







Faster than Fast: Efficient Approximate Boolean Operations on Dense Triangular Mesh Models

Yuanhui Xiao¹ , Ming Chen² , Shouxin Chen³  and Shenglian Lu⁴ 

¹Guangxi Normal University, xiao20231115@163.com

²Guangxi Normal University, hustcm@hotmail.com

³Guangxi Normal University, chen.csx@outlook.com

⁴Guangxi Normal University, lsl@gxnu.edu.cn

Corresponding author: Ming Chen, hustcm@hotmail.com

Abstract. For the processing of large-scale triangular meshes, the speed of Boolean operations is crucial. This paper presents a new method for performing high-speed Boolean operations on large-scale triangular meshes using the powerful ray tracing performance of the latest OptiX engine. The proposed method transforms the two time-consuming steps of triangle-triangle intersection calculation and inside/outside classification into a ray tracing problem that can be solved with the OptiX engine, and solutions are proposed for degenerate and coplanar conditions. The method is compared to the state-of-the-art QuickCSG, LibIGL, Cork, CGAL Nef, and CGAL Core methods on the Thingi10K dataset. The experimental results show that the proposed method has efficiency and stability advantages for enormous triangular meshes.

Keywords: Boolean operation, ray tracing, intersection calculation, collision detection, Optix engine.

DOI: <https://doi.org/10.14733/cadaps.2025.1007-1026>

1 INTRODUCTION

As a fundamental algorithm of three-dimensional (3D) modeling, 3D Boolean operations are utilized extensively in computer-aided design (CAD)/computer-aided manufacturing (CAM), virtual reality, computer vision, robotics, and other fields. In recent years, with the improvement of industrial manufacturing precision and the development of 3D printing technology, the size of meshes that need to be processed has dramatically increased, posing a challenge to the speed of Boolean operations.

Unlike tree [13],[14],[33] and volumetric representation [18],[21],[36],[38] methods that are designed specifically for parallel computation, OptiX [29], NVIDIA's latest RTX ray tracing engine, performs efficient massively parallel ray intersection testing and creates acceleration structures based on the bounding volume hierarchy (BVH) [16] tree. In this paper, the two steps of calculating the triangle-triangle intersection and inside/outside classification are recast as the

problem of ray and triangle intersection. These two steps are accelerated by the OptiX engine, thereby accelerating the entire Boolean operation.

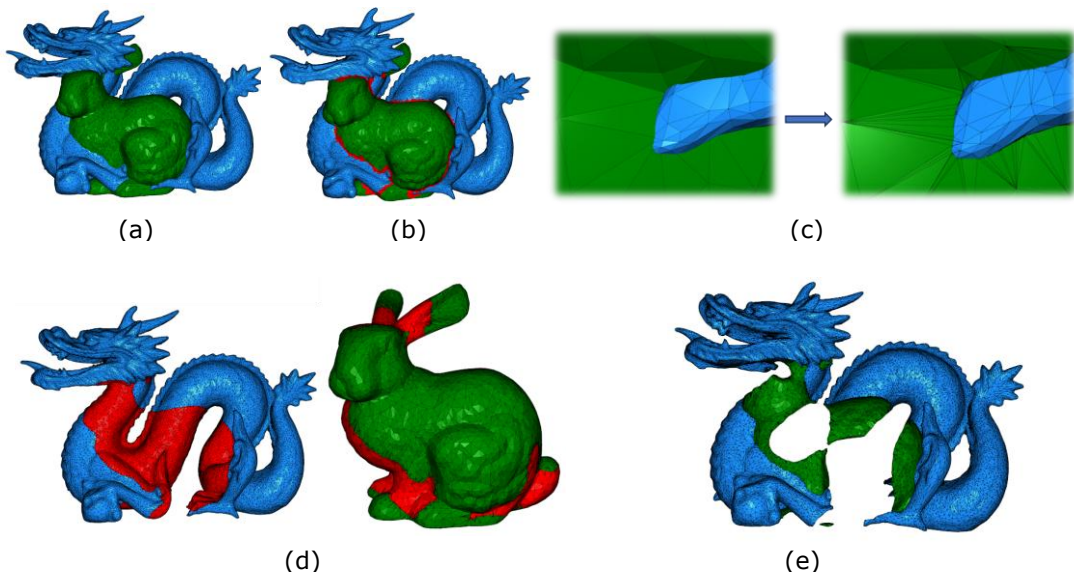


Figure 1: Boolean operation procedure of triangular mesh based on ray tracing. (a) Input two closed, nonself-intersecting 2-manifold models. (b) Use ray tracing to perform the intersection test and compute the intersection points; the red portion of the diagram represents the set of intersecting triangles. (c) Use the CDT algorithm to triangulate the intersecting triangle. (d) Use ray tracing to complete the inside/outside classification; the red portion of the figure is determined to be inside another model. (e) According to Boolean operation rules, the dragon-bunny result is obtained.

The proposed method requires the input triangular meshes to be closed, orientable, nonself-intersecting, and nondegenerate 2-manifolds. Preprocessing is necessary if the aforementioned conditions are not satisfied. This paper's method consists of three main steps (see Fig. 1):

Step 1: Intersection test and intersection calculation (see Fig. 1(b)). Rays were emitted along each side of the triangle, intersection tests were conducted with another model involving Boolean operations, and the intersections were calculated. This step is accomplished using the ray tracing method; the intersection points are calculated using Eq. (2), and degenerate cases are handled.

Step 2: Triangle tessellation (see Fig.1 (c)). Using the constrained Delaunay triangulation (CDT) algorithm [10], the intersection points in each intersecting triangle were connected to constrained line segments as input, and the intersecting triangles were triangulated. Each intersecting triangle was subdivided into multiple subtriangles following segmentation. In this step, abnormal triangles are processed to increase stability. This step is accelerated in parallel using the OpenMP [6] policy and runs on the CPU.

Step 3: Inside/outside classification (see Fig. 1(d)). After triangle tessellation, there are no triangles that are both inside and outside the target model; all triangles are either inside or outside. The final essential step is to classify all triangles as either inside or outside and then to retain the triangles in accordance with the rules of Boolean operations to obtain the final result of Boolean operations(see Fig.1 (e)). In this step, ray tracing is utilized once more to complete the inside/outside classification.

The main contributions of this paper are as follows:

(1) Triangle-triangle intersection detection was transformed into a triangle-ray intersection problem. Using ray tracing, the intersection region and intersection points of the model were determined.

(2) In this paper, the model's topological information was completely disregarded for triangle inside/outside classification. Ray tracing technology was extensively utilized to classify each triangle as belonging to the inside/outside.

The rest of this paper is arranged as follows: Section 2 reviews the literature on Boolean operations; Section 3 describes how the NVIDIA OptiX ray tracing engine works and how it computes the intersection points between rays and triangles. Section 4, 5, and Section 6 provide details of the proposed methods. Section 7 shows experimental and comparative results; Section 8 summarizes the paper.

2 RELATED WORK

The primary research areas for triangular mesh Boolean operations are speed and stability. To accelerate Boolean operations, the two methods of bounding volumes [1],[20] and space partitioning [23] are widely used. The bounding volume method includes the axis-aligned bounding box (AABB) [1],[23],[31], oriented bounding box (OBB) [7],[15],[37], and K-Dop [22]. The primary objective of the preceding method is to construct a bounding box that drastically reduces the number of triangles used for intersection tests. Compared to the bounding volume methods, the advantage of space partitioning methods is that it is easier to implement parallel operations. Space partitioning methods include octrees [8],[14],[30],[33], uniform grids [11],[12], kd-trees [13], binary space partitioning (BSP) trees [3],[27] and so on. Douze et al. [13] used kd-tree acceleration and implemented fast Boolean operations on meshes through parallel computation. Campen and Kobbelt [5] combined octrees and BSP to increase efficiency and decrease storage costs.

Another bottleneck of Boolean operations is the problem of stability, which is also the focus of many scholars. Three factors, namely, self-intersection, primitive degeneration, and rounding errors of floating-point arithmetic, affect the stability and exactness of Boolean operations. A common idea is to describe the model using implicit representations, including vertex-based representations (V-reps) and plane-based representations (P-reps). Compared with V-reps, P-reps have better stability and can obtain exact results even under fixed precision arithmetic. Literature on the V-rep approach includes the work of Douze et al. [13], De Magalha et al. [12], and Zhou et al. [39]. Douze et al. [13] improved performance by sacrificing the stability of the algorithm; they used winding number vectors in combination with a smart pruning strategy to evaluate Booleans quickly. Zhou et al. [39] used the arbitrary precision algorithm to compute all triangle-triangle intersections. The triangle is determined as part of the result by the winding number of the corresponding volumetric cell. De Magalha et al. [12] used large rational numbers to avoid rounding errors and deal with the vertices of one object incident on the face of the other object through simulation of simplicity, further improving stability. Epsilon-tweaking [14] and numerical perturbation [13] are also used to improve stability and achieve quasi-robustness.

