] School of Creative Media, City University of Hong Kong, Hong Kong
[e] Faculty of Science and Technology, University of Macau, Macau
[f] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Boolean operations
Plane-based geometry
Constructive solid geometry

ABSTRACT

Constructive Solid Geometry (CSG) is one of the popular techniques that is widely applied in 3D modeling. It combines primitive solids using Boolean operations. However, the trade-off between efficiency and robustness of Boolean evaluation is difficult to balance. Previous methods sacrifice either efficiency or robustness to achieve advantages in one perspective. Recent works attempt to achieve excellent performance in both aspects through replacing the conventional vertex-based representations (V-reps) with plane-based representations (P-reps) of polyhedrons. Different from V-reps, the P-reps use plane coefficients as meta-data and can lead to benign robustness. However, methods using P-reps have disadvantages in efficiency compared to methods using V-reps. In this paper, we proposed a Boolean evaluation approach that absorbs both the efficiency of V-reps based methods and robustness of P-reps based methods. We design a Boolean evaluation method combining P-reps with V-reps. The P-reps information is utilized for exact predicate computation while information in V-reps is collected for fast topology query and coarse tests. Our proposed approach is variadic: it evaluates a Boolean expression regarding multi-input meshes as a whole rather than a tree of decomposed binary operations. We conduct massive experiments and compare our results with those generated by the state-of-the-art methods. Experimental results show that our approach is robust for solid inputs and has advantages in performance compared to some previous non-robust methods.

1. Introduction

Constructive Solid Geometry (CSG) is a popular modeling technique for Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM). Through using regularized Boolean operations (union, intersection and difference) (Requicha, 1977; Tilove and Requicha, 1980), complex models can be easily constructed with combined primitives. There are mainly two categories of Boolean evaluation methods which are different in processing the intersections between primitives. One

* Corresponding author.
E-mail address: shengbin@sjtu.edu.cn (B. Sheng).
[1] The first three authors contribute equally to this work.

https://doi.org/10.1016/j.cagd.2018.03.021
0167-8396/

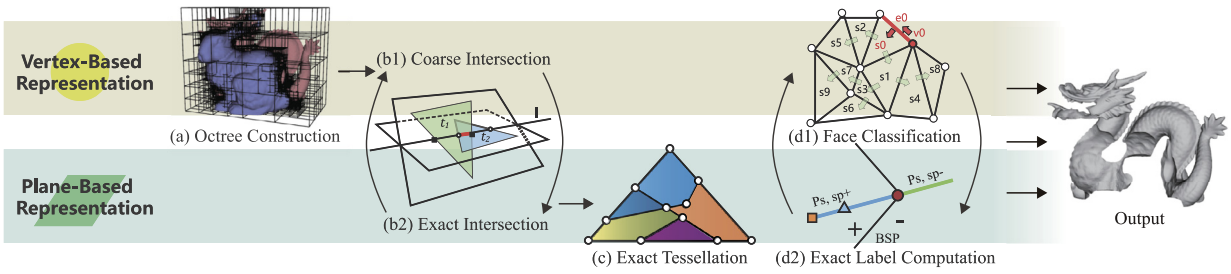

Fig. 1. An overview of our Boolean evaluation framework based on hydrid representations. Vertex-based computation is included to maintained high efficiency and plane-based computation is conducted to ensure exact geometry predicates. Our approach mainly contains three stages: intersection computation, face tessellation and face classification.

category is the *approximate method* (Pavić et al., 2010; Biermann et al., 2001; Wang, 2011). These methods approximately fit the vertexes after discretizing the intersection areas, then rearrange the topology. The other category is the *exact method* (Douze et al., 2015; Ogáyar-Anguita et al., 2015; Zhou et al., 2016). These methods are characterized by preservation of the vertex positions and maximum maintenance of multiple input elements (such as faces, vertexes, and topology). Many applications apply exact methods instead of approximate methods to pursue better accuracy. In addition, exact methods provide an accurate mapping between the input surfaces and the output meshes. This map benefits the transfer of surface information such as face colors and materials. We focus on the stream of exact methods in this paper.

There exists a trade-off between *robustness* and *efficiency* when designing Boolean algorithms. The robustness of the methods is usually guaranteed by exact arithmetic (Barki et al., 2015; Zhou et al., 2016), which is significantly slower than methods using normal floating-point arithmetic. Some methods apply other techniques such as epsilon-tweaking (Laidlaw et al., 1986; Segal, 1990; Feito et al., 2013) and numerical perturbation (Douze et al., 2015) but can only achieve **quasi-robustness** (Shewchuk, 1999). One important feature of these quasi-robust methods is the polyhedron representations based on vertexes (V-reps). However, V-reps is not the only choice. Sugihara and Iri (1989) proposed the plane-based representation (P-reps) for polyhedrons. The advantages of applying P-reps as primary geometric information is that the rudimentary modeling operations can be conducted robustly. Under P-reps, the evaluation of Boolean expressions is free from constructing new primary geometry information. In other words, no **constructions** are needed when applying P-reps and the computations are restricted to **predicates** only. Some researchers (Bernstein and Fussell, 2009; Campen and Kobbelt, 2010) picked up the P-reps and coupled them with Binary Space Partitioning (BSP) structures to develop an exact and robust Boolean operator. Although these plane-based methods are generally faster than those relying on exact arithmetic (Hachenberger and Kettner, 2005, 2006; Granados et al., 2003), plane-based methods are still limited by the high computational complexity of the BSP algorithms. Additionally, to make the boundary representations of polyhedrons compatible with the BSP structure, extra steps of conversion and connectivity reconstruction are inevitable and further deteriorates the runtime performance.

To properly handle the trade-off between efficiency and robustness, we develop a robust approach for Boolean evaluation, which has sound robustness with consistent solid inputs without sacrificing efficiency. To achieve excellent performance both in robustness and efficiency, we design a hybrid representation of solids combining P-reps with V-reps as shown in Fig. 1. It absorbs the advantages of high efficiency of methods based on V-reps and strong robustness of methods using P-reps simultaneously. In our approach, we take advantages of the V-reps information for coarse intersection and efficient neighboring face queries in the face classification process. The P-reps information is used for exact geometry computations. Our approach successfully increases the efficiency by avoiding constructions with exact arithmetic.

The architecture of our approach is more similar to vertex-based methods (Feito et al., 2013; Zhou et al., 2016) than BSP-based methods (Bernstein and Fussell, 2009; Campen and Kobbelt, 2010). Note that our approach is a systematic solution of robust Boolean evaluations rather than a simple improved version of the robust vertex-based method. Our approach mainly contains three stages: intersection detection, face tessellation and face classification. In the first stage, triangle intersections are encoded into sets of planes, which are used for determining exact tessellations (see Fig. 1 (b1) & (b2)). Subsequently, faces in the tessellated meshes are classified using small local BSP trees, which are also compatible to the P-reps for ensuring exactness (Fig. 1 (d1) & (d2)). Besides the feature of high efficiency and robustness, our method is also **variadic** (Zhou et al., 2016), which means our approach is able to evaluate the whole input meshes without decomposition. These features save a large amount of evaluation time by avoiding repetitive computation. In sum, our approach has the following contributions:

- **Plane-based intersection test:** We develop a triangle–triangle intersection test suitable for geometric objects in plane-based representation.
- **Deferred tessellation using Tess-Graph:** The tessellations of faces are conducted after all the intersections are detected and refined. To avoid errors caused by extra conversion, we develop a Tess-Graph to extract subfaces.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

- **Accelerated face classification:** In the final stage of determining whether a face belongs to the result meshes, we take advantages of label consistency to accelerate the classifying process. In addiction, evaluation results are cached for simplifying Boolean expression.

The rest of this paper is arranged as follows: Section 2 reviews literature of methods on Boolean evaluation; Section 3 provides background of Boolean evaluation based on V-reps and P-reps. We address the key problems on combining two representation in our approach; Section 4 provides an overview on our approach; Section 5, 6 & 7 provide technical details on the three stages of our approach respectively; Section 8 provides the experimental and comparison results; Section 9 summarizes the paper.

2. Related work

Boolean methods have been widely applied since their proposal (Laidlaw et al., 1986; Requicha and Voelcker, 1985; Lin et al., 2016). When applying Boolean operations on 3D solid models, robustness and efficiency is always concerned. Different techniques are applied to balance the performance and robustness.

V-reps based methods and exact arithmetic: Using V-reps as the numerical and geometrical substrates is a common practice for many tasks in computer graphics including boolean methods. Previous methods based on V-reps (Ogáyar-Anguita et al., 2015; Douze et al., 2015; Feito et al., 2013; Updegrove et al., 2016) often try to pursue robustness by using exact arithmetic which provides the most promising solution to the numerical robustness problem. Fortune et al. (Fortune and Van Wyk, 1993; Fortune, 1995) handle Boolean operations on solids bounded by piecewise linear surfaces. Their methods are optimized and integrated into CGAL (Hachenberger and Kettner, 2005; Granados et al., 2003). Keyser et al. (1999a, 1999b, 2004) develop a system called ESOLID for boundary evaluation with exact arithmetic in the curved domain. Some other approaches apply the interval computation technique. Hu et al. (1996a, 1996b) adopt rounded interval arithmetic for computing Boolean operations on solids which contains spline surfaces. Fang et al. (1993) and Segal (1990) apply intervals (which they named as tolerances) to maintain the algorithm's decision records. To pursue robustness with exact arithmetic, it seems inevitable to introduce extra computation and memory. Thus, these methods may be impractical when they are applied to arbitrary solids with many triangles. The state-of-the-art method (Zhou et al., 2016) attempts to fix a large range of topological deficiencies and achieve solid outputs through self-intersecting tests and vertex-rounding iterations. It has excellent performance in robustness and sound execution speed. But when compared to the QuickCSG (Douze et al., 2015), it still has disadvantages in efficiency. In fact, Douze et al., the authors of QuickCSG (Douze et al., 2015), pursue extreme efficiency at the cost of robustness. They restrict the input to embedded polyhedron which satisfies general position assumption. They also assume that coplanar does not exist. However, this assumption is often violated in CAD modeling. It is worth to note that their method is variadic. In other words, their evaluation of Boolean expressions treats the input meshes as an entirety.

P-reps based methods and BSP techniques: V-reps is not the only choice for representing polyhedrons in graphical systems. Sugihara and Iri (1989) first introduced the concept of Plane-based representations (P-reps) of polyhedrons in 1989. They develop the P-reps in order to avoid topological inconsistency originating from the numerical errors. Their method is based on the knowledge that when the original geometric data are represented by a finite number of bits, the relative topological configuration of two or more geometric objects can be computed accurately in finite precision. All the geometric objects in P-reps are represented by the coefficients of the surface equations. The relative topological relation between geometric objects can be converted to a four-plane problem, which can be solved precisely by finite-precision computation. Fortune (1997) noticed the advantages of P-reps and combined it with symbolic perturbation techniques at the cost of inflated algorithmic complexity.

Besides V-reps and P-reps, the Binary Space Partitioning (BSP) structures are also able to represent polyhedral objects. The BSP tree is a binary tree representing a space which is subdivided into convex sets by hyperplanes. It is originally applied to organize an arbitrary set of polygons to obtain a fast solution for the visible surface problem. Naylor, Amanatides and Thibault (Thibault and Naylor, 1987; Thibault, 1987; Naylor et al., 1990) discovered that BSP trees can provide exact representations of an arbitrary polyhedron in any dimension. In addition, they provide a method of converting the boundary representation of polyhedron to a BSP tree. Thus, evaluating a Boolean expression in the form of BSP tree become possible.

Bernstein and Fussell (2009) take advantages of both P-reps and BSP trees and develop a Boolean evaluation method which is unconditionally robust with consistent inputs. Different from vertex-based methods, the robustness of their method does not depend on exact arithmetic. Its time complexity is $O(mn)$ where m and n are the sizes of the BSP trees. Campen and Kobbelt (2010) made improvement through localizing BSP operation with an octree. In their method, mesh refinement is only conducted in the neighbor of intersections. Thus, it requires two extra operations: an extra separation of the intersection and the non-intersection region, and a reconstruction of topology.