The method of P-reps is described in the literature [9],[17],[28],[34],[35]. Hachenberger et al. [17] computed Boolean operations by combining an exact algorithm with the plane-based Nef-polyhedron [28]. Although quite reliable, their method has performance and memory consumption issues. To balance stability and efficiency, Sheng et al. [34] used a hybrid representation method that combined V-reps and P-reps. P Trettner et al. [35] used a plane-based representation for the input meshes along with recently introduced homogeneous integer coordinates to ensure exactness and used a formulation of the algorithm via generalized winding numbers and mesh arrangements to ensure reliability and robustness. Cherchi et al. [9] used an improved mesh arrangement method and an internal and external classification system based on exact ray projection to ensure reliability and robustness. However, when used to large-scale triangular meshes, which usually have more than 200k triangles, it still shows limitations.

3 EVALUATION AND CALCULATION OF INTERSECTIONS

The proposed method is implemented based on the NVIDIA OptiX ray tracing engine [29]. This section briefly introduces how to calculate the intersection of a ray and a triangle using its properties.

3.1 Ray-Triangle Intersection

The proposed method requires many ray-triangle intersection detections. In this paper, the emitted ray $\mathbf{R}(t)$ can be determined by the origin O and a normalized direction \mathbf{D} :

$$\mathbf{R}(t) = O + t\mathbf{D} \quad (3.1)$$

Using ray tracing, it is possible to determine if $\mathbf{R}(t)$ intersects a target triangle. A method similar to the Moller-Trumbore algorithm [26] is used to calculate the intersection points: a triangle is defined by three vertices V_0 , V_1 and V_2 , and a point, $T(u,v)$, on a triangle is given by

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2 \quad (3.2)$$

where (u,v) are the barycentric coordinates, which must fulfill $u \geq 0$, $v \geq 0$ and $u+v \leq 1$. Computing the intersection between the ray, $\mathbf{R}(t)$, and the triangle, $T(u,v)$, is equivalent to setting $\mathbf{R}(t) = T(u,v)$; then, rearranging the terms gives:

$$\begin{pmatrix} -\mathbf{D} & V_1 - V_0 & V_2 - V_0 \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = O - V_0 \quad (3.3)$$

By solving the linear system of equations above, the barycentric coordinates (u,v) and the distance t from the ray origin to the intersection point can be found.

We set the tracing range $[t_{min}, t_{max}]$ when utilizing ray tracing for intersection calculation or detection (see Fig. 2). If the distance t of the intersection points determined by tracing falls within this range, the objects are deemed to be intersecting; otherwise, they are deemed to be nonintersecting.

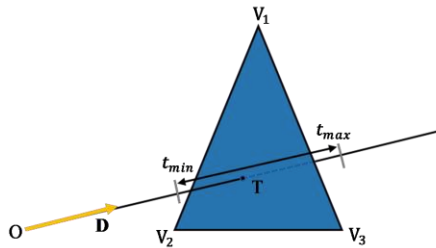


Figure 2: The ray intersects the triangle at point T ; $[t_{min}, t_{max}]$ are preset tracing ranges.

4 INTERSECTION COMPUTATION

Before calculating the intersection point, it is typically necessary to make an intersection determination: we must determine which pairs of triangles may intersect, and then begin the calculation of the intersection point to save time. Object or space hierarchy algorithms, such as BVH[25] and octree[32], are frequently used to search for rays intersecting triangles. To increase efficiency, this paper uses the specialized ray tracing acceleration structures of the OptiX ray tracing engine to complete the intersection tests, which are performed entirely on the GPU.

4.1 Intersection Test

$\{M_i\}_{i \in \{1,2\}} = (V_j, T_j)$ denotes the meshes engaged in Boolean operations, where V_j is the set of vertices and T_j is the set of triangles. M_i is a closed, orientable, nonself-intersecting and nondegenerate 2-manifold. When M_i is input, the topological connection information of its points, edges, and faces is constructed by traversal. This paper transforms the triangle intersection test between M_1 and M_2 into a ray and triangle intersection test by emitting rays along each edge of every triangle in T_j . As shown in Fig. 3, a ray (yellow arrow) is emitted along each edge of the triangle. Using the emission ray \mathbf{R}_1 along edge V_1V_2 of triangle T_1 as an illustration, the parameters are constructed as follows:

- Set vertex V_1 as the origin O of the ray.
- Set the normalized direction of vector V_1V_2 as the ray direction \mathbf{D} .
- Set $t_{min} = 0$, $t_{max} = |V_1V_2|$, and $|V_1V_2|$ as the distance between V_1 vertices V_2 .

In the OptiX ray tracing engine, any hit program is invoked when the acceleration traversal finds that a ray intersects a primitive (triangle). In the any hit program, the barycentre coordinates (u, v) of the intersection point and the index of the intersected triangle can be obtained by invoking the API; the intersection point P_i is then computed using equation 3.2. The intersection calculated using equation 3.2 moves along the surface due to rounding error. To improve the accuracy of the intersection P_i , the intersection P_i is projected onto the original ray. The set E_i stores the index IDs of the two adjacent triangles that the ray shares, and the set H_i stores the index ID of the triangle that the ray hits. The relationship between the two pieces of information and the intersection P_i is recorded as $\{E_i, H_i, P_i\}$ and utilized in the next step of triangle tessellation.

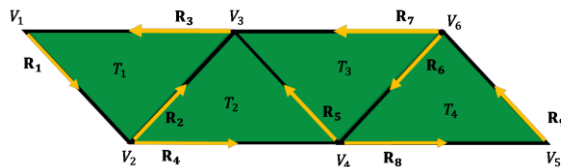


Figure 3: Rays are emitted along the edges of M_i , and ray tracing is performed to complete the intersection test and calculation.

4.2 Handling Degenerate Situations

Most of the intersection cases belong to the two cases listed in Fig. 4, for which the correct intersection points can be calculated through the method of Section 4.1. Due to the accumulation of numerical error in the single-precision floating-point calculations of the OptiX engine and the characteristics of the OptiX engine itself, in some cases, it is not possible to stably obtain the intersection points. There are two cases requiring special handling: two triangles intersecting at a point or edge and coplanar triangles.

Due to the high efficiency of the OptiX engine, this paper performs no filtering and assumes that the three conditions listed above are possible on all edges. This paper addresses the above two cases by emitting more rays. The specific methods are as follows:

Intersection at a point or on an edge: This includes the case of two triangles intersecting at a point (see Fig. 5(a)) or on an edge (see Fig. 5(b)). As shown in Fig. 5(a), the rays \mathbf{R}_1 emitted

along edge V_1V_2 and rays \mathbf{R}_2 emitted along edge V_2V_3 intersect triangle T' at the same points $P(V_2)$ (points P and V_2 share the same location); point P is the end of ray \mathbf{R}_1 and the origin of ray \mathbf{R}_2 . In this case, the intersection occurs precisely at the boundary values of the trace range $t_{min} = 0$ and $t_{max} = |V_1V_2|$. Due to the accumulation of numerical error, OptiX occasionally fails to obtain the intersection point. This paper applies a small perturbation 10^{-6} to the origin and end points of each edge before and after tracing and then emits an additional ray to detect the presence of an intersection.



Figure 4: The two most common triangle-triangle intersection cases. The triangles T' and T belong to two input models. (a) The rays \mathbf{R}_1' and \mathbf{R}_3' emitted from T' intersect triangle T at points P_1 and P_2 , respectively. (b) The ray \mathbf{R}_1 emitted from triangle T' and the ray \mathbf{R}_3 emitted from triangle T intersect at points P_1 and P_2 , respectively.