Methods using approximate arithmetic: Since methods based on exact arithmetic may be impractical and have difficulties in processing degenerated surface–surface intersection, some researches try to compute approximate Boolean operations instead of exact ones. Most of these methods have excellent performance in execution. Biermann et al. (2001) design a

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

method for computing approximate results of Boolean operations on free-form solids bounded by multi-resolution subdivision surfaces. Smith and Dodgson (2007) design a topologically robust algorithm which can guarantee the connectivity of the output as long as the input representation has valid connectivity. Varadhan et al. (2004) accelerated their methods through reducing the complexity of output meshes. Wang (2011) reduced redundant tessellation by preserving the non-intersected areas of the input meshes. Zhao et al. (2011) adopt a compact representation of axis-aligned surfels using Layered Depth Images (LDI) as a bridge and performing Boolean operations on the structured points. With the development of General-Purpose computing on Graphics Processing Units (GPGPU), some researchers (Hable and Rossignac, 2005; Ogayar et al., 2006) explore the computational power of the graphic cards to accelerate the Boolean evaluations.

3. Preliminary

Our proposed approach is based on comprehensive investigation and analysis of previous methods. A closer look at the causes of non-robustness of previous methods helps to understand the architecture of our approach. In this section, we will give a brief introduction of Boolean evaluations and locate the origin that influence robustness. Also, the basics of plane-based representation is presented and some of the definitions will be recalled. Then we analyze the key problems that making the P-reps coexists with V-reps.

3.1. Boolean evaluation

Requicha (1980) formalized the approach of modeling with n input solids through combining pairwise operations in arbitrary boolean expressions as Constructive Solid Geometry (CSG). A boolean expression of CSG can also be represented as a binary tree where leaves represent primitives and the nodes stand for operation. Since solid objects are usually represented by segmenting its boundary into a finite number of bounded subfaces (usually called Boundary representations of solids), the boolean expression is actually determining whether the subfaces of a primitive belongs to the meshes of final result or not. Let $\boldsymbol{s}$ be a certain face. Its relationship with the final mesh is determined by the Boolean expression f:

$$f(\boldsymbol{\Lambda}(\boldsymbol{s})) = f(\lambda_1(\boldsymbol{s}), \lambda_2(\boldsymbol{s}), \cdots, \lambda_n(\boldsymbol{s})), \tag{1}$$

where $\lambda_i(\boldsymbol{s})$ is the space label with respect to primitive M_i. Each label has four conditions: completely inside (*in*), completely outside (*out*), on the boundary with consistent normal vector (*same*) or with opposite normal vector (*oppo*). To compute $f(\boldsymbol{\Lambda}(\boldsymbol{s}))$, the labels with respect to all of the primitives (represented as $\boldsymbol{\Lambda}(\boldsymbol{s})$) are required. It has been proved in Douze et al. (2015), Feito et al. (2013) that $\boldsymbol{s}$ is on the surface of the final mesh if and only if $f(\boldsymbol{\Lambda}(\boldsymbol{s})) = same$. Unfortunately, not all the input faces can be classified correctly. Faces near the intersections of primitives are partially attributed to the final mesh. Thus, an extra step of detecting meshes interactions is required before face classification. Input meshes are then tessellated to ensure that every face is classified into one of four conditions (*in*, *out*, *same*, *oppo*).

Most of the existing Boolean methods adapt the two-step scheme: intersection computation and face classification. There are three sources that numerical errors may easily be included and cause failure in robustness of these Boolean methods:

- Inaccurate computation of the intersection between faces.
- Inconsistent topology produced in face tessellation.
- Inconsistent computed results of the space label and the location of faces.

To achieve robust Boolean evaluations, operations and computations need to avoid numerical errors in the above three sources. Our approach attempt to prevent numerical errors through applying plane-based representation in the numeric substrates in these three sources.

3.2. Plane-based representation

Plane-based geometry is suggested to be suitable for conducting computation with fixed precision efficiently. We employ plane-based representation to tackle the numerical errors from the three sources we previously mentioned. The plane-based representation takes effect both in metric data but also topological data. In the conventional solid modeling systems, the metric data are usually represented with coordinates with vertexes. To apply P-reps in our approach, metric data are represented by the coefficients of face equations. Let $ax + by + cz + d = 0$ be the equation of a plane $\boldsymbol{p}$, where (a, b, c, d) are coefficients in float point numbers. These coefficients are fundamental metric data in our approach. In terms of topological data, all the geometric objects can be represented by planes. A point v can be regarded as the intersection of three planes thus we can express the point v with a non-trivial plane triple $\boldsymbol{v}$: $(\boldsymbol{p}_v^0 \cap \boldsymbol{p}_v^1 \cap \boldsymbol{p}_v^2)$. Similarly, when a line is regarded as the intersection of two planes, it is able to be represented as: $\boldsymbol{l}$: $(\boldsymbol{p}_l^0 \cap \boldsymbol{p}_l^1)$. Let the normal of a plane $\boldsymbol{p}$ be denoted as $\boldsymbol{n}(\boldsymbol{p})$, then the positive direction of the line $\boldsymbol{l}$ is defined by $\boldsymbol{n}(\boldsymbol{p}_l^0) \times \boldsymbol{n}(\boldsymbol{p}_l^1)$. In P-reps, a polygonal face can be represented using a supporting plane and several bounding planes. When trying to express a polygonal face $\boldsymbol{s}$ which has n edge, a supporting plane $\boldsymbol{p}_{s,sp}$ on which the face lies, and a set bounding planes $\{\boldsymbol{p}_{s,b}^i \mid i = 0, 1, ..., n-1\}$ are used. Each edge line $\boldsymbol{e}_{\boldsymbol{s}}^i$ is represented by the intersection $\boldsymbol{p}_{s,sp} \cap \boldsymbol{p}_{s,b}^i$. Vertex $\boldsymbol{v}_s^i$ is represented by the intersection of $\boldsymbol{p}_{s,sp}$ and two consecutive bounding planes.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

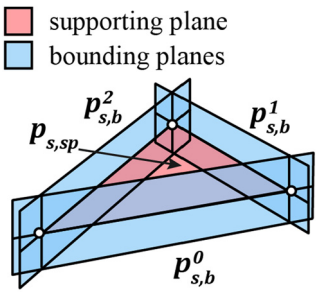

Fig. 2. Plane-based representation of a triangle. A triangle can be represented as an intersection of four planes: one supporting plane and three bounding planes.

The normal of a plane $\boldsymbol{p}$ is denoted as $\boldsymbol{n}(\boldsymbol{p})$. Specially, when trying to applying P-reps on a triangular face, four planes are needed as it is shown in Fig. 2.

We compare difference in representing a vertex with P-reps and V-reps. A vertex is represented by four planes which means with $4 \times 3 = 12$ parameters under P-reps while it takes only 3 parameters under V-reps. Thus, it easily seems that computation under P-reps can be expensive. It suggests that at least three times computational burden compared to the methods used with V-reps. This is the reason for methods based on pure P-reps do not have advantages in efficiency. Previous researches try to speed up with numerical filters (Sugihara and Iri, 1989; Banerjee and Rossignac, 1996), but the executive speed of pure plane-based methods still has distance from satisfaction. The problem of computational cost deteriorates when P-reps meet the BSP technique. BSP techniques are substantially based on planes and naturally compatible with P-reps in implementation. But the drawbacks of the BSP algorithms slower down the total speed of the P-reps based Boolean methods. For one thing, the time complexity of BSP algorithms is high. For another, BSP structure destroys geometry connectivity. It requires extra operations to reconstruct topological connectivity.

3.3. Combining V-reps and P-reps

Based on the analysis above, we arrive at an idea that to achieve high efficiency and strong robustness simultaneously through combining the P-reps and V-reps. In our approach, V-reps are applied for coarse tests and fast connectivity queries and P-reps are used in exact predicates. Through the hybrid usage of these two representations, we hope to gain the benefits of efficiency by using V-reps and robustness by using P-reps.

However, the above win–win solution cannot be achieved through direct implementation. The difficulty lies in the dimension differences. The difficulty lies in two aspects. One is the conversion from V-reps to P-reps. To avoid introducing numerical errors as far as possible, we implement the exact conversion method by Campen and Kobbelt (2010). Their converting method is able to guarantee exact calculation of the plane coefficients with controlled precision. The precision can be set according to input vertex coordinate precisions. Their approach is able to handle full IEEE 754 single precision input. In our implementation, the plane coefficient is embedded into double-precision floating point numbers. Since modern hardwares are optimized for adaptive exact predicates for floating point numbers (Shewchuk, 1997; Priest, 1991), the calculation can be conducted efficiently with determinant sign predicates (Shewchuk, 1997). One of the fundamental predicates is the relative position between a point and a plane. Suppose we have a point v, which is the intersection of the three planes $\mathbf{p}, \mathbf{q}$ and $\mathbf{r}$. Thus, the plane representation of v is $\mathbf{p} \cap \mathbf{q} \cap \mathbf{r}$. Let $\mathbf{s}$ be a plane which we are going to justify its relative relationship with the point v. Let the plane equation be

$$f(x_i) = a_i x + b_i y + c_i z + d_i, \text{ where } i \in \{p, q, r, s\} \tag{2}$$

Then, we calculate the following determinant:

$$\Delta = \begin{vmatrix} a_p & b_p & c_p \\ a_q & b_q & c_q \\ a_r & b_r & c_r \end{vmatrix} * \begin{vmatrix} a_p & b_p & c_p & d_p \\ a_q & b_q & c_q & d_q \\ a_r & b_r & c_r & d_r \\ a_s & b_s & c_s & d_s \end{vmatrix} \tag{3}$$

With the sign of Δ, we are able to obtain the relative position between the point v and the plane s. If $\Delta > 0$, point v is on the positive direction of the plane s, which mean the point is on the same side of the plane normal. The computation above is conducted with static filter technique (Shewchuk, 1997) to achieve exactness.

Another difficulty lies in the dimension differences. Most of the V-reps based algorithms perform in low dimensions which makes it difficult to be implemented under P-reps. For instance, one common operation in V-reps based 2D triangulation and face classification is to project vertex coordinates onto 2-Dimensional planes or 1-Dimensional lines. To conduct similar operation under P-reps is difficult because the vertex coordinates are not included in P-reps. To solve the dimensional incompatibility, our solutions is to format the operations in low dimensions into high dimensions. The following are

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

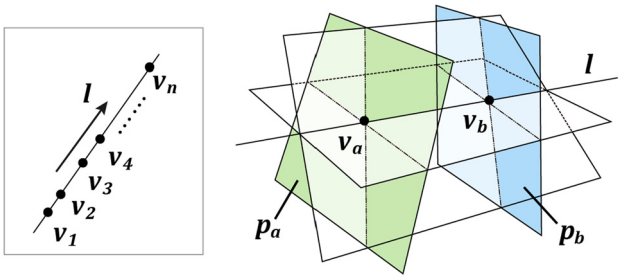

Fig. 3. Geometric configuration of linear ordering of points in 3D under P-reps. Points $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$ are both on line $\boldsymbol{l}_{ab}$. We convert this problem into the plane ordering of $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ along $\boldsymbol{l}_{ab}$.

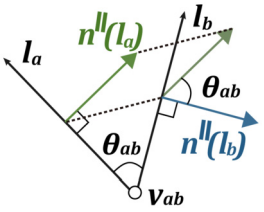

Fig. 4. Illustration of Theorem 1. As it is shown, the angle between $\boldsymbol{n}^{\parallel}(\boldsymbol{p}_a)$ and $\boldsymbol{n}^{\parallel}(\boldsymbol{p}_b)$ is exactly θ_{ab}.

three typical problems in 2D and we converted it into problems in 3D which can be solved within P-reps. The computation between objects in P-reps can be accomplished with predicates only.

Point-line orientation: The point-line orientation describes the relative position between a point and a line. In a 2D situation with V-reps, the point-line orientation is determined through computation using the line parameters and point coordinates. When computing point-line orientation in 3D with P-reps, we converted it to the problem of defining the relative position between a point and a plane which contains the line. Given a line $\boldsymbol{l}$: $(\boldsymbol{p}_l^0 \cap \boldsymbol{p}_l^1)$ and a point $\boldsymbol{v}$ within plane $\boldsymbol{p}_l^0$, we determine the point-line orientation though the relative position between $\boldsymbol{v}$ and $\boldsymbol{p}_l^1$. When $\boldsymbol{v}$ lies within $\boldsymbol{p}_l^1$, point $\boldsymbol{v}$ is obviously on line $\boldsymbol{l}$. Similarly, when point $\boldsymbol{v}$ lies in the positive direction of $\boldsymbol{p}_l^1$, it is on the right sides of $\boldsymbol{l}$.

Linear ordering of points: The linear order of points along one line can be decomposed into the problem of binary comparison (see Fig. 3). Given a line $\boldsymbol{l}$ with two points on it, $\boldsymbol{v}_a$: $(\boldsymbol{p}_a^0 \cap \boldsymbol{p}_a^1 \cap \boldsymbol{p}_a^2)$, and $\boldsymbol{v}_b$: $(\boldsymbol{p}_b^0 \cap \boldsymbol{p}_b^1 \cap \boldsymbol{p}_b^2)$, we need to determine their relative order along $\boldsymbol{l}$. To solve this problem, we choose one plane ($\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ in Fig. 3 right) that is not parallel with $\boldsymbol{l}$ from the P-rep of each point, then convert this problem into a problem of determining the linear order of planes. The converted problem can be solved by the method of Banerjee and Rossignac (1996). The chosen planes should have the same orientation with respect to $\boldsymbol{l}$ (positive dot product between the plane normal and the normal of $\boldsymbol{l}$), and unqualified planes need to be flipped.

Circular ordering of lines: During face tessellation, the neighborhoods of intersections are required. In other words, it requires circular ordering of directed lines around a vertex (see Fig. 5). We sort the lines in a divide-and-conquer way based on the relative order of each pair of lines. In this way, the problem is converted to computing the circular order of two lines $\boldsymbol{l}_a$ and $\boldsymbol{l}_b$ in a plane $\boldsymbol{p}_0$. The order is determined by the sign of $\sin\theta_{ab}$, where $\theta_{ab} \in (-\pi, \pi)$ is the angle from $\boldsymbol{l}_a$ to $\boldsymbol{l}_b$ in the top-view of $\boldsymbol{p}_0$. The sign of $\sin\theta_{ab}$ is the same as the sign of $\boldsymbol{n}(\boldsymbol{p}_0) \cdot (\boldsymbol{l}_a \times \boldsymbol{l}_b)$. However, directly computing this equation requires extra precision to explicitly compute $\boldsymbol{l}_a$ and $\boldsymbol{l}_b$. To avoid extra computation, we found an efficient solution with Theorem 1. Theorem 1 enables us to obtain the sign of $\sin\theta_{ab}$ through calculation on plane normals. Since the plane normals are three dimensional vectors, the calculation of Equation (7) is equivalent to computing a 3×3 determinant, whose elements are all floating-point numbers.

Theorem 1. *Given two directed lines $\boldsymbol{l}_a$: $(\boldsymbol{p}_0 \cap \boldsymbol{p}_a)$ and $\boldsymbol{l}_b$: $(\boldsymbol{p}_0 \cap \boldsymbol{p}_b)$ within plane $\boldsymbol{p}_0$, the following relation always stands:*

$$sign(\sin\theta_{ab}) = sign(\boldsymbol{n}(\boldsymbol{p}_0) \cdot (\boldsymbol{n}(\boldsymbol{p}_a) \times \boldsymbol{n}(\boldsymbol{p}_b))) \tag{4}$$

Proof. First, $\boldsymbol{n}(\boldsymbol{p}_a)$ and $\boldsymbol{n}(\boldsymbol{p}_b)$ are orthogonally decomposed along $\boldsymbol{n}(\boldsymbol{p}_0)$:

$$\begin{aligned} \boldsymbol{n}(\boldsymbol{p}_a) &= \boldsymbol{n}^{\parallel}(\boldsymbol{p}_a) + \boldsymbol{n}^{\perp}(\boldsymbol{p}_a) \\ \boldsymbol{n}(\boldsymbol{p}_b) &= \boldsymbol{n}^{\parallel}(\boldsymbol{p}_b) + \boldsymbol{n}^{\perp}(\boldsymbol{p}_b), \end{aligned} \tag{5}$$

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

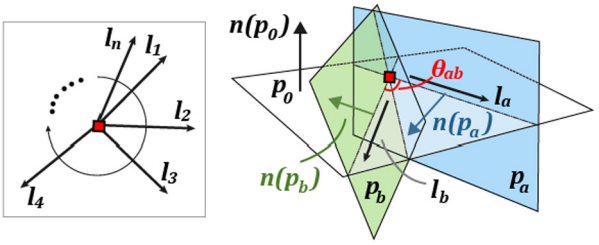

Fig. 5. Geometric configuration of circular ordering of lines in 3D under P-reps. $\boldsymbol{l}_a : (\boldsymbol{p}_0 \cap \boldsymbol{p}_a)$ and $\boldsymbol{l}_b : (\boldsymbol{p}_0 \cap \boldsymbol{p}_b)$ are within plane $\boldsymbol{p}_0$.

where the superscript $\parallel$ refers to the component parallel with $\boldsymbol{p}_0$ and $\perp$ means the component orthogonal to $\boldsymbol{p}_0$. Since $\boldsymbol{n}(\boldsymbol{p}_0)$ is orthogonal to $\boldsymbol{p}_0$, we get:

$$\boldsymbol{n}(\boldsymbol{p}_0) \cdot (\boldsymbol{n}(\boldsymbol{p}_a) \times \boldsymbol{n}(\boldsymbol{p}_b)) = \boldsymbol{n}(\boldsymbol{p}_0) \cdot (\boldsymbol{n}^{\parallel}(\boldsymbol{p}_a) \times \boldsymbol{n}^{\parallel}(\boldsymbol{p}_b)). \tag{6}$$

On the other hand, the angle between $\boldsymbol{n}^{\parallel}(\boldsymbol{p}_a)$ and $\boldsymbol{n}^{\parallel}(\boldsymbol{p}_b)$ is exactly θ_{ab} (see Fig. 4). Therefore,

$$sign(\sin\theta_{ab}) = sign(\boldsymbol{n}(\boldsymbol{p}_0) \cdot (\boldsymbol{n}^{\parallel}(\boldsymbol{p}_a) \times \boldsymbol{n}^{\parallel}(\boldsymbol{p}_b))). \tag{7}$$

By (6) and (7), the theorem is proved. □

4. Overview: Boolean evaluation based on hybrid representation

Our approach contains three stages: intersection computation, deferred tessellation and face classification. There are similarities between our approach and some vertex-based Boolean method (Zhou et al., 2016; Feito et al., 2013). However, our approach is not a simple plane-based implementation of the vertex-based algorithms. Instead, our approach is equipped with specially design algorithms to tackle the numerical errors from different sources: triangle–triangle intersection tests, triangle tessellation, and polygon classification. We give a brief introduction of the three stages of our approach in the following paragraphs and more technical details are provided in the next three sections.

Intersection computation: At the beginning, we construct an octree on all the input meshes to make our method variadic and speedup the computation. In the first stage, our main task is detecting the intersections. The computation of the intersections between pairs of triangles is based on the Möller's algorithm (Möller, 1997). To reduce the numerical errors caused by the vertex-based implementation of the Möller's algorithm, all the intersections are represented using planes (Fig. 1(b2)). We only retained the vertex-based coarse test to preserve the fast early rejection (Fig. 1(b1)). Degenerated cases in intersection (such as coplanar) are carefully processed. In addition, octrees are applied to speed up our computation (Fig. 1(a)).

Deferred tessellation: After all the intersections between triangles are determined, the input meshes need to be subdivided to ensure all intersections occur on edges and vertexes (Fig. 1(c)). In our approach, an intersection refinement is introduced to resolve the overlapped regions between the intersections. The refinement is essential to support variadic Boolean evaluation. Then we construct a structure graph called *tess-graph* to guide the exact tessellation of each face. The subdivided faces are polygons rather than triangles.

Face classification: In the final stage of approach, faces that satisfy the Boolean expression from the tessellated meshes are collected to generate the final mesh. In this stage, we utilize the connectivity information to propagate space labels in a flood-filling manner (Fig. 1(d1)). The seed face label is exactly computed by a local BSP constructed according to the neighborhood of the face (Fig. 1(d2)). In this stage, we focus on pursuing high efficiency and try to accelerate every computing steps.

5. Intersection computation

The initial stage of our approach is to detect intersections between faces with triangle–triangle intersection tests. One of the simple and efficient ways is to apply the Möller's algorithm (Möller, 1997). However, the original Möller's algorithm is vertex-based and can not be directly applied with plane-based geometry. In addition, it easily introduces numerical errors in float point arithmetics. Thus, we develop a plane-based intersection algorithm based on Möller's algorithm.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

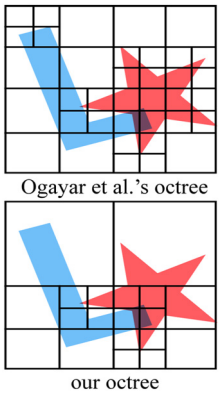

Fig. 6. Octree comparison.

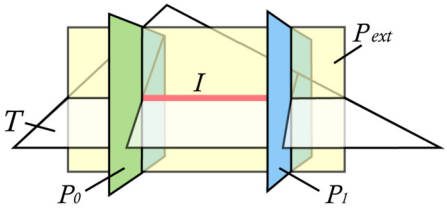

Fig. 7. The geometry of planes in a Plane-Based Intersection representation (PBI-rep). In our approach, the intersecting lines are restored in PBI-rep. The red line segment ($\mathcal{I}$) is the intersection represented by $\mathcal{I}$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

5.1. Space division

As intersection detection is performed between each pair of faces, space division is necessary to reduce the number of testing pairs. In our approach, the space division is conducted with an adaptive octree. The intersections between triangles and octree nodes are efficiently detected using the separating axis theorem (Gottschalk et al., 1996). The leaves of octree are classified into two categories. If all faces that intersect a leaf belong to the same mesh, the leaf is called a *normal cell*. Otherwise, it is regarded as a *critical cell*. The triangle–triangle intersection tests are performed in the *critical cell* only. We make a comparison between our octree and that of Ogayar et al. (2006). As it is shown in Fig. 6, the normal cells in our octree are not divided in our octree. Ogayar et al. (2006) divide all the normal cells in their octree for the purpose of the point-in-polyhedron test (Frisken and Perry, 2002), which has a limited enhancement in performance. Thus, we omit the division of the *normal cell* for saving computational time. This simplification for time saving is significantly especially when intersections between primitives are sparsely located in small regions.

5.2. Plane-based intersection test

Our intersection test is a plane-based algorithm based on the Möller's algorithm. We adapt the original Möller's algorithm to plane-based geometry and develop the corresponding plane-based algorithm. In addition to making the algorithm theoretically correct, we also provide the implementation details.

5.2.1. Plane-based intersection representation

In our approach, an intersection line segment $\mathcal{I}$ is stored as $\{T, \boldsymbol{P}_{ext}, \boldsymbol{P}_0, \boldsymbol{P}_1, \mathcal{N}\}$. We name it as the Plane-Based Intersection representation (PBI-rep, see Fig. 7). The first component, T, indicates which triangle $\mathcal{I}$ lies in. $\boldsymbol{P}_{ext}$ indicates the plane that $\mathcal{I}$ lies on. T should not be in $\boldsymbol{P}_{ext}$. Thus, the first two components indicate that $\mathcal{I}$ lies on the line $T \cap \boldsymbol{P}_{ext}$. Then, the two endpoints of $\mathcal{I}$ are $T \cap \boldsymbol{P}_{ext} \cap \boldsymbol{P}_0$ and $T \cap \boldsymbol{P}_{ext} \cap \boldsymbol{P}_1$. The last component, $\mathcal{N}$, represents the *intersection neighborhood* of $\mathcal{I}$. It is defined as the neighboring faces of $\mathcal{I}$, which are not from the primitive of T. $\mathcal{N}$ can be a single face or a set faces from one or more input primitives.

For example, two triangle faces, t_1 and t_2, originating from meshes M_i and M_j respectively, intersect. Two intersections $\mathcal{I}_{12}$ on t_1 and $\mathcal{I}_{21}$ on t_2 are generated. For $\mathcal{I}_{12}$, $T = t_1$ and $\boldsymbol{P}_{ext} = \boldsymbol{p}_{t_2,sp}$. $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$ are boundary planes of t_2, which will be discussed later. The last component $\mathcal{N} = t_2$ in general. Sometimes, $\mathcal{I}_{12}$ may lie on the edge of t_2 (see the four different conditions in Fig. 10). These are degenerate cases, called *edge intersection*. In this situation, the $\mathcal{N}$ is the set of all the faces from M_j adjacent to that edge. Edge intersection will be discussed in section 5.3.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

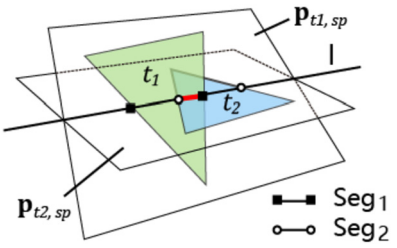

Fig. 8. Plane-based intersection detection. Seg_1 is the intersection between $\boldsymbol{p}_{t_2,sp}$ and t_1. Seg_2 is the intersection between $\boldsymbol{p}_{t_1,sp}$ and t_2. The intersection between t_1 and t_2 (segment in red line) is the overlapped region of Seg_1 with Seg_2.