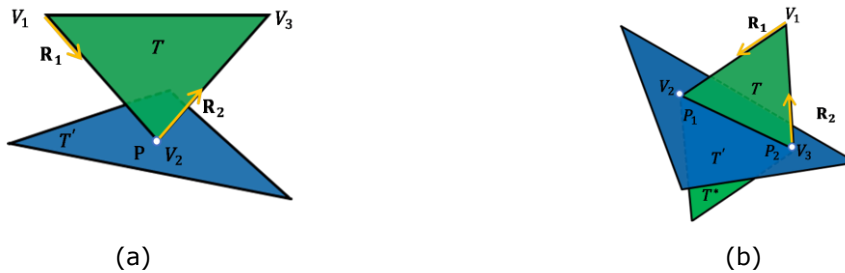


Figure 5: Two triangles intersect at a point or at an edge. (a) The rays \mathbf{R}_1 and \mathbf{R}_2 emitted from triangle T intersect triangle T' at point $P(V_2)$. (b) The rays \mathbf{R}_1 and \mathbf{R}_2 emitted from triangle T intersect triangle T' at points $P_1(V_2)$ and $P_2(V_3)$, respectively, where T^* is a triangle adjacent to T .

Coplanar: Coplanar refers to the case in which two triangles intersect in the same plane. The OptiX engine directly determines that the emitted rays do not intersect for coplanar triangles. Notice that coplanar triangles are divided into two regions (T and T' in Fig. 6(b)): a coplanar region $T' \cap T$ and noncoplanar region $(T' - T) \cup (T - T')$. The edges of the coplanar triangle clip the coplanar region and the noncoplanar region (red dashed box in Fig. 6(a)), so finding the intersection point of the coplanar triangle at the boundary of the two areas is crucial for separating the coplanar region from the noncoplanar region. Since the input mesh is closed and is a 2-manifold, the triangles at the edges of each coplanar region must necessarily intersect adjacent triangles. As shown in Fig. 6(a), if triangle T' is coplanar with triangle T , then triangle T' must

intersect triangle T^* , which is adjacent to triangle T . At the boundary of the coplanar region, the ray R_1 emitted along edge V_2V_3 must intersect with T^* , and the intersection point P between the coplanar and noncoplanar regions can be obtained.

Numerous experiments demonstrate that for the intersection points at the boundaries of coplanar regions, rays emitted in one direction of an edge may not intersect, but rays emitted in the opposite direction must intersect. In this paper, after ray tracing is completed along each edge, ray tracing is carried out again in the opposite direction. If an intersection occurs, the index is compared with the triangle index hit during the first ray tracing to determine whether it is a missing intersection. Since one side of a triangle can only intersect another triangle once and the index of each triangle is unique, it is guaranteed that no duplicate intersections are introduced.



Figure 6: (a) The triangle T' is coplanar with T , and T^* is an adjoint triangle of T and is not coplanar with T' and T . At the boundary of the coplanar region, the ray R_1 emitted along the edge V_2V_3 intersects with T^* at point P . (b) The coplanar region $T' \cap T$ and noncoplanar region $(T' - T) \cup (T - T')$.

5 TRIANGLE TESSELLATION

After the intersection calculation, the intersection points in each intersecting triangle are known, and then the intersection points are connected into line segments as constraints (yellow line segments in Fig. 7(b)). This information is input into the CDT [10] algorithm to complete the triangulation in 2D (see Fig. 7(c)). To increase the speed of this operation, this paper uses OpenMP [6] to implement the CDT algorithm in parallel, which is implemented on the CPU.

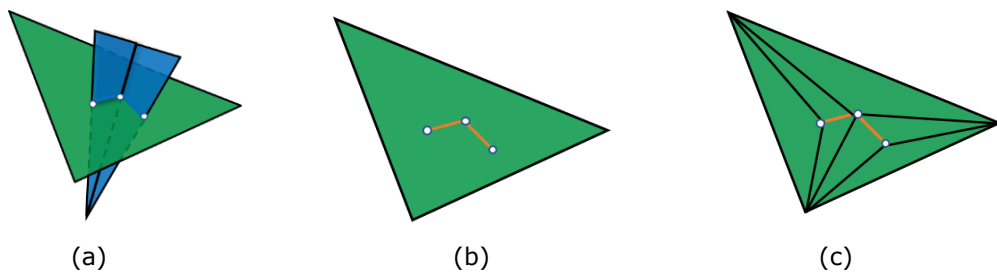


Figure 7: An overview of triangle tessellation. (a) Compute the intersection to obtain the intersection points. (b) Connecting intersecting points to line segments as constraints. (c) Using the CDT algorithm to perform triangulation in 2D.

5.1 Constrained Line Segment

Before using the CDT algorithm to conduct triangulation of intersecting triangles, we need to connect the intersection points as constraint segments. Comparing the ID sets E_i of the emitting ray triangle index and H_i of the intersecting triangle index connects the two intersection points. Assume the two points $P = \{E_i, H_i, P_i\}$ and $Q = \{E'_i, H'_i, Q'_i\}$, where P_i and Q'_i are the coordinates of the two points. $\exists T_0 \in E_i$, $\exists T_1 \in H_i$, $\exists T'_0 \in E'_i$ and $\exists T'_1 \in H'_i$. When the following two conditions are satisfied by the four triangle indices T_0 , T_1 , T'_0 and T'_1 , the points P and Q are joined as a constrained line segment.

- If $T_0 = T'_0$ and $T_1 = T'_1$, then connect points P and Q . As shown in Fig. 4(a), P_1P_2 is a constrained line segment.
- If $T_0 = T'_1$ and $T_1 = T'_0$, then connect points P and Q . As shown in Fig. 4(b), P_1P_2 is a constrained line segment.

When a ray intersects multiple triangles at one point (see Fig. 8), the Optix engine can return only one index of intersecting triangles, and the constrained line segment will be missing. In this case, the intersection points are identified by evaluating the barycentric coordinate components $(u, v, 1 - u - v)$, and then the topological information of the vertices, edges, and faces is queried to determine the index ID of the triangle corresponding to the intersection points. If one of the components of the barycentric coordinates is 1, the intersection is on the vertex (see Fig. 8(a)), and if one of the components of the barycentric coordinates is 0, the intersection is on one edge (see fig. 8(b)).



Figure 8: The rays intersect with multiple triangles at one point. (a) The intersection point is on one vertex. (b) The intersection point is on one edge.

5.2 Triangulation

In this step, the triangle is triangulated using the CDT algorithm in the third-party library Fade2D [24] in 2D. To ensure reliability and robustness, we projected the vertex and intersection segment of the intersecting triangle onto the axis alignment plane with the largest area and then triangulated it. Assume that the normalized normal vector of the triangle is \mathbf{n} and that the normalized direction vectors of the X-, Y-, and Z-axes are \mathbf{x} , \mathbf{y} , and \mathbf{z} , respectively. The cosine of the projection angle is computed using the formula $\cos\theta = \max\{|\mathbf{n} \cdot \mathbf{x}|, |\mathbf{n} \cdot \mathbf{y}|, |\mathbf{n} \cdot \mathbf{z}|\}$, and the plane with the greatest $\cos\theta$ is chosen as the projection plane. After projection, the vertex order of the triangle may change, which must be reverted.

Due to the influence of the error of the calculation accuracy on the intersection points, the intersection points that should be on the edge of the triangle are not on the edge lines, resulting in incorrect triangulation results. As shown in Fig. 9, when the intersection points are not on the

edges of the triangle, the triangulation produces a triangle with an area close to 0. We provide a real-world illustration of this situation (see Fig. 10). When the point of intersection is very close to the boundary line, the following measures are taken:

- The two vertices of each edge and the intersection points obtained by tracing along this edge are put into the point set P_i . If all three vertices of a triangle in the triangulation result belong to P_i , the triangle is eliminated and reconstructed locally. In ray tracing, the intersection points can be obtained consecutively along the direction of tracing, making P_i simple to construct.
- If the distance d between two points in an intersecting triangle is less than 10^{-6} , the points are merged into one.

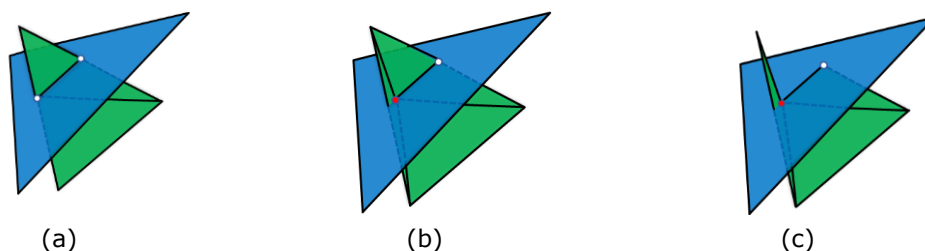


Figure 9: The error in calculating the intersection points results in an abnormal triangle whose area after triangulation is close to 0. (a) No calculation error exists, and the correct triangulation result is obtained. (b) The calculation error of the intersection point, which should be on the boundary line, is shifted to the red point, and after triangulation, a triangle with a near-zero area is generated. (c) Intersecting triangles cannot be triangulated correctly, resulting in incorrect results.