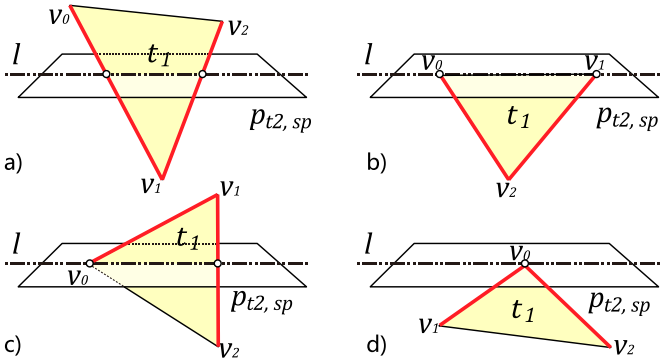

Fig. 9. We denote the signed distance from point v_i to plane $\boldsymbol{p}_{t_2,sp}$ as d_i. The four conditions of intersection between t_1 and $\boldsymbol{p}_{t_2,sp}$ are: a) $d_0 \cdot d_2 < 0$, $d_1 \cdot d_2 < 0$; b) $d_0 = 0$, $d_1 = 0$, $d_2 \neq 0$; c) $d_0 = 0$, $d_1 \cdot d_2 < 0$; d) $d_0 = 0$, $d_1 \cdot d_2 > 0$. Endpoints of Seg_1 are the intersections between $\boldsymbol{p}_{t_2,sp}$ and the related edges of t_1 (bold red lines).

5.2.2. Plane-based intersection detection

Before conducting intersection detection, we first convert each triangle to its P-reps. That is a supporting plane $\boldsymbol{p}_{t,sp}$ surrounded by three bounding planes $\{\boldsymbol{p}^i_{t,b} \mid i = 0, 1, 2\}$. Then we compute the intersections between two triangles t_1 and t_2 (shown in Fig. 8) with the following steps:

1. Testing whether t_1 intersects $\boldsymbol{p}_{t_2,sp}$, the supporting plane of t_2. The same test is also carried out between t_2 and $\boldsymbol{p}_{t_1,sp}$. If the intersections do not exist, early rejection is returned.
2. Compute the intersection between t_1 and $\boldsymbol{p}_{t_2,sp}$, denoted as Seg_1, and the intersection between t_2 and $\boldsymbol{p}_{t_1,sp}$, denoted as Seg_2 respectively.
3. Determining whether t_1 intersect t_2 by computing the overlap between Seg_1 and Seg_2. The overlap is determined according to the linear orders of the endpoints in Seg_1 and Seg_2.

Different from the original Möller's algorithm and previous methods, our plane-based intersection algorithm does not use the costly arbitrary precision arithmetic to produce exact coordinates. The exact computation is achieved with the usage of plane-based representation. In the first step, the computation of signed distances involves the vertex coordinates and the supporting planes of triangles only. Therefore, if the early rejections occur, the bounding planes are not needed at all. Additionally, the supporting plane is represented by four double-precision floating-point numbers with the conversion method of Campen and Kobbelt (2010). Let us denote precisions of the first three parameters and the last parameter to be L_a and L_d, respectively. There exists a relation $L_d = L_a + L + 1$, where L is the precision of the coordinates of input vertexes. This relation allows us to compute the signed distance exactly in double-precision. With these two facts, our early rejection is equally efficient as the rejection in original Möller's algorithm.

If early rejection is not triggered and t_1 and t_2 are not coplanar, Seg_1 and Seg_2 are computed. In fact, what we need to obtain is the endpoints of Seg_1 and Seg_2 in P-reps. The endpoints are intersections between the supporting plane of one triangle and an edge of the other triangle. By computing the bounding planes, the endpoints can be implicitly represented by plane triples. For example, the endpoint of the red overlap segment in Fig. 8 (which is represented in an empty circle), can be expressed with $\boldsymbol{p}_{t_2,sp} \cap \boldsymbol{p}_{t_1,sp} \cap \boldsymbol{p}^i_{t_1,b}$, where $\boldsymbol{p}^i_{t_1,b}$ is the bounding plane of the related edge. Fig. 9 shows all of the possible intersection conditions between t_1 and $\boldsymbol{p}_{t_2,sp}$. The coplanar situation will be discussed specifically in section 5.3. With the P-reps of the endpoints, whether Seg_1 and Seg_2 have overlap segment can be easily determined according to the linear ordering of endpoints.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

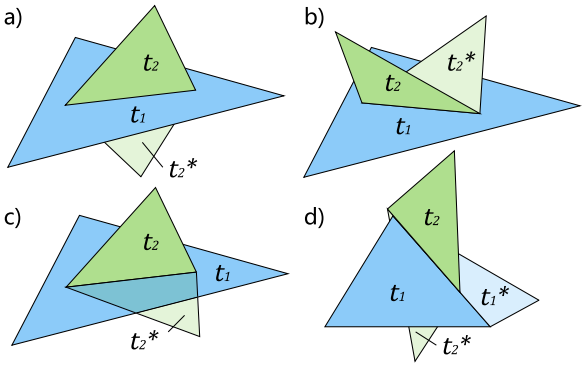

Fig. 10. There are four conditions of two triangles intersect on an edge. Triangle faces t_1 and t_1^* are companion faces. Similarly, t_2 and t_2^* are companion faces. a) t_2 and t_2^* are on different sides of t_1. b) t_2 and t_2^* are on the same side of t_1. c) t_2^* is coplanar with t_1. d) Both t_1 and t_2 have companion faces: the intersection is an edge intersection for both t_1 and t_2 (instead of only for t_2 is the previous three conditions).

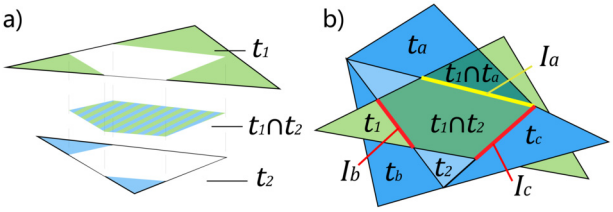

Fig. 11. a) Coplanar situation. t_1 and t_2 intersect in 2D, dividing each other into a convex overlapping area and an exclusive area. b) Possible configurations of the companion faces. t_a is coplanar with t_1, while t_b and t_c are on different sides of t_1. The blue triangles originate from the same mesh. The yellow segment is not necessarily belonged to the final mesh, and may not be detected. The red segments can be detected during intersection tests between t_1 and the companion triangles (t_b and t_c).

5.3. Handling degenerate situations

Intersections between triangles are line segments in most circumstances. But they may possibly be points in a convex area. The intersections may also be on a primitive edge. The existence of degenerate situations seriously affects the robustness of our Boolean operations. We classify the degenerate cases into three categories and provide simple but effective methods to handle these cases.

Intersection at a point: If two triangles intersect at a single point (e.g., Fig. 9(d)), the intersection cannot be represented with our PBI-reps. To guarantee correct tessellation in this situation, the intersection point is added into the related triangles. In this simple way, we avoid introducing extra intersection line segment.

Intersection on an edge: When a triangle (as t_1) intersects on the edge of another triangle (as t_2), the generated intersection is more complicated. In the case that intersection appears on the edge of triangle, t_2 has a neighboring face (as t_2^*) which intersects t_1 on the same edge (see Fig. 10 for different cases). The intersection neighborhood is a set of all faces that share a common edge, rather than a single face. For example, in Fig. 10(a), the intersection neighbors of t_1 are t_2 and t_2^*. t_2 and t_2^* are also called the *companion* triangles. The intersection between t_1 and t_2 will be detected again during intersection detection between t_1 and t_2^*. When cases of intersection on the edge appear, we only need to handle the duplication in tessellation.

Coplanar: Coplanar refers to the case that two triangle faces intersect within a common plane. As it is shown in Fig. 11(a), t_1 and t_2 divide each other into two areas: a convex overlapping area and an exclusive area. In our approach, we regard coplanar as an except case. Coplanar cases are processed as no intersections. This processing method is reasonable if cases of intersections on the edge can be properly handled. In this way, we get rid of the complex situations of 2D intersections without having any negative effects on the topological correctness. According to our observation, we notice that t_1 is actually clipped by edges of t_2 (see Fig. 11(a)). This suggests that the coplanar between two triangles can be regarded as special cases of intersection on the edge. But in coplanar cases, there can be three edge intersections at most (red and yellow line segments in Fig. 11(b)). Similar to the cases of intersection on the edge, the intersections in the coplanar cases will be detected multiple times in triangles that share common edges. In an extreme situation that all the companion triangles are coplanar, none of them will be detected in the intersection (the yellow segment in Fig. 11(b)). Fortunately, the case that all the faces of intersection neighborhood are all within the same plane is quite rare. If such an intersection enters

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

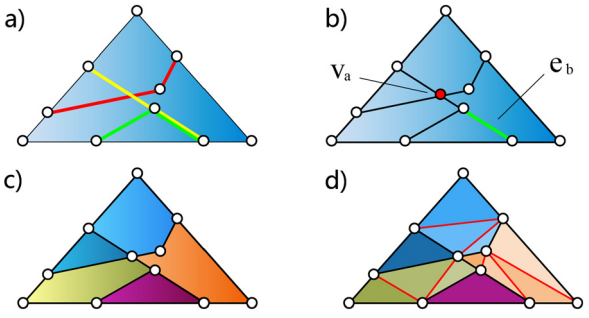

Fig. 12. a) Different colors indicate that the intersections originate from different meshes. The yellow and red intersections intersect at a point. The yellow intersection overlaps with the green intersection. b) After refinement, we introduce a new vertex $\boldsymbol{v}_a$, and merge overlapping intersections into a single edge $\boldsymbol{e}_b$. c) Our tessellation method does not guarantee that all of the faces are triangular. d) If triangulation is performed, new edges (red line segments) are introduced and double-precision is not enough to hold the plane coefficients of their P-reps.

the surface of the final mesh, the surface in the intersection neighborhood is definitely a plane. Therefore, it is safe to omit these types of intersections because the intersection is not necessary to be an edge in the final model. Since the coplanar intersections are not specially processed, the overlapping areas ($t_1 \cap t_2$ and $t_1 \cap t_a$) may have different tessellations in different primitives. But our approach is free from the inconsistent topology in the final mesh. In fact, there is no need to worry about inconsistent topology because faces from a certain primitive in such coplanar areas are either collected or abandoned together. The coplanar areas that become the surface of the final mesh will inherit only one of the tessellations.

6. Deferred tessellation

Tessellation is performed on each intersecting triangle after intersection computation. It is conducted after all the intersections are detected rather than clipping triangles incrementally at each step of intersection computing. In this stage, we first performs an intersection refinement to guarantee that intersections intersect each other only at endpoints. Then, we perform our minimal tessellation based on *tess-graph*, which is a graph-like description of the intersections on a given face.

Many methods use Constrained Delaunay Triangulation (CDT) to perform tessellation on triangles. In these methods, triangle faces are treated as the convex target zones and the triangulation is conducted using the intersections as constraints. However, those CDT algorithms (Chew, 1989; Preparata and Shamos, 1985) are performed in 2D. For 3D applications, the CDT algorithm needs to be mapped to three dimensions. Even after mapping, implementing a plane-based CDT requires the introduction of extra planes. Extra planes are included to ensure that each subface is a triangle (see Fig. 12(d)). In practice, we notice that even double-precision storage is not enough for storing the coefficients of these planes. Therefore, instead of applying CDT, we conduct a minimal tessellation on each triangle. With the minimal tessellation, subdivided faces become general polygons instead of triangles.

6.1. Intersection refinement

From the intersection test in the previous stage, we are equipped with the intersection information of the triangles. However, the intersecting lines may also intersect each other. As it is shown in Fig. 12(a), the new points can either be generated by crossing of intersecting lines (e.g., the cross point generated by the red and yellow lines) or the overlap between intersecting lines. The purpose of the refinement is to locate the points generated by the crossing of intersecting lines, and optimize the triangle–triangle intersections information before the final tessellation. We apply intersection refinement on each intersecting triangle. For a triangular face t, we collect all of the intersections in PBI-reps on t as a set $\boldsymbol{\Gamma}(t) = \{\mathcal{I} | \boldsymbol{T} = \boldsymbol{t}\}$. Note that the three edges of t are also included into the $\boldsymbol{\Gamma}(t)$ for they are also used in subface extraction. For the PBI-reps of the three edges of t, their intersection neighbor component $\mathcal{N}$ is set as N/A because they are not real intersections. The intersection refinement is conducted on the set $\boldsymbol{\Gamma}(t)$ using plane-based geometric predicates only and we denote the refined set as $\boldsymbol{\Gamma}'(t)$. The refinement mainly contains the following two steps: coincidence elimination and intersection resolving.

Coincidence elimination: If two intersecting lines have the same endpoints, they are coincident intersections. Because our intersection is recorded in PBI-rep, to determine whether two intersections $\mathcal{I}_1 = \{t, \boldsymbol{P}^1_{ext}, \boldsymbol{P}^1_0, \boldsymbol{P}^1_1, \mathcal{N}^1\}$ and $\mathcal{I}_2 = \{t, \boldsymbol{P}^2_{ext}, \boldsymbol{P}^2_0, \boldsymbol{P}^2_1, \mathcal{N}^2\}$ are coincident, we need to compare their third and forth components. That is whether $\boldsymbol{P}^1_0 = \boldsymbol{P}^2_0$ and $\boldsymbol{P}^1_1 = \boldsymbol{P}^2_1$. Since intersections are non-directive, the order of endpoints does not affect the determination of coincidence. In other words, when $\boldsymbol{P}^1_0 = \boldsymbol{P}^2_1$ and $\boldsymbol{P}^1_0 = \boldsymbol{P}^2_1$, $\mathcal{I}_1$ and $\mathcal{I}_2$ are also coincident. Coincident intersections are merged together. Suppose $\mathcal{I}_1$ and $\mathcal{I}_2$ are coincident, we merged them into one intersection $\mathcal{I}_{merged} = \mathcal{I}_1 \cup \mathcal{I}_2 = \{T, \boldsymbol{P}_{ext}, \boldsymbol{P}_0, \boldsymbol{P}_1, \mathcal{N}^1 \cup \mathcal{N}^2\}$. Note that the first four components are directly inherited from $\mathcal{I}_1$ or $\mathcal{I}_1$. The intersection neighborhood component $\mathcal{N}$ is a union of the neighbor information from $\mathcal{I}_1$ and $\mathcal{I}_1$. It suggests that $\mathcal{I}_{merged}$ may contain faces from different primitives after this phase.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

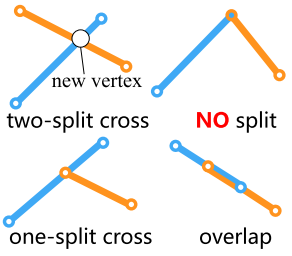

Fig. 13. Conditions of intersections.

Intersection resolving: After eliminating coincident intersections in $\mathbf{\Gamma}(t)$, we need to detect whether the remained intersecting lines intersect in any location other than endpoints. We conduct intersection test between intersecting lines from different primitives. The intersection test is a series computation on point-line orientation in P-reps. Given two intersection $\mathcal{I}_1 = \{T, \boldsymbol{P}^1_{ext}, \boldsymbol{P}^1_0, \boldsymbol{P}^1_1, \mathcal{N}^1\}$ and $\mathcal{I}_2 = \{T, \boldsymbol{P}^2_{ext}, \boldsymbol{P}^2_0, \boldsymbol{P}^2_1, \mathcal{N}^2\}$, we first compute the point-line orientation between the line $\mathcal{I}_2$ and the two endpoints of $\mathcal{I}_1$. Then compute the point-line orientation between the line $\mathcal{I}_1$ and the two endpoints of $\mathcal{I}_2$. Through the relative position of the four endpoints, we can easily determine whether $\mathcal{I}_1$ and $\mathcal{I}_2$ intersect or not. As it is shown in Fig. 13, there are three types of intersection that may occur between intersecting lines. In these situations, at least one of intersecting line is split and a new vertex is generated at the cross. The P-rep of the new generate vertex is $\boldsymbol{P}^1_{ext} \cap \boldsymbol{P}^2_{ext} \cap t$. In the situation of overlap, the PBI-rep of the overlapping segment can be easily constructed through absorb the new vertexes as the endpoint components and partially inherit other components from $\mathcal{I}_1$ and $\mathcal{I}_2$. Note that the coincidence elimination will be repeated again after intersection resolving. Because if two intersections are colinear, resolving their overlap can produce new coincident intersections.

6.2. Tessellation by tess-graph

To perform tessellation and extract subfaces, we develop a graph description of the tessellated face topology, called *Tess-Graph*. For each intersecting triangle face t, a tess-graph is constructed according to the intersections contained in the set $\mathbf{\Gamma}'(t)$. In a tess-graph, intersections are represented by the connections between nodes. The nodes of the tess-graph represent the endpoints of the intersections. The tess-graph can be easily constructed according to the PBI-rep in $\mathbf{\Gamma}'(t)$. Equipped with the tess-graph, subfaces can be easily extracted by constructing valid loops. A valid loop is a loop in the tess-graph which satisfies two criteria:

- the direction of the loop should be consistent with the face normal;
- consecutive connections on the loop should be adjacent by a circular order.

Each valid loop corresponds to an intersection-free face. After all of the valid loops are determined from the tess-graph, the corresponding face is tessellated into a series of subfaces (see Fig. 15). Noted that to facilitate face classification in the next stage, we restore the neighboring components $\mathcal{N}$ of the PBI-reps on the edges of each subfaces when constructing the valid loops.

6.3. Tess-graph with multiple components

The tess-graphs we generate are not necessarily a connected graph. If a tess-graph contains more than one connected component, we need to merge identical valid loops to generate polygons with the non-zero genus. To find identical loops, we construct an auxiliary connection C_{ext} for each inner component, which connects a vertex $\boldsymbol{v}_o$ on the outer component and a vertex $\boldsymbol{v}_i$ on the current inner component (see Fig. 14(a)). After that, we search among the connections belonging to the current inner component that intersects C_{ext} to find the one (as C_1) which is the nearest to $\boldsymbol{v}_o$. Then another search is conducted among the connections not belonging to the current inner component that intersects C_{ext}, and find the one (as C_2) which is the nearest to C_1. Finally, by locating the valid loops that C_1 and C_2 within, we find the identical pair of loops.

To guarantee that C_{ext} has an exact P-rep in double-precision, an arbitrary vertex of the current triangle is chosen as the $\boldsymbol{v}_o$. On the other hand, $\boldsymbol{v}_i$ must be the vertex generated by the intersection between a triangle and an edge (as $\boldsymbol{e}$). All intersection points introduced by triangle–triangle intersections are of this type. In this way, we obtain three vertexes with exact coordinates ($\boldsymbol{v}_o$ and the two endpoints of $\boldsymbol{e}$). Therefore, we can construct the plane on which C_{ext} lies by the same method of generating supporting planes of triangle faces (see Fig. 14(b)).

7. Face classification

In this stage, the space label vector $\mathbf{\Lambda}(\boldsymbol{s})$ of the faces are computed and faces are determined whether it can become a part of the final mesh according to the evaluation results $\lambda_f(\boldsymbol{s})$. All the faces are classified by evaluating their space

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

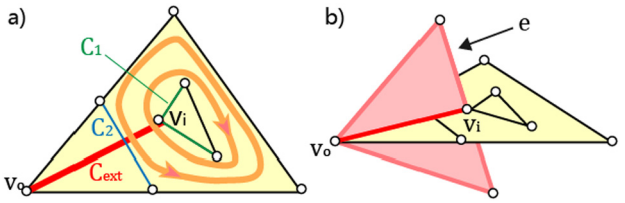

Fig. 14. a) We construct an auxiliary connection to connect each inner component with outer component (the red line segment). By checking the intersections on the auxiliary connection, we find that the orange loops are identical. b) We construct the P-rep of the auxiliary connection by $\boldsymbol{v}_o$ and the exact coordinates of the endpoints of the edge $\boldsymbol{e}$.

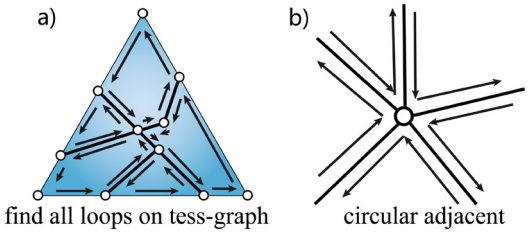

Fig. 15. a) To tessellate a triangle, all the valid loops on its corresponding tess-graph should be found. The direction of the loops must be coherent with the triangle normal (assuming here that the normal points to the outside of the paper). b) Each circular adjacent edge pair is an angle of the tessellated polygon.

label vectors $\boldsymbol{\Lambda}(\boldsymbol{s})$ with the boolean expression f (Equation (1)). Adjacent faces likely share the same space label vectors. Our classification method takes advantages of the spatial coherence of the label vectors and propagates the classification results between neighboring faces. The main challenge in the propagation is to efficiently determine whether two adjacent faces do share the same label and how to calculate the label vectors. For arbitrary adjacent faces $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, the space label vectors $\boldsymbol{\Lambda}(\boldsymbol{s}_1)$ and $\boldsymbol{\Lambda}(\boldsymbol{s}_2)$ differ if their shared edge $\boldsymbol{e}_{12}$ lies on the surface of any other primitive. Recall that the neighbor information of intersections (the $\mathcal{N}$ from PBI-rep) is retrieved in the edge of the subfaces the stage of deferred tessellation. Thus, some neighboring faces must be stored on $\boldsymbol{e}_{12}$. In addition, the neighbor information of intersections can help us to locate the different components between the space label vectors $\boldsymbol{\Lambda}(\boldsymbol{s}_1)$ and $\boldsymbol{\Lambda}(\boldsymbol{s}_2)$, and thus allow us to quickly compute the new space label with BSP technique. The outline of our classification is shown in Algorithm 1.

Algorithm 1 Fast face classification.

Input: Tessellated primitives and Boolean expression f
Output: Classification $f(\boldsymbol{\Lambda}(\boldsymbol{s}_i))$ of each face $\boldsymbol{s}_i$

```
Select a proper seed face s_0;
Compute the seed label vector Λ(s_0);
PROPAGATE( s_0 , Λ(s_0));

function PROPAGATE( s , Λ(s))
    Compute f(Λ(s));
    for each neighboring face s_{s,i} do
        if s_i has been classified then
            continue;
        end if
        if there are PBI-reps I_k on e(s_{s,i}, s) then
            compute Λ(s_{s,i}) by Λ(s) and I_k;
            PROPAGATE( s_{s,i} , Λ(s_{s,i}));
        else
            PROPAGATE( s_{s,i} , Λ(s));
        end if
    end for
end function
```

7.1. Label computation and propagation

Previous methods like Feito et al. (2013), Ogáyar-Anguita et al. (2015) evaluate the space label of a face through computing the space label of the barycenter. This computation is based on the fact that the face can be treated as a whole, and all inner points of the face share one label vector. In the methods like Zhou et al. (2016), label computation is based on points inside the cell, and the face labels are determined according to the labels on the two sides of the face. However, all the methods above require introductions of new points which take rational numbers space to restore coordinates. If the accuracy of the stored coordinates decreases, round-off errors may occur. To avoid involving extra points, we utilize the

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

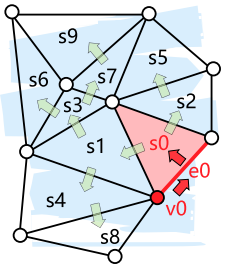

Fig. 16. Label propagation.

vertexes of a face to compute the space label. To conduct label computation based on vertexes, we need to solve two discrepancies. One discrepancy is that the space label of a vertex is not always same to the space label of the face. The other discrepancy is that the space label of a face has two more values than the space label of a vertex. For the space label of a vertex $\boldsymbol{v}$, $\lambda(\boldsymbol{s}) \in \{in, out, on\}$. But for the space label of a face ($\boldsymbol{s}$), $\lambda(\boldsymbol{s}) \in \{in, out, same, oppo\}$. This discrepancy is attributed to the orientation of a face. The label *same* and *oppo* are in fact sub labels of the label *on*.

To deal with these discrepancies, we optimize the propagating path and the initial computation. We start from a seed vertex $\boldsymbol{v}_0$, which is one of the endpoints of an edge $\boldsymbol{e}_0$ of a face $\boldsymbol{s}_0$. We assume that the label vector $\boldsymbol{\Lambda}(\boldsymbol{v}_0)$ is known. Then we use $\boldsymbol{\Lambda}(\boldsymbol{v}_0)$ to compute $\boldsymbol{\Lambda}(\boldsymbol{e}_0)$, and then $\boldsymbol{\Lambda}(\boldsymbol{s}_0)$. As shown in Fig. 16, the trace of label propagation is:

$$\boldsymbol{v}_0 \rightarrow \boldsymbol{e}_0 \rightarrow \boldsymbol{s}_0 \rightarrow \boldsymbol{s}_1 \rightarrow \boldsymbol{s}_2 \rightarrow \ldots \quad . \tag{8}$$

There are three basic paths in the propagation: $\boldsymbol{v_e} \rightarrow \boldsymbol{e}$, $\boldsymbol{e_s} \rightarrow \boldsymbol{s}$ and $\boldsymbol{s}_i \rightarrow \boldsymbol{s}_j$, where $\boldsymbol{v_e}$ is the endpoint of $\boldsymbol{e}$, $\boldsymbol{e_s}$ is the edge of $\boldsymbol{s}$, and $\boldsymbol{s}_i$ is adjacent to $\boldsymbol{s}_j$. The operations are described in detail in the following paragraphs. Before providing detail description to these three paths, we prove an important Theorem 2.