Figure 10: A real-world illustration. (a) Due to calculation errors, the result is a lot of tiny triangles. (b) After taking measures, the right result is obtained.

6 INSIDE/OUTSIDE CLASSIFICATION

After triangle tessellation, each triangle is either completely inside or completely outside of the model. Zhou et al. [39] and P Trettner et al. [35] used the winding number to evaluate the model arrangement and inside/outside classification. We utilize the high efficiency of OptiX intersection judgement and adopt a simple method to complete inside/outside classification: A ray is emitted from the centre of gravity G of each triangle towards the centre C of the model, and then the inside or outside the triangle is determined by the sign of $\cos\theta$, where θ is the angle between the direction vector D of the ray and the normal vector n of the nearest intersecting triangle (see Fig.11). As ray tracing is performed very quickly on the GPU, the entire procedure is completed in a short amount of time. The specific steps are as follows:

- Assuming the intersecting models are M_1 and M_2 , a slight shift from each triangular centre of gravity G of M_1 along its own normal direction (the translation is to prevent coplanar cases) is performed, and then G is used as the ray's origin O .
- GC is the direction vector of the ray, and C is the centre of model M_2 .
- Set $t_{min} = 0$ and t_{max} to be the length of the model M_1 bounding box.

Since the input model is closed, the ray must hit the triangle of model M_2 . In addition to the case where the rays are coplanar with the triangle, there are two cases where the rays do not intersect: one where the centre of the model is empty (see Fig. 12) and the other where the two models do not intersect. In both cases, the triangle is outside the model M_2 . To reduce the coplanar probability, a ray is emitted in the opposite direction when the rays do not intersect. The inside/outside classification rules are as follows:

- If rays intersect and $\cos\theta > 0$, it is determined that the triangle lies inside model M_2 ; otherwise, it is determined that the triangle lies outside model M_2 .
- If the rays do not intersect, another ray is emitted along the opposite direction. If both rays do not intersect, then the triangle is determined to be outside.

After the above steps, all triangles in models M_1 and M_2 are classified as belonging to the inside/outside. The final Boolean operation result is obtained by applying the reservation removal rules(see Table 1).

<i>Operation type</i>	<i>Kept triangles for M_1</i>	<i>Kept triangles for M_2</i>
$M_1 \cap M_2$	Inside M_2	Inside M_1
$M_1 \cup M_2$	Outside M_2	Outside M_1
$M_1 - M_2$	Outside M_2	Inside M_1
$M_2 - M_1$	Inside M_2	Outside M_1

Table 1: The rules of keeping triangles for M_1 and M_2 .

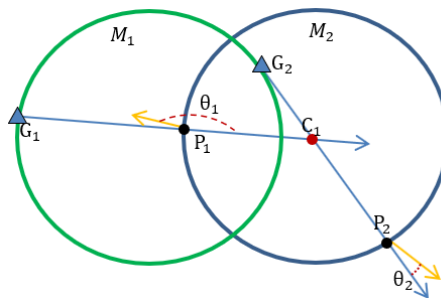


Figure 11: M_1 and M_2 represent two models. The normal vectors (yellow arrows) of triangles in model M_2 are directed outwards, and C_1 is the centre of model M_2 . The ray emitted by the triangle of the barycentre G_1 intersects M_2 at point P_1 , and $\cos\theta_1 < 0$, so the triangle is judged to be outside M_2 . The ray emitted by the triangle of the barycentre G_2 intersects M_2 at point P_2 , and $\cos\theta_2 > 0$, so the triangle is judged to be inside M_2 .

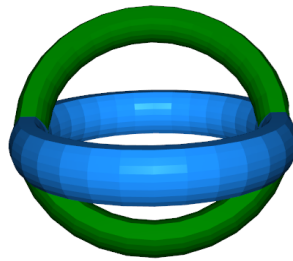


Figure 12: The centres of the two models are empty, and rays emitted toward the model centres may not intersect.

7 EXPERIMENTS AND RESULTS

The proposed algorithm is implemented in C++, and the test PC is configured with an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz x 2 and an NVIDIA Ge-Force RTX 3090. First, we compare the running times, and the comparison methods include CGAL Nef Polyhedra [4], CGAL Corefine [4], Cork [2], QuickCSG [13] and Zhou's method [39] in LibIGL [19]. For a more thorough evaluation of the performance of the proposed method, a self-union comparison was conducted on the Thingi10K [40] dataset with QuickCSG, the quickest Boolean operation algorithm currently available. Second, the Hausdorff distance was used to validate the method's precision and robustness. Methods used for this comparison include CGAL Nef Polyhedra [4], QuickCSG[13], and Zhou's method [39] in LibIGL [19].

7.1 Comparison with State-of-the-Art Methods

First, six examples were used to compare the proposed algorithm with CGAL Nef Polyhedra [4], CGAL Corefine [4], Cork [2], QuickCSG [13], and Zhou's method [39] in LibIGL [19]. The number of input triangles ranges from approximately 10k to 1.6 M, and the models come from the Thingi10K dataset. The running results of the union, intersection and difference sets of the six experiments are shown in Table 2 and Fig. 17. Table 2 demonstrates that the proposed methods are quicker than all other methods except for Example 1. Example 1 demonstrates that when the number of triangles in the input model is approximately 10k, the operating time of the proposed method is comparable to that of the QuickCSG method. The reason for this is that when the number of input triangles is minimal, the proposed method requires a significant amount of time to construct a ray tracing pipeline, resulting in a slower overall time than QuickCSG. In the remaining examples, the method is faster than QuickCSG. This is because the proposed method utilizes the GPU to perform intersection computations and face classification, which significantly accelerates these two steps. As the number of input triangles increases, the advantages become increasingly apparent.

In addition to the number of triangles in the input model, the number of intersecting triangles also affects the runtime. The effect of the number of intersecting triangles on the runtime of the proposed method was evaluated in three examples and compared to QuickCSG. Three self-union experiments were conducted by slightly rotating the model three times so that the number of input triangles remains constant, and the number of intersecting triangles increases. The number of input triangles for the three examples was 300k, 2.17 M, and 2.20 M, respectively, and the input models were derived from the Thingi10K dataset and the QuickCSG project homepage. The running results of the union, intersection, and difference sets of the three cases are shown in Fig. 18 and Table 3. The experimental results indicate that the number of triangles in the input model remains constant. Both the proposed method and the QuickCSG method slow as the number of

intersecting triangles increases, but the proposed method takes less time overall. The time spent in each stage is shown in Table 3. The proposed method's time evidently increased primarily at the tessellation stage with the increase in the number of intersecting triangles. Due to the advantage of GPU high-speed parallel computing, the time-consuming changes in intersection computation and inside/outside classification steps are not obvious.

Example	Model [*]	Face Num [!]	Intersect Face Num	Ours	QuickCSG	LibIGL	Cork	CGAL Nef	CGAL Core
1	74890 \cup 104738	15k	0.4K	0.02	0.02	3.51	1.10	26.15	1.18
2	45090 \cup 46780	38k	1K	0.04	0.07	25.52	2.36	73.93	2.87
3	57937 \cup 441711	147k	2K	0.05	0.22	131.8	9.23	571.6	18.23
4	461109 \cup 1063863	324k	4.3K	0.07	0.46	69.905	24.9	4331.5	25.19
5	99269 \cap 87687	630k	15.6K	0.16	1.19	147.11	48.17	-	59.88
6	461112 \cup 461115	1.6M	29.1K	0.25	2.81	368.46	90.40	-	118.7

- The process is terminated after running for a long time.

* $A_1 \cup A_2$, A_1 and A_2 are the numbers of the models in the Thingi10K dataset.

! The face num is the number of input triangles.

Table 2: Computation time statistics (seconds).