Theorem 2. *Given the partial orders on $\succ$ in and on $\succ$ out, the following relationship is true for labels within the tessellated primitives:*

$$\lambda_k(\boldsymbol{v}_{\boldsymbol{e}_s}) \succeq \lambda_k(\boldsymbol{e}_s) \succeq \lambda_k(\boldsymbol{s}), \tag{9}$$

where $\lambda_k(x)$ is the label of x for a certain primitive M_k.

Proof. We prove this theorem by contradiction. When $\lambda_k(\boldsymbol{v}_{\boldsymbol{e}_s}) \succeq \lambda_k(\boldsymbol{e}_s)$ is not satisfied, it means for a certain label λ_k, $\lambda_k(\boldsymbol{v}_{\boldsymbol{e}_s})$ is *IN* or *OUT*, but $\lambda_k(\boldsymbol{e}_s) \neq \lambda_k(\boldsymbol{v}_{\boldsymbol{e}_s})$. Let us assume that $\lambda_k(\boldsymbol{v}_{\boldsymbol{e}_s}) = in$ and $\lambda_k(\boldsymbol{e}_s) = on$ or *OUT*.

Because $\boldsymbol{v}_{\boldsymbol{e}_s}$ is inside of a closed regular set (solid) M_k, according to the continuity of space, any point in $U(\boldsymbol{v}_{\boldsymbol{e}_s})$, which is the neighborhood of $\boldsymbol{v}_{\boldsymbol{e}_s}$, should be inside of M_k. And because $\boldsymbol{e}_s^\circ \cap U(\boldsymbol{v}_{\boldsymbol{e}_s})$ is not empty (superscript $\circ$ means interior), $\boldsymbol{e}_s$ should be inside of M_k, which contradicts our assumption. □

Theorem 2 indicates that when we perform $\boldsymbol{v_e} \rightarrow \boldsymbol{e}$ and $\boldsymbol{e_s} \rightarrow \boldsymbol{s}$, from a low dimension to a high dimension, we are actually making choices in whether to preserve *ON* label or switch it to *IN*/*OUT*.

$\boldsymbol{e_s} \rightarrow \boldsymbol{s}$: According to Theorem 2, if $\lambda_k(\boldsymbol{e}_s) \neq on$, then $\lambda_k(\boldsymbol{e}_s) = \lambda_k(\boldsymbol{s})$. Conversely, when $\lambda_k(\boldsymbol{e}_s) = on$, we can build a trivial BSP (Thibault and Naylor, 1987) according to the neighboring faces which belong to M_k stored in $\boldsymbol{e_s}$. The BSP can be used to compute the evaluation value $\lambda_k(\boldsymbol{s})$ if a point can be sampled from $\boldsymbol{s}^\circ \cap \boldsymbol{U}(\boldsymbol{e}_s)$, where $\boldsymbol{U}(\boldsymbol{e}_s)$ is the neighborhood space of $\boldsymbol{e_s}$, and $\boldsymbol{s}^\circ$ stands for the interior of $\boldsymbol{s}$. However, we cannot guarantee that such a point can be found with an exact representation of double-precision. Thus, we achieve the $\boldsymbol{e_s} \rightarrow \boldsymbol{s}$ operation in an alternative way.

Let $\boldsymbol{h}_{\boldsymbol{s}}^+$ be the half of the supporting plane of $\boldsymbol{s}$ which lies on the interior side of $\boldsymbol{e_s}$ (see Fig. 18). By Theorem 3, the labels of points on the half-plane $\boldsymbol{h}_{\boldsymbol{s}}^+$ are identical. Since $\boldsymbol{h}_{\boldsymbol{s}}^+ \cap \boldsymbol{s} \neq \varnothing$, we can compute $\lambda_k(\boldsymbol{s})$ by determining $\lambda_k(\boldsymbol{v}_x)$, where the vertex $\boldsymbol{v}_x \in \boldsymbol{h}_{\boldsymbol{s}}^+$. For polygons, we can always find such a vertex, which is represented exactly (by planes or vertex coordinates). In addition, we assign each BSP splitting plane a normal vector by the normal of the triangle it contains, from which we can decide the orientation (*SAME* or *OPPO*) when $\lambda_k(\boldsymbol{s}) = on$.

Theorem 3. *The BSP is constructed by the neighboring faces around a certain edge $\boldsymbol{e}$. Then the space label of all the points on the interior of the half-plane $\boldsymbol{h}$ are the same, if 1) $\boldsymbol{e}$ is within $\boldsymbol{h}$, and 2) $\boldsymbol{e}$ is the boundary of $\boldsymbol{h}$.*

Proof. The correctness of the theorem is obvious because the supporting planes of the neighboring faces always cross the edge $\boldsymbol{e}$ and extend to infinity. □

$\boldsymbol{v_e} \rightarrow \boldsymbol{e}$: If $\lambda_k(\boldsymbol{v_e}) = on$, whether $\lambda_k(\boldsymbol{e}) = on$ can be evaluated by checking whether there are neighboring faces from M_k on $\boldsymbol{e}$. If $\lambda_k(\boldsymbol{e}) \neq on$, we need to search the location of $\boldsymbol{v_e}$ in M_k. The connectivity information contained in our tessellated meshes

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

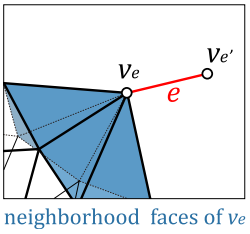

Fig. 17. Finding neighborhood faces in M_k.

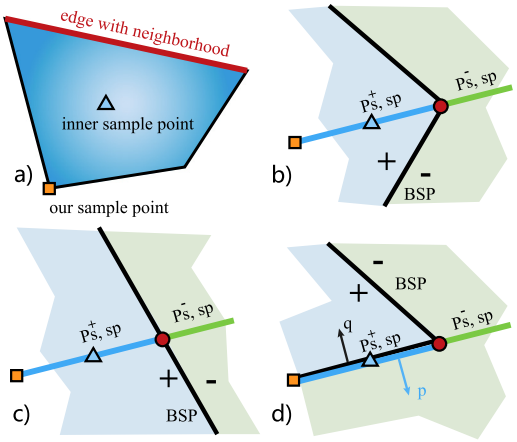

Fig. 18. a) Since the inner sample point includes geometric constructions, we choose the face vertex (orange) instead of computing the face label. b–d) Different types of classification. The simple structure of the neighborhood BSP guarantees the same classification result for all of the points on $\boldsymbol{p}^{+}_{\boldsymbol{s},sp}$. In d), the classification result is *ON* so it is necessary to determine the orientation from the normal of the BSP splitting plane q and the face normal p. In this case, the label should be *OPPO* because p and q are opposite.

allow us to easily find all neighboring faces of $\boldsymbol{v_e}$ (Fig. 17). After that, we can use a BSP-based classification method to evaluate the high dimension label $\lambda_k(\boldsymbol{e})$. Here, we use the other endpoint $\boldsymbol{v'_e}$ as the sample point for the BSP classification. The BSP-based classification method is similar to that in $\boldsymbol{e_s \rightarrow s}$.

$\boldsymbol{s_i \rightarrow s_j}$: In this path, we determine whether $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ should share the same space label. The $\lambda_k(\boldsymbol{s}_j)$ cannot inherit from $\lambda_k(\boldsymbol{s}_i)$ if their shared edge restores information of intersection neighborhood. The space label vector $\Lambda(\boldsymbol{s}_j)$ is updated based on $\Lambda(\boldsymbol{s}_i)$ with the neighboring faces stored in their shared edges. Knowing which labels differ according to the primitives that these neighboring faces belong to, the recomputation of the label vectors is exactly the same as in $\boldsymbol{e_s \rightarrow s}$.

7.2. Seed label generation

The space label of the seed vertex $\boldsymbol{v_0}$ can be generated by a point-in-polyhedron test (Ogayar et al., 2005), using the octree as an acceleration structure (Frisken and Perry, 2002). However, a simpler strategy can be adopted, which chooses a vertex with known labels as the seed. The space labels of a vertex with the maximum x-coordinate are either *OUT* (*IN*, if the complement is applied on the mesh) or *ON*. The *ON* label can be determined by connectivity. Exceptions may occur due to our ignorance of coplanar situations. Vertexes may not be added to the primitive, even if they are on the surface of the primitive. Fortunately, if a seed vertex is chosen whose neighboring faces are not all coplanar, such exceptions can be effectively avoided.

Label vectors can propagate within a certain mesh, and between different meshes across shared edges. Thus, in most cases, only a single seed vertex is needed for classification. However, if there are more than two connected components among tessellated meshes, extra seeds are required. The labels of these additional seeds have to be computed by point-in-polyhedron tests.

7.3. Acceleration by caching

In the situation that the CSG tree is large and contains hundreds of primitive nodes, the evaluations of Boolean expressions f with space label vectors of each face can be time-consuming. To accelerate our approach, we take advantages of the label space coherence to reduce the computation time by caching the evaluation results.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

Our basic strategy is to cache the final evaluation results $f(\mathbf{\Lambda(s)})$. Faces which share the same label vectors are clustered together. In a large CSG, a certain primitive often intersects with only a few other primitives $\Theta = \{M_{n_1}, M_{n_2}, \cdots, M_{n_x}\}$. All of the faces in this primitive have the same labels with respect to primitives which are not in Θ. Therefore, once we determine these fixed labels and simplify the Boolean function, we can compute the final label in this primitive based on the simplified expression. An intermediate-result cache strategy is also applied for simplification in our approach. A Boolean expression can be simplified if some components of label vector are known. For example, assume we have a Boolean expression $f = M_1 \cup (M_2 \cap M_3 - M_4)$. Given the values of two labels $\lambda_1(\boldsymbol{s}_i) = out$, $\lambda_2(\boldsymbol{s}_i) = in$, the expression can be rewritten as $f(\lambda_1 = out, \lambda_2 = in) = out \cup (in \cap M_3 - M_4)$. Using the combination rules, we can simplify the expression to $f(\lambda_1 = out, \lambda_2 = in) = M_3 - M_4$.

8. Experimental results

Our method is designed for boolean evaluations on multi input of primitives (Requicha and Voelcker, 1985). Our current implementation requires primitives to be triangular solid meshes without self-intersection. Geometry connectivity is also required as inputs. To evaluate the performance of our approach, we implement our approach in C++ and apply it to massive models with different features. All the experiments are run on a desktop computer with an Intel Core i5 CPU and 8GB of RAM. We compare the results generated by our approach with previous methods including CGAL (Hachenberger and Kettner, 2006), Cork (Bernstein, 2013), QuickCSG (Douze et al., 2015), Carve (Sargeant, 2011), the online service of Campen and Kobbelt's plane-based method (Campen and Kobbelt, 2010; Kobbelt, 2010) and the method by Zhou et al. (2016) which is distributed in libigl. Our implementation, which is used for the following experiments and comparisons, is tailored to input given in single precision floating point vertex coordinates. It is guaranteed to be robust and give correct results if the input meshes are solids (free of self-intersections and free of degenerate triangles). Of course we can truncate any input to single precision, but this can in some cases cause self-intersections or degeneracies, such that our method cannot be applied correctly. Some of the robust methods we compare to, by contrast, more generally handle full double precision input robustly. This must be taken into consideration when interpreting the results.

8.1. Self-union on Thingi10K dataset

The Thingi10K dataset (Zhou et al., 2016) contains 9956 models in .stl files, which are heavily biased towards 3D printing modeling by amateurs or semi-professionals. However, not all the 9956 models are suitable for comparison. Because models in .stl files are triangle streams rather than meshes. We merge the exact coincident vertices to reconstruct connectivity and check the cleanness of all of the models. Among the 9956 meshes, 4509 models meet the requirements of solid. The face numbers of these meshes generally follow a log-normal distribution with the average of 17000. Our experiments are conducted on these solid meshes. To enable comparison, we also apply different methods on the qualified models with CGAL (Hachenberger and Kettner, 2006), Cork (Bernstein, 2013), Carve (Sargeant, 2011), QuickCSG (Douze et al., 2015) and LibiGl (Zhou et al., 2016) respectively. The method by Campen and Kobbelt (Campen and Kobbelt, 2010; Kobbelt, 2010) is not included in the comparison of this section. The web service of Campen and Kobbelt (Campen and Kobbelt, 2010; Kobbelt, 2010) does not support batch processing. Therefore, we made a few tests manually. However, roughly a half of them failed. But for a fair comparison, the manual tests are not included.

Our approach successfully generates results in 98.8% of the tests. Although there is still a small 1.2% gap compared to the-state-of-the-art LibiGL, our approach is still competitive in robustness. By robust, we mean that algorithms successfully produce output for solid input. Cleanness and completeness of the output are also essential. The cleanness of resulting meshes is evaluated by checking, open boundaries, self-intersecting (the VCG library (Cignoni, 2015)) and the total signed incidence of every edge (Zhou et al., 2016). Solid meshes are free of self-intersections and open boundaries. Meshes with zero total signed incidence in every edge is a strict subclass of PWN meshes. These criteria are necessary conditions for a solid mesh. As it is shown in Fig. 19, over 80% of our results are solids, which outperform other methods except for the state-of-the-art LibiGL. It is easy to notice relatively low rates of success for Cork and QuickCSG. The reason is that both methods are based on general position assumption, which is violated in the case of self-union. Even after perturbation, solid results can hardly be produced as shown in Fig. 23.

Although our approach successfully generates desired results in the test, there are still failure cases and unclean results in our experiments. The reasons lie on the degenerate faces and close-to-degenerate faces from the inputs. The normal vectors of degenerate faces cannot be defined and cause failure in face classification. In addition, close-to-degenerate faces may become degenerate faces or lead to self-intersecting (see Fig. 20) after conversion to their P-reps. This is caused by the coordinates rounding-off required in the conversion method we applied (Campen and Kobbelt, 2010). The vertexes of the final mesh are represented as either planes or vertex coordinates. The vertexes originating from the input meshes have exact coordinates, and the vertexes newly introduced by the intersection between meshes have only P-reps, and require round-off when computing their coordinates. Although our approach guarantees the correct topology in the final mesh, round-off errors may still cause topological deficiencies. This is called the **vertex rounding** problem. This problem can be solved iteratively with the method by Zhou et al. (2016). However, this step is not always necessary such as in the situation that pursues extreme efficiency.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

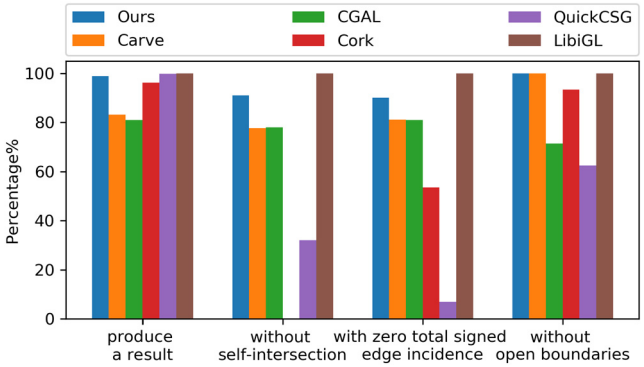

Fig. 19. The result meshes of our method for the self-union of Thingi10k are cleaner than most previous methods. The Cork and QuickCSG assume the general position and cannot handle self-union at all, so they failed to produce solids in most of the cases.

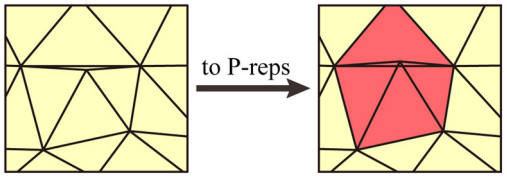

Fig. 20. After coordinates approximation (right), the "thin" triangle in the middle intersects adjacent faces and results in self-intersection on the mesh (red triangles).

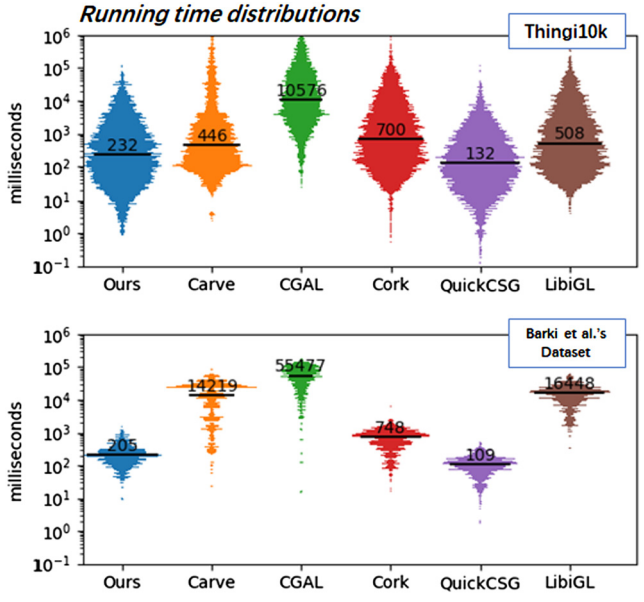

Fig. 21. The performance of tests on Thingi10k (Zhou et al., 2016) and Barki et al.'s dataset (Barki et al., 2015). We outline the median of processing time for each method. The different histogram areas result from the different success rates. Our approach is competitive with the QuickCSG (Douze et al., 2015) and is faster than the state-of-the-art LibiGL (Zhou et al., 2016).

In terms of the computation time, our method outperforms most of the other robust methods (see the violin histogram in Fig. 21 *top*). It is worth to note that, all though our approach is 1.2% lower than the state-of-the-art LibiGL in the rate of generating reasonable results in self-union tests, our approach is more than 2 times faster than LibiGL. Even compared with the fastest non-exact QuickCSG, our method takes only about two times of the processing time. We profile each stage of our method, and the results indicate the bottleneck is at intersection computation (see Fig. 24 *top*). The number of the triangles is also an important factor that influence the processing time. From Fig. 22, the linear relationship between triangle number and processing time implies the ability of our method to handle large Boolean evaluations.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

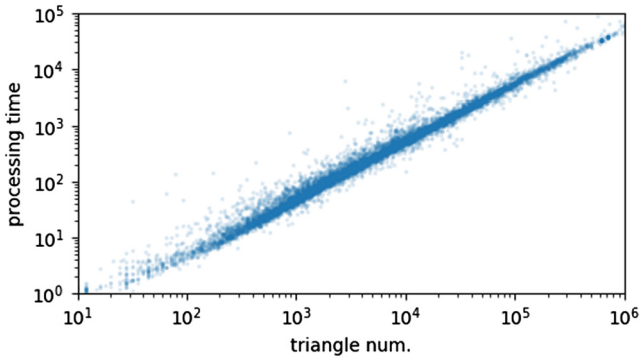

Fig. 22. For the self-union tests on Thingi10k, there exists a linear relation between the triangle number the processing time of our approach.

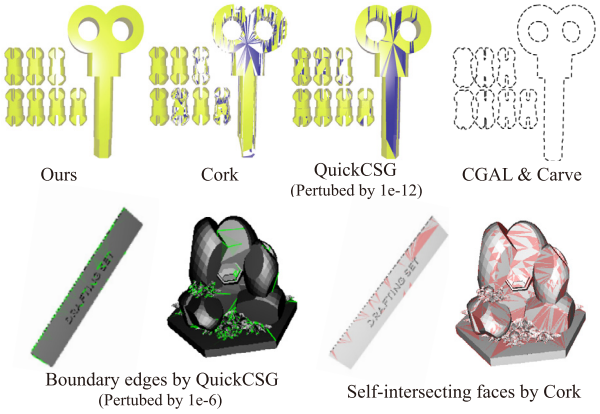

Fig. 23. *First row*: the results of self-union of *components*. QuickCSG (adding perturbation of 1e−12) and Cork cannot produce valid results. CGAL and Carve simple crashed and did not give any result for this example. *Second row*: In our self-union tests, QuickCSG tends to produce results with many boundary edges (green lines), and Cork tends to produce many self-intersecting faces (red faces).

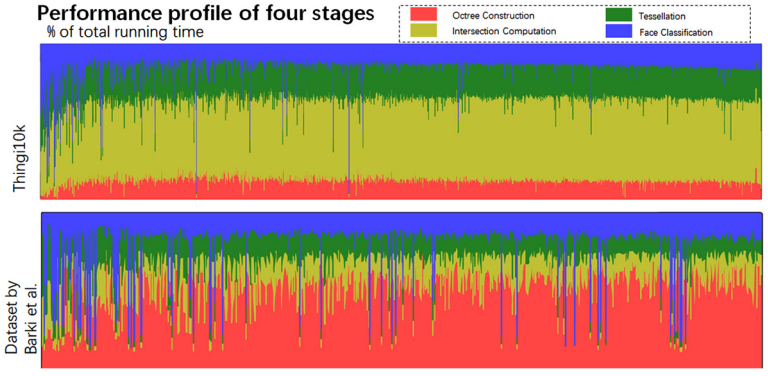

Fig. 24. Performance profile of tests on Thingi10k (Zhou et al., 2016) and Barki et al.'s dataset (Barki et al., 2015). The time percentage of different stages are represented in different color. Each one-pixel column represents a test. All the tests are sorted by total processing time. Noted that different from the performance profile on Thingi10k, in the experiments on the dataset by Barki et al., the bottleneck shifts from intersection computation to Octree constructions.

8.2. Binary Boolean operations

The most common situations of Boolean evaluations in CAD are binary operation. 3D model designer and professionals frequently refine their models by adding or subtracting one shape in their tasks. Thus, we perform comparisons of binary Boolean evaluations to test whether our approach can be properly applied to this situation. We conduct a series of union and intersection tests on the dataset of Barki et al. (2015). The dataset of Barki et al. contains 26 triangle meshes. All of them are closed and manifold. We exhaustively perform union, intersection, and both asymmetric differences for all pairs, producing $26 \times (26-1)/2 \times 4 = 1300$ results.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

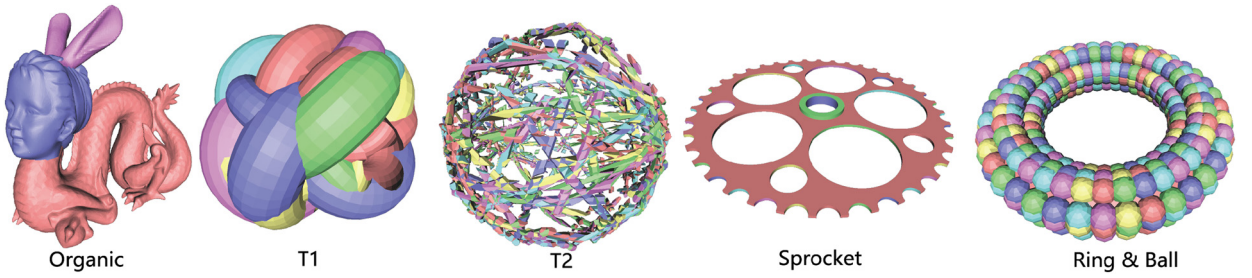

Fig. 25. The resulting meshes of our method for variadic Boolean evaluations. From the left to the right is Organic, T1, T2, Sprocket, and Ring & Ball, respectively. The performance of our method is better on cases with loosely distributed primitives (T2, Sprocket, Ring & Ball).

The configuration of our tests includes the general position assumption. This assumption enables Cork and QuickCSG to produce correct results in more cases than during self-union tests. Note that such tests are performed only in the situation that completeness and coplanar faces are emphasized in CSG modeling. As it is shown in Fig. 21 *bottom*, our approach has significant advantages compared to other exact methods. When compared to the non-exact method QuickCSG, our approach has slight disadvantages in speed. It is worth to note that the bottleneck shifts from intersection computation to octree construction in this test (see Fig. 24 *bottom*). The shift of the bottleneck is caused by the reduction in intersections. Compared with self-union tests, the binary Boolean evaluations generally process fewer intersections. Another point that worth to notice is our approach have dramatic differences in performance when comparing with Carve. The differences is caused by our localizing the Boolean evaluations within intersection areas.