Model	Tests	Face Num [!]	Intersect Face Num	QuickCSG	Ours [*]				
					Total	Step1	Step2	Step3	Step4
Armadillo \cup Armadillo	1	300k	14.7k	0.652	0.127	0.031	0.004	0.092	0.0004
	2		18.1k	0.742	0.184	0.031	0.004	0.149	0.0004
	3		24.8k	0.863	0.213	0.031	0.004	0.178	0.0004
Buddha \cup Buddha	1	2.17M	73.7k	3.604	0.503	0.079	0.007	0.416	0.001
	2		83.1k	3.941	0.551	0.078	0.008	0.464	0.001
	3		99k	4.610	0.617	0.082	0.009	0.525	0.001
815486 \cup 815486	1	2.20M	68.3k	2.857	0.522	0.084	0.007	0.430	0.001
	2		129.9k	3.576	0.806	0.080	0.009	0.716	0.001
	3		193.9k	5.028	1.095	0.083	0.009	1.001	0.002

* Steps 1, 2, 3, and 4 are building ray-tracing pipe, intersection computation, triangle tessellation and in/out classification, respectively.

! The face num is the number of input triangles.

Table 3: Computation time statistics (seconds).

7.2 Self-Union on Thingi10K Dataset

For a more comprehensive evaluation of the performance of the proposed method, a self-union experiment was conducted on the Thingi10K dataset [40]. The Thingi10K dataset contains

9956 .stl files in total. The proposed method necessitates that the input model be a triangular solid mesh without self-intersection that is a closed 2-manifold; 5050 out of 9956 models satisfy this aforementioned criterion. This study evaluates each model by rotating it so that it self-intersects. As demonstrated in Section 7.1, only the QuickCSG method is comparable to the proposed method in terms of performance, so this section only compares the self-union operation to the QuickCSG method. The number of input triangles is used to number the 5050 models in ascending order. The models were separated into three categories based on the number of input triangles: fewer than 10k, 10k to 50k, and greater than 50k. There were 3,237, 1,144, and 669 models in these ranges, respectively. The average number of input triangles in the input models for the three groups was 2.8k, 20.3k, and 264k, respectively. The running times of the comparison experiments are shown in Fig.13. In the comparison experiment with fewer than 10k input triangles, the mean value of the proposed method was slower than that of QuickCSG. In comparison experiments with the number of input triangles ranging from 10k to 50k, the mean value of the proposed method was slightly faster than that of the QuickCSG method. In comparison experiments with greater than 50k input triangles, the mean time of the proposed method was approximately five times faster than that of QuickCSG, especially in the second half of the line chart, when the number of triangles was greater than 1 M and the maximum number was approximately 5 M.

Fig.14 demonstrates that when the number of input triangles is small, the building ray-tracing pipe phase becomes the bottleneck of the proposed method. As the number of input triangles increases, the tessellation stage becomes the bottle-neck of the proposed method. Because the intersection computation and in-side/outside classification phases are executed on the GPU, the elapsed time is minimal.

7.3 Correctness and Robustness

To verify the correctness and robustness of the proposed algorithm, the Hausdorff distance between the proposed algorithm's results and the exact method's results was calculated. We selected 2000 models from the Thingi10K dataset with 1k to 10k input triangles. To ensure the reliability of the experiment, the results of the CGAL Nef Polyhedra [4] exact method were taken as the true value, and the proposed method was compared with QuickCSG [13] and Zhou's method [39] in LibIGL [19]. Among the comparison methods, CGAL Nef Polyhedra and Zhou's method in LibIGL are exact methods, and the proposed method and QuickCSG method are inexact methods. The Hausdorff distance distribution of 2000 test cases is shown in Fig.15. Since CGAL Nef Polyhedra and Zhou's method in LibIGL are both exact methods, their results are nearly identical. The QuickCSG method has a 57.5% distribution in the interval (1E-12, 1E-5), and the proposed method has a 93.8% distribution in the interval (1E-8, 1E-5). Although the QuickCSG method is more accurate than the proposed method, the proposed method is more reliable and robust. Based on the distribution of the interval (1E-5, 1E-2), the proposed method produces some incorrect results due to its imprecise calculation and the sensitive threshold. Compared to the QuickCSG method, the proposed method makes significant improvements.

To further demonstrate that the proposed method can deal with coplanar conditions, two more complex coplanar cases are tested, and the results are compared with those of Zhou's method in LibIGL. As shown in Fig.16, the red box in the input contains coplanar areas (i.e., green and blue frequently alternate), with up to three coplanar areas per case. Compared with Zhou's method in LibIGL, the precise method in the coplanar region shows that the output results of union and intersection are consistent.

8 CONCLUSIONS

This paper proposes a new, straightforward and effective method for Boolean operations on large-scale triangular meshes. The main idea is to convert intersection computation and inside/outside classification, two typical steps in Boolean operations, into a ray and triangle intersection problem,

which is solved by the GPU using the NVIDIA OptiX ray tracing engine. The performance of these two steps is significantly enhanced, and a feasible solution for degeneracy is proposed.

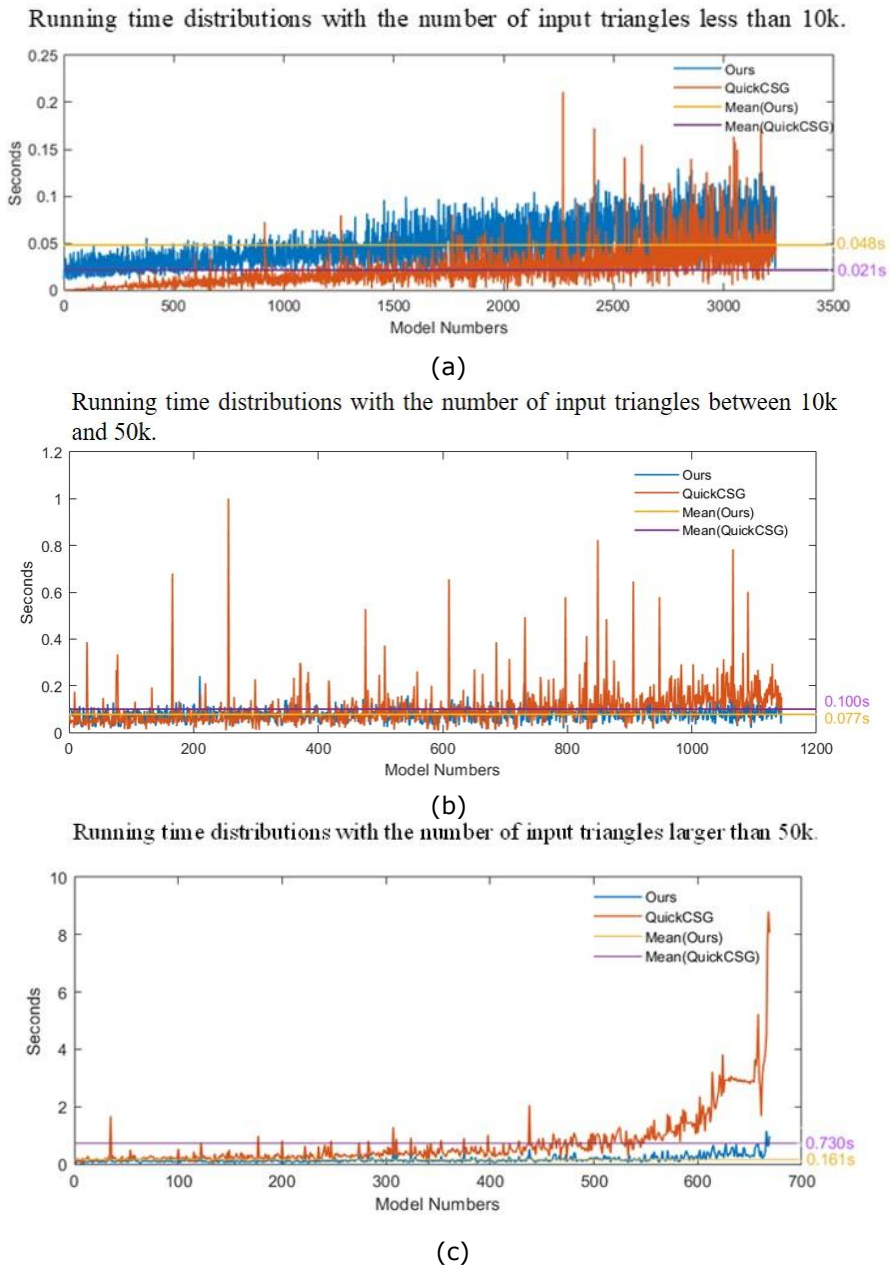


Figure 13: Three groups of self-union comparison experiments, (a), (b), and (c), were conducted between the proposed method and the QuickCSG method, with the number of input triangles ranging from fewer than 10k to between 10k and 50k and greater than 50k, respectively. Models are numbered by the number of input triangles in ascending order. The proposed method in the

first group of experiments is slower than the QuickCSG method, while the proposed method in the second and third groups of experiments is faster. As the number of input triangles increases, the proposed method's performance advantage becomes increasingly apparent.

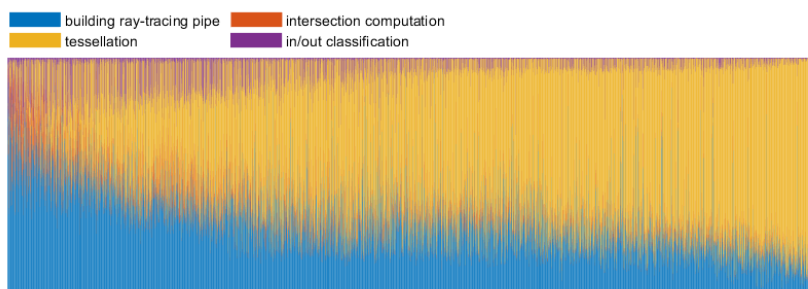


Figure 14: The time percentages of different stages are represented in different colours. Each one-pixel column represents a test. The main bottleneck of our algorithm is triangle tessellation.

The results show that the proposed method is more efficient and reliable than QuickCSG, which is also an inexact method for large-scale triangular meshes. This method can be applied to applications that do not require accurate Boolean operations but require real-time results, and the output results can also be used as the initial value and reference value of accurate Boolean operations to improve their speed. One future improvement is to ensure accuracy and robustness by building a robust ray-tracing engine. Improving the performance of the triangulation phase is another goal.

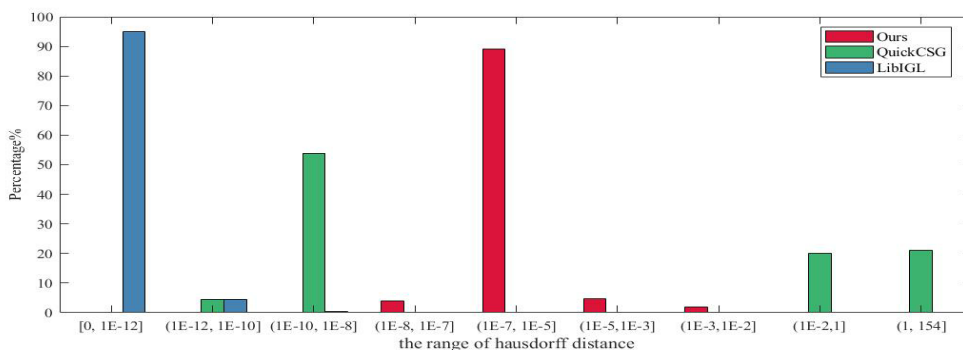


Figure 15: The Hausdorff distance distribution for 2000 test cases. The results of the proposed method are mainly distributed in the interval $(1E-7, 1E-5]$. The results of the QuickCSG method are mainly distributed in the interval $(1E-10, 1E-8]$. The results of the LibIGL method are mainly distributed in the interval $(0, 1E-12]$.

REFERENCES

- [1] Bergen, G.v.d.: Efficient collision detection of complex deformable models using aabb trees. Journal of graphics tools, 2(4), 1–13, 1997. <http://doi.org/10.1080/10867651.1997.10487480>
- [2] Bernstein, G.: Cork Boolean library, <https://github.com/gilbo/cork> 2013.

- [3] Bernstein, G.; Fussell, D.: Fast, exact, linear booleans. In Computer Graphics Forum, vol. 28, 1269–1278. Wiley Online Library, 2009. <http://doi.org/10.1111/j.1467-8659.2009.01504.x>
- [4] Board, C.: Cgal, computational geometry algorithms library, <http://www.cgal.org>.

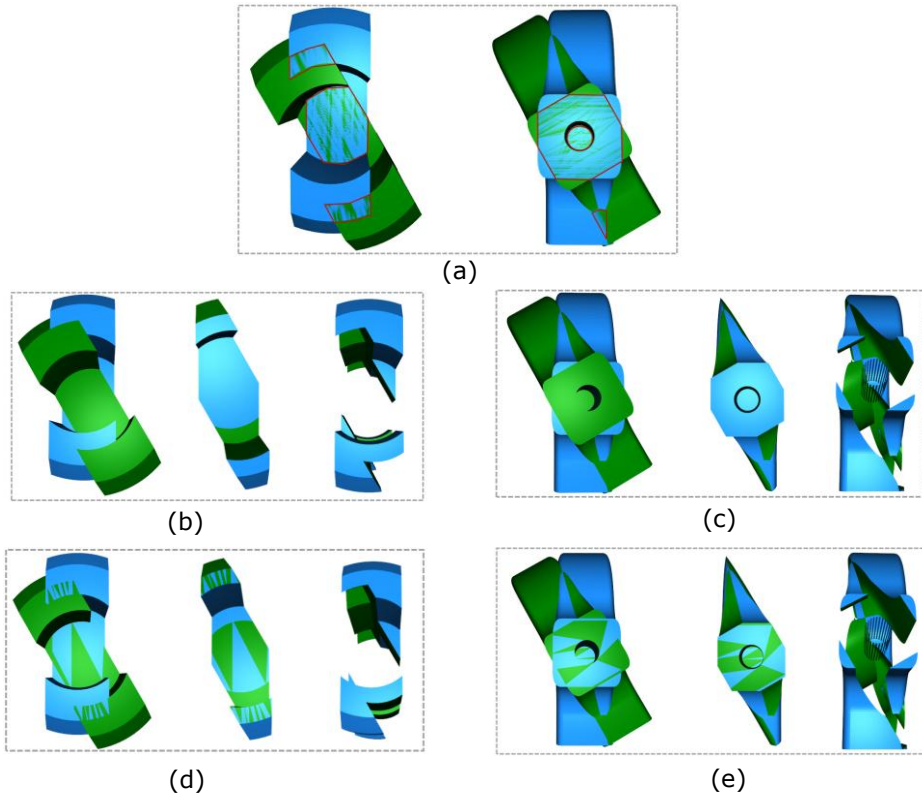
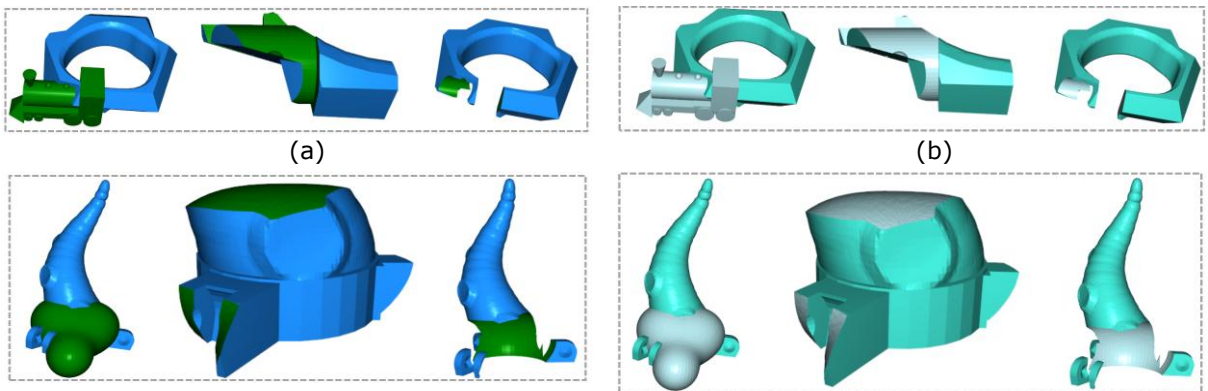


Figure 16: (a) are two inputs for complex coplanar cases. The red boxes are coplanar areas. (b) and (c) are the outputs of the proposed method, which are the union, intersection, and difference sets. In the coplanar region, the proposed method determines the coplanar region of the green model to be outside and the coplanar region of the blue model to be inside. (d) and (e) are the outputs of Zhou's method in LibIGL, which are the union, intersection, and difference sets. In the coplanar region, the result of Zhou's method in LibIGL has both green and blue.