8.3. Variadic Boolean operations

To evaluate the performance of our approach when evaluating large CSGs with thousands and thousands of faces, we reproduce the experiments of Douze et al. (2015) (see Fig. 25). Some of the variadic test cases are not presented because of the 8GB memory limitation. Since only QuickCSG and our method are variadic, comparisons with the other methods are performed by decomposing CSG into binary Boolean operations. As shown in Table 1, our method is more efficient than other exact methods in most cases. But our approach still has room for acceleration when compared to Quick CSG. The reason for disadvantages in speed when facing QuickCSG is that our approach exhaustively computes all intersections between primitives to pursue good results. Although QuickCSG has advantages in execution, it sometimes generates deficit results even with solid inputs (please refer to supplemental materials). We add an extreme CSG with 801 primitives (the Ring & Ball in Table 1) to evaluate the effect of our label caching acceleration. This example is large and each of its primitive usually intersects with less than 10 other primitives. The evaluation time of *Ring & Ball* was 38.1 s without enabling the label caching mechanism. With label caching acceleration, the computational time is reduced to 20 s. Therefore, by applying label caching, the total computation time drops to only half of the original.

9. Summary

In this paper, we proposed a novel efficient and robust Boolean evaluation approach based on hybrid representation. Our approach is variadic and able to efficiently handle large meshes while robust with solid inputs. To achieve a win–win solution in the trade-off between efficiency and robustness, we creatively embed the P-reps and V-reps into the process of Boolean evaluation simultaneously. The P-reps allow us to conduct exact computation with predicates only and avoid accumulating numerical errors. The V-reps are used for coarse tests and fast neighborhood queries to reduce the amount of slow plane-based computation. The experimental results show that our approach has performance advantages compared to other exact methods, guaranteeing robustness under consistent inputs. One future direction for improving our approach is to optimize the computation of the CSGs that contain a lot of meshes within a small area. In these cases, many computed intersections will not appear as edges in the final mesh, leading to unnecessary tessellation. Optimization is needed to alleviate this problem. Another direction of developing our method is to loosen the input limits of our approach. The inputs of our approach are limited to solids. The input requirements of our method may be extended to Piecewise-constant integer generalized Winding Number (PWN) meshes, that allow self-intersection. We believe that this would be an interesting and valuable extension of our work.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (Nos. 61572316, 61671290), National High-tech R&D Program of China (863 Program) (No. 2015AA015904), the Key Program for International S&T Cooperation Project of China (No. 2016YFE0129500), the Science and Technology Commission of Shanghai Municipality (Nos. 16DZ0501100, 17411952600), the Interdisciplinary Program of Shanghai Jiao Tong University (No. 14JCY10), the Research Grants Council, University Grants Committee (Nos. 28200215, CityU 11237116), City University of Hong Kong (No. 7004915), and ACIM-SCM.

Table 1
Computation time statistics of the evaluations of large CSGs (seconds).

No.	Model	Face Num.	Mesh Num.	CGAL (Hachenberger and Kettner, 2006)	Cork (Bernstein, 2013)	Carve (Sargeant, 2011)	Quick CSG (Douze et al., 2015)	LibiGL (Zhou et al., 2016)	Our approach[†]				
									Total	Step 1	Step 2	Step 3	Step 4
1	Organic	219k	6	–	14.3	63.1	0.580	56.79	2.75	0.892	1.32	0.397	0.118
2	T1	80k	50	1.00k	18.5	10.4	0.388	11.2	14.4	0.691	2.71	8.11	2.87
3	T2	7k	50	2.81k	–	16.0	0.804	14.5	5.52	0.162	1.11	3.29	0.746
4	Sprocket	11k	52	211	–	4.26	(0.132)[*]	3.18	0.386	0.093	0.105	0.149	0.034
5	Ring & Ball	146k	801	–	–	187	(1.10)	166	20.0	1.04	3.55	8.61	6.68

[†] *Steps 1, 2, 3, and 4* are octree construction, intersection computation, tessellation and classification, respectively.
[*] The bracket indicates that although QuickCSG gives the answer, the result meshes are full of topology deficiencies that contain thousands of boundary edges.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cagd.2018.03.021.

References

Banerjee, R.P., Rossignac, J.R., 1996. Topologically exact evaluation of polyhedra defined in CSG with loose primitives. Comput. Graph. Forum 15 (4), 205–217.
Barki, H., Guennebaud, G., Foufou, S., 2015. Exact, robust, and efficient regularized Booleans on general 3D meshes. Comput. Math. Appl. 70 (6), 1235–1254.
Bernstein, G., 2013. Cork Boolean library. https://github.com/gilbo/cork.
Bernstein, G., Fussell, D., 2009. Fast, exact, linear Booleans. Comput. Graph. Forum 28 (5), 1269–1278.
Biermann, H., Kristjansson, D., Zorin, D., 2001. Approximate Boolean operations on free-form solids. In: ACM SIGGRAPH, pp. 185–194.
Campen, M., Kobbelt, L., 2010. Exact and robust (self-) intersections for polygonal meshes. Comput. Graph. Forum 29 (2), 397–406.
Chew, L.P., 1989. Constrained Delaunay triangulations. Algorithmica 4 (1–4), 97–108.
Cignoni, P., 2015. Visualization and computer graphics library. http://vcg.isti.cnr.it/vcglib.
Douze, M., Franco, J.-S., Raffin, B., 2015. QuickCSG: Arbitrary and Faster Boolean Combinations of n Solids. PhD thesis. Inria-Research Centre Grenoble–Rhône-Alpes; INRIA.
Fang, S., Bruderlin, B., Zhu, X., 1993. Robustness in solid modelling: a tolerance-based intuitionistic approach. Comput. Aided Des. 25 (9), 567–576.
Feito, F.R., Ogáyar, C.J., Segura, R.J., Rivero, M., 2013. Fast and accurate evaluation of regularized Boolean operations on triangulated solids. Comput. Aided Des. 45 (3), 705–716.
Fortune, S., 1995. Polyhedral modelling with exact arithmetic. In: Proceedings of ACM Symposium on Solid Modeling and Applications, pp. 225–234.
Fortune, S., 1997. Polyhedral modelling with multiprecision integer arithmetic. Comput. Aided Des. 29 (2), 123–133.
Fortune, S., Van Wyk, C.J., 1993. Efficient exact arithmetic for computational geometry. In: Annual Symposium on Computational Geometry, pp. 163–172.
Frisken, S.F., Perry, R.N., 2002. Simple and efficient traversal methods for quadtrees and octrees. J. Graph. Tools 7 (3), 1–11.
Gottschalk, S., Lin, M.C., Manocha, D., 1996. Obbtree: a hierarchical structure for rapid interference detection. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. ACM, pp. 171–180.
Granados, M., Hachenberger, P., Hert, S., Kettner, L., Mehlhorn, K., Seel, M., 2003. Boolean operations on 3D selective Nef complexes: data structure, algorithms, and implementation. In: ESA, vol. 3. Springer, pp. 654–666.
Hable, J., Rossignac, J., 2005. Blister: GPU-based rendering of Boolean combinations of free-form triangulated shapes. ACM Trans. Graph. 24 (3), 1024–1031.
Hachenberger, P., Kettner, L., 2005. Boolean operations on 3D selective Nef complexes: optimized implementation and experiments. In: Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling. ACM, pp. 163–174.
Hachenberger, P., Kettner, L., 2006. 3D Boolean operations on Nef polyhedra. In: CGAL Editorial Board (Ed.), CGAL User and Reference Manual. CGAL, 3.
Hu, C.-Y., Patrikalakis, N.M., Ye, X., 1996a. Robust interval solid modelling. Part I: representations. Comput. Aided Des. 28 (10), 807–817.
Hu, C.-Y., Patrikalakis, N.M., Ye, X., 1996b. Robust interval solid modelling. Part II: boundary evaluation. Comput. Aided Des. 28 (10), 819–830.
Keyser, J., Culver, T., Foskey, M., Krishnan, S., Manocha, D., 2004. Esolid – a system for exact boundary evaluation. Comput. Aided Des. 36 (2), 175–193.
Keyser, J., Krishnan, S., Manocha, D., 1999a. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. I: representations. Comput. Aided Geom. Des. 16 (9), 841–859.
Keyser, J., Krishnan, S., Manocha, D., 1999b. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. II: computation. Comput. Aided Geom. Des. 16 (9), 861–882.
Kobbelt, L., 2010. WebBSP 0.3 Beta.
Laidlaw, D.H., Trumbore, W.B., Hughes, J.F., 1986. Constructive solid geometry for polyhedral objects. ACM SIGGRAPH Comput. Graph. 20 (4), 161–170.
Lin, Y.-H., Li, Y.-F., Zio, E., 2016. A reliability assessment framework for systems with degradation dependency by combining binary decision diagrams and Monte Carlo simulation. IEEE Trans. Syst. Man Cybern. Syst. 46 (11), 1556–1564.
Möller, T., 1997. A fast triangle–triangle intersection test. J. Graph. Tools 2 (2), 25–30.
Naylor, B., Amanatides, J., Thibault, W., 1990. Merging BSP trees yields polyhedral set operations. ACM SIGGRAPH Comput. Graph. 24 (4), 115–124.
Ogayar, C.J., Feito, F.R., Segura, R.J., Rivero, M., 2006. GPU-based evaluation of Boolean operations on triangulated solids. In: SIACG 2006: Ibero-American Symposium in Computer Graphics.
Ogayar, C.J., Segura, R.J., Feito, F.R., 2005. Point in solid strategies. Comput. Graph. 29 (4), 616–624.
Ogáyar-Anguita, C.J., García-Fernández, Á., Feito-Higueruela, F.R., Segura-Sánchez, R.J., 2015. Deferred boundary evaluation of complex CSG models. Adv. Eng. Softw. 85, 51–60.
Pavić, D., Campen, M., Kobbelt, L., 2010. Hybrid Booleans. Comput. Graph. Forum 29 (1), 75–87.
Preparata, F.P., Shamos, M.I., 1985. Computational Geometry: An Introduction. Technical report. Springer-Verlag, New York, USA.
Priest, D.M., 1991. Algorithms for arbitrary precision floating point arithmetic. In: Proceedings of the IEEE Symposium on Computer Arithmetic. 1991, pp. 132–143.
Requicha, A.A.G., 1977. Mathematical Models of Rigid Solid Objects. Technical report. University of Rochester, Rochester, NY, USA.
Requicha, A.A.G., 1980. Representations for rigid solids: theory, methods, and systems. ACM Comput. Surv. 12 (4), 437–464.
Requicha, A.A., Voelcker, H.B., 1985. Boolean operations in solid modeling: boundary evaluation and merging algorithms. Proc. IEEE 73 (1), 30–44.
Sargeant, T., 2011. Carve CSG Boolean library. https://github.com/VTREEM/Carve.
Segal, M., 1990. Using tolerances to guarantee valid polyhedral modeling results. ACM SIGGRAPH 24 (4), 105–114.
Shewchuk, J.R., 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete Comput. Geom. 18 (3), 305–363.
Shewchuk, J.R., 1999. Lecture notes on geometric robustness. In: Interpolation, Conditioning, and Quality Measures. Eleventh International Meshing Roundtable.
Smith, J.M., Dodgson, N.A., 2007. A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic. Comput. Aided Des. 39 (2), 149–163.
Sugihara, K., Iri, M., 1989. A solid modelling system free from topological inconsistency. J. Inf. Process. 12 (4), 380–393.
Thibault, W.C., 1987. Application of binary space partitioning trees to geometric modeling and ray-tracing. J. Am. Med. Assoc. 184.
Thibault, W.C., Naylor, B.F., 1987. Set operations on polyhedra using binary space partitioning trees. ACM SIGGRAPH Comput. Graph. 21 (4), 153–162.
Tilove, R., Requicha, A.A., 1980. Closure of Boolean operations on geometric entities. Comput. Aided Des. 12 (5), 219–220.
Updegrove, A., Wilson, N.M., Shadden, S.C., 2016. Boolean and smoothing of discrete polygonal surfaces. Adv. Eng. Softw. 95, 16–27.
Varadhan, G., Krishnan, S., Sriram, T., Manocha, D., 2004. Topology preserving surface extraction using adaptive subdivision. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing. ACM, pp. 235–244.
Wang, C.C., 2011. Approximate Boolean operations on large polyhedral solids with partial mesh reconstruction. IEEE Trans. Vis. Comput. Graph. 17 (6), 836–849.
Zhao, H., Wang, C.C., Chen, Y., Jin, X., 2011. Parallel and efficient Boolean on polygonal solids. Vis. Comput. 27 (6), 507–517.
Zhou, Q., Grinspun, E., Zorin, D., Jacobson, A., 2016. Mesh arrangements for solid geometry. ACM Trans. Graph. 35 (4), 39.

Please cite this article in press as: Sheng, B., et al. Accelerated robust Boolean operations based on hybrid representations. Comput. Aided Geom. Des. (2018), https://doi.org/10.1016/j.cagd.2018.03.021