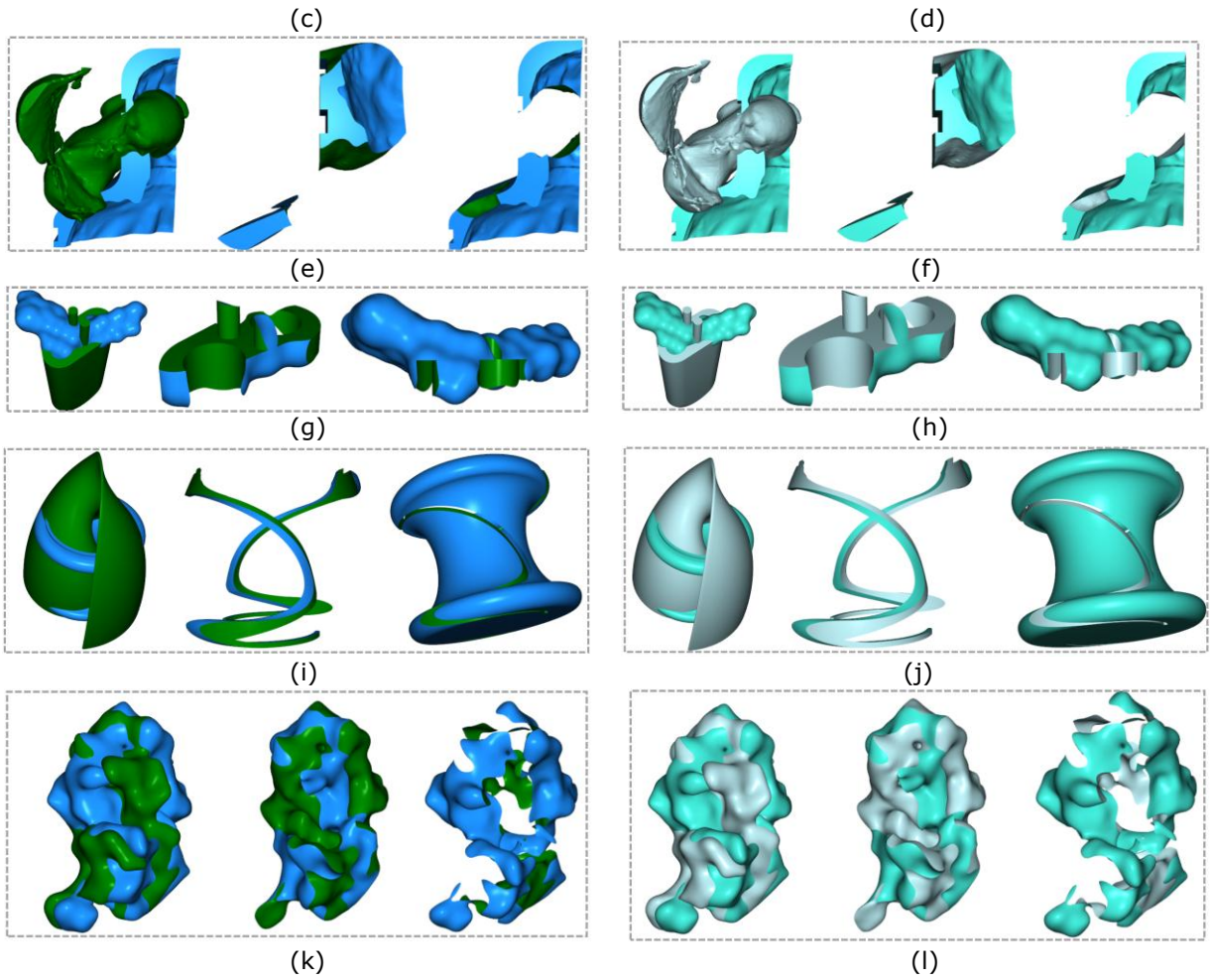
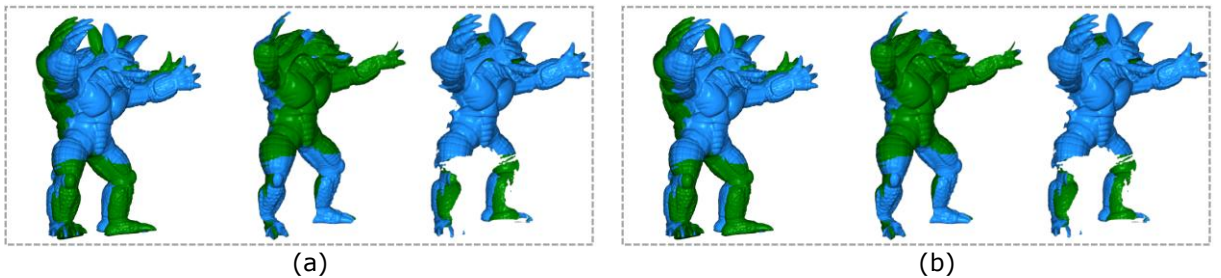


Figure 17: The outputs of the six examples in Table 1 are the union, intersection, and difference sets. The model is from the Thingi10K Dataset. (a), (c), (e), (g), (i) and (k) are the outputs of the proposed method. (b), (d), (f), (h), (j) and (l) are the outputs of Zhou's method in LibIGL.



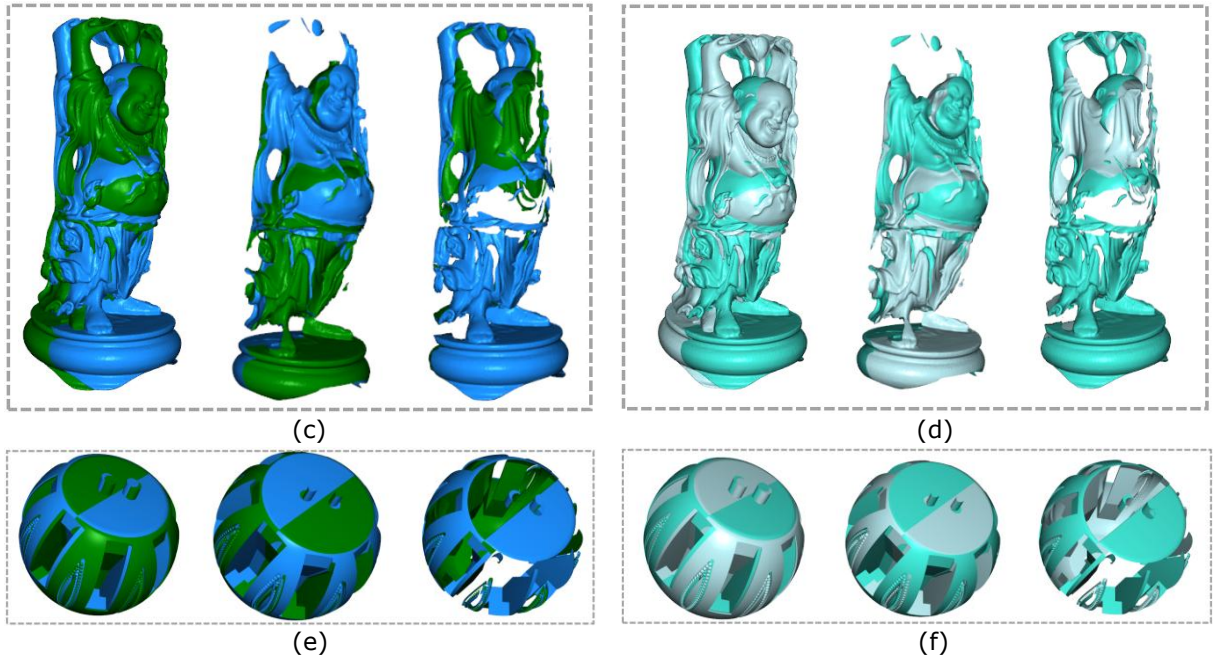


Figure 18: The outputs of the three examples in Table 2 are the union, intersection, and difference sets. The model is from the Thingi10K Dataset and QuickCSG project homepages. (a), (c) and (e) are the outputs of the proposed method. (b), (d) and (f) are the outputs of Zhou's method in LibIGL.

- [5] Campen, M.; Kobbelt, L.: Exact and robust (self-) intersections for polygonal meshes. In *Computer Graphics Forum*, vol. 29, 397–406. Wiley Online Library, 2010. <http://doi.org/10.1111/j.1467-8659.2009.01609.x>
- [6] Chandra, R.: *Parallel programming in OpenMP*. Morgan kaufmann, 2001.
- [7] Chang, J.W.; Wang, W.; Kim, M.S.: Efficient collision detection using a dual obb-sphere bounding volume hierarchy. *Computer-Aided Design*, 42(1), 50–57, 2010. <http://doi.org/10.1016/j.cad.2009.04.010>.
- [8] Chen, Y.; Wang, C.C.: Layer depth-normal images for complex geometries: Part one-accurate modeling and adaptive sampling. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 43277, 717–728, 2008. <http://doi.org/10.1115/DETC2008-49432>.
- [9] Cherchi, G.; Pellacini, F.; Attene, M.; Livesu, M.: Interactive and robust mesh booleans. *ACM Transactions on Graphics (TOG)*, 41(6), 1–14, 2022. <http://doi.org/10.1145/3550454.355546>.
- [10] Chew, L.P.: Constrained delaunay triangulations. In *Proceedings of the third annual symposium on Computational geometry*, 215–222, 1987.
- [11] de Magalhães, S.V.; Andrade, M.V.; Franklin, W.R.; Li, W.; Gruppi, M.G.: Exact intersection of 3d geometric models. In *Geoinfo 2016, XVII Brazilian Symposium on GeoInformatics*, 2016. <http://doi.org/10.1111/j.1467-8659.2009.01545.x>.
- [12] de Magalhães, S.V.G.; Franklin, W.R.; Andrade, M.V.A.: An efficient and exact parallel algorithm for intersecting large 3-d triangular meshes using arithmetic filters. *Computer-Aided Design*, 120, 102801, 2020. <http://doi.org/10.1016/j.cad.2019.102801>.
- [13] Douze, M.; Franco, J.S.; Raffin, B.: Quickcsg: Fast arbitrary boolean combinations of n solids. arXiv preprint arXiv:1706.01558, 2017. <http://doi.org/10.48550/arXiv.1706.01558>.

- [14] Feito, F.R.; Ogáyar, C.J.; Segura, R.J.; Rivero, M.: Fast and accurate evaluation of regularized boolean operations on triangulated solids. *Computer-Aided Design*, 45(3), 705–716, 2013. <http://doi.org/10.1016/j.cad.2012.11.004>.
- [15] Gottschalk, S.; Lin, M.C.; Manocha, D.: Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 171–180, 1996. <http://doi.org/10.1145/237170.237244>.
- [16] Gunther, J.; Popov, S.; Seidel, H.P.; Slusallek, P.: Realtime ray tracing on gpu with bvh-based packet traversal. In *2007 IEEE Symposium on Interactive Ray Tracing*, 113–118. IEEE, 2007. <http://doi.org/10.1109/RT.2007.4342598>.
- [17] Hachenberger, P.; Kettner, L.; Mehlhorn, K.: Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. *Computational Geometry*, 38(1-2), 64–99, 2007. <http://doi.org/10.1016/j.comgeo.2006.11.009>.
- [18] Heidelberger, B.; Teschner, M.; Gross, M.H.: Volumetric collision detection for deformable objects. *CS technical report*, 395, 2003. <http://doi.org/10.3929/ethz-a-006665865>.
- [19] Jacobson, A.; Panozzo, D.; Schüller, C.; Diamanti, O.; Zhou, Q.; Pietroni, N.; et al.: libigl: A simple c++ geometry processing library. *Google Scholar*, 2013.
- [20] Jiménez, P.; Thomas, F.; Torras, C.: 3d collision detection: a survey. *Computers & Graphics*, 25(2), 269–285, 2001. [http://doi.org/10.1016/S0097-8493\(00\)00130-8](http://doi.org/10.1016/S0097-8493(00)00130-8).
- [21] Jones, M.W.; Baerentzen, J.A.; Sramek, M.: 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4), 581–599, 2006. <http://doi.org/10.1109/TVCG.2006.56>.
- [22] Klosowski, J.T.; Held, M.; Mitchell, J.S.; Sowizral, H.; Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE transactions on Visualization and Computer Graphics*, 4(1), 21–36, 1998. <http://doi.org/10.1109/2945.675649>.
- [23] Kockara, S.; Halic, T.; Iqbal, K.; Bayrak, C.; Rowe, R.: Collision detection: A survey. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, 4046–4051. IEEE, 2007. <http://doi.org/10.1109/ICSMC.2007.4414258>.
- [24] Kornberger, B.: C++ constrained delaunay triangulation fade2d, <https://www.geomat/fade2d/html/indexhtml>.
- [25] Lauterbach, C.; Garland, M.; Sengupta, S.; Luebke, D.; Manocha, D.: Fast bvh construction on gpus. In *Computer Graphics Forum*, vol. 28, 375–384. Wiley Online Library, 2009. <http://doi.org/10.1111/j.1467-8659.2009.01377.x>.
- [26] Möller, T.; Trumbore, B.: Fast, minimum storage ray-triangle intersection. *j graph tools*; 2 (1): 21–8, 1997. <http://doi.org/10.1080/10867651.1997.10487468>.
- [27] Naylor, B.; Amanatides, J.; Thibault, W.: Merging bsp trees yields polyhedral set operations. *ACM Siggraph Computer Graphics*, 24(4), 115–124, 1990. <http://doi.org/10.1145/97880.97892>.
- [28] Nef, W.: Beiträge zur theorie der polyeder: mit anwendungen in der computergraphik. (No Title), 1978.
- [29] Parker, S.G.; Bigler, J.; Dietrich, A.; Friedrich, H.; Hoberock, J.; Luebke, D.; McAllister, D.; McGuire, M.; Morley, K.; Robison, A.; et al.: Optix: a general purpose ray tracing engine. *Acm transactions on graphics (tog)*, 29(4), 1–13, 2010. <http://doi.org/10.1145/1778765.1778803>.
- [30] Pavić, D.; Campen, M.; Kobbelt, L.: Hybrid booleans. In *Computer Graphics Forum*, vol. 29, 75–87. Wiley Online Library, 2010. <http://doi.org/10.1111/j.1467-8659.2009.01545.x>.
- [31] Qin, Y.; Luo, Z.; Wen, L.; Feng, C.; Zhang, X.; Lan, M.; Liu, B.: Research and application of boolean operation for triangular mesh model of underground space engineering-boolean operation for triangular mesh model. *Energy Science & Engineering*, 7(4), 1154–1165, 2019. <http://doi.org/10.1002/ese3.335>.
- [32] Seers, T.: Fast mesh-mesh intersection using ray-tri intersection with octree spatial partitioning. *MathWorks File Exchange*, 2015.

- [33] Sheng, B.; Li, P.; Fu, H.; Ma, L.; Wu, E.: Efficient non-incremental constructive solid geometry evaluation for triangular meshes. *Graphical Models*, 97, 1–16, 2018. <http://doi.org/10.1016/j.gmod.2018.03.001>.
- [34] Sheng, B.; Liu, B.; Li, P.; Fu, H.; Ma, L.; Wu, E.: Accelerated robust boolean operations based on hybrid representations. *Computer Aided Geometric Design*, 62, 133–153, 2018. <http://doi.org/10.1016/j.cagd.2018.03.021>.
- [35] Trettner, P.; Nehring-Wirxel, J.; Kobbelt, L.: Ember: exact mesh booleans via efficient & robust local arrangements. *ACM Transactions on Graphics (TOG)*, 41(4), 1–15, 2022. <http://doi.org/10.1145/3528223.3530181>.
- [36] Wang, C.C.: Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE transactions on visualization and computer graphics*, 17(6), 836–849, 2010. <http://doi.org/10.1109/TVCG.2010.106>.
- [37] Yongbin, J.; Liguang, W.; Lin, B.; Jianhong, C.: Boolean operations on polygonal meshes using obb trees. In *2009 International Conference on Environmental Science and Information Application Technology*, vol. 1, 619–622. IEEE, 2009. <http://doi.org/10.1109/ESIAT.2009.128>.
- [38] Zhao, H.; Wang, C.C.; Chen, Y.; Jin, X.: Parallel and efficient Boolean on polygonal solids. *The Visual Computer*, 27, 507–517, 2011. <http://doi.org/10.1007/s00371-011-0571-1>.
- [39] Zhou, Q.; Grinspun, E.; Zorin, D.; Jacobson, A.: Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, 35(4), 1–15, 2016. <http://doi.org/10.1145/2897824.2925901>.
- [40] Zhou, Q.; Jacobson, A.: Thingi10k: A dataset of 10,000 3D-printing models. *arXiv preprint arXiv:1605.04797*, 2016. <http://doi.org/10.48550/arXiv.1605.04797>.