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

Authors:
Yuanhui Xiao, xiao20231115@163.com, Guangxi Normal University
Ming Chen, hustcm@hotmail.com, Guangxi Normal University
Shouxin Chen, chen.csx@outlook.com, Guangxi Normal University
Shenglian Lu, lsl@gxnu.edu.cn, Guangxi Normal University

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

DOI: 10.14733/cadconfP.2024.295-300

Introduction:
As a fundamental algorithm of three-dimensional (3D) modelling, 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 [14, 6, 5] and volumetric representation [16, 15, 10, 8] methods that are designed specifically for parallel computation, OptiX [13], 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) [7] 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.

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. the intersection points are calculated using Eq. (2.1) and degenerate cases are handled.

Step 2: Triangle tessellation (see Fig. 1 (c)). Using the constrained Delaunay triangulation (CDT) algorithm [4], 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. This step is accelerated in parallel using the OpenMP [3] 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. Then, retain the triangles in accordance with the rules of boolean operations to obtain the final result of boolean operations.(see Fig. 1 (e)).

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.

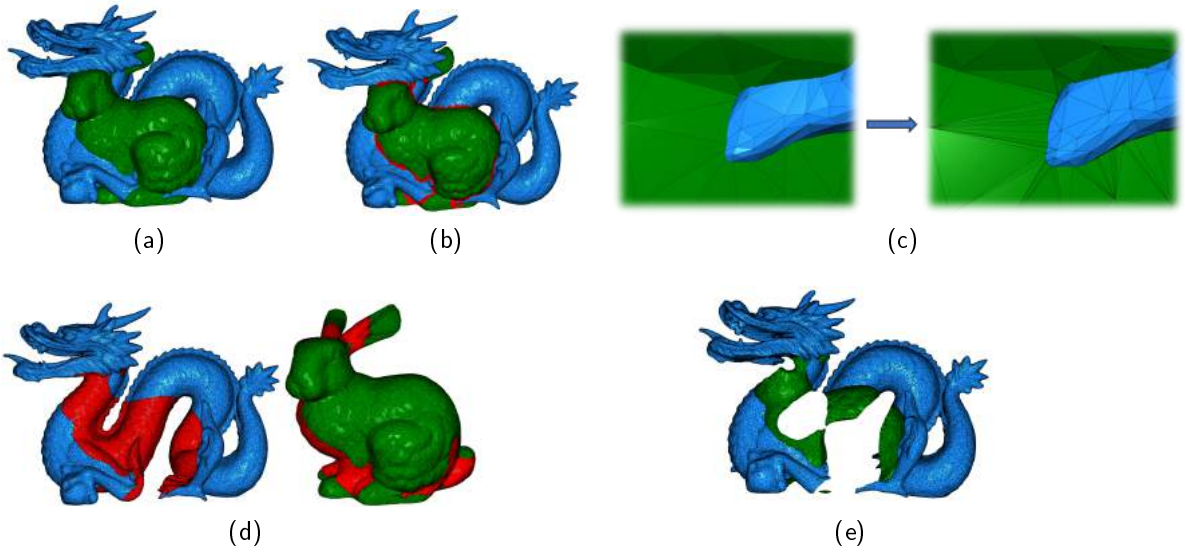

Fig. 1: Boolean operation procedure of triangular mesh based on ray tracing. (a) Input two closed, nonself-intersecting 2-manifold models. (b) Use ray tracing to perform the intersection test and compute the intersection points; the red portion of the diagram represents the set of intersecting triangles. (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.

Intersection Computation:

A method similar to the Moller-Trumbore algorithm [12] 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

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2 \tag{2.1}$$

where (u, v) are the barycentric coordinates, which must fulfil $u \geq 0$, $v \geq 0$ and $u+v \leq 1$.

$\{M_i\}_{i\in\{1,2\}} = (V_j, T_j)$ denotes the meshes engaged in Boolean operations, where V_i is the set of vertices and T_i 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 T_i. As shown in Fig. 2, 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:

1. Set vertex V_1 as the origin O of the ray.
2. Set the normalized direction of vector $\mathbf{V_1V_2}$ as the ray direction $\mathbf{D}$.
3. 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, the 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 intersect triangle can be obtained by invoking the API; the intersection point P_i is then computed using Eq. (2.1).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.

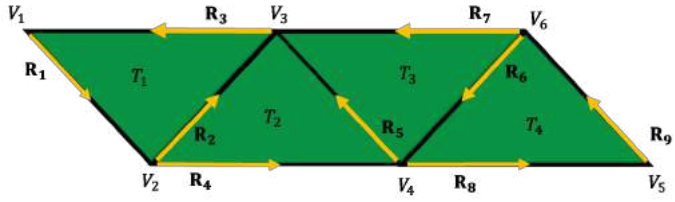

Fig. 2: Rays are emitted along the edges of M_i, and ray tracing is performed to complete the intersection test and calculation.

Due to the accumulation of numerical errors in the single-precision focal-point calculations of the OptiX engine, In two triangles intersecting at a point or edge and in coplanar triangle cases, the intersection points cannot be obtained stably.We use numerical perturbations to improve both cases.

Triangle Tessellation:

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.

1. If $T_0 = T_0^{'}$ and $T_1 = T_1^{'}$, then connect points P and Q. As shown in Fig. 3 (a), P_1P_2 is a constrained line segment.
2. If $T_0 = T_1^{'}$ and $T_1 = T_0^{'}$, then connect points P and Q. As shown in Fig. 3 (b), P_1P_2 is a constrained line segment.

The triangle is triangulated using the CDT algorithm in the third-party library Fade2D [11] in 2D.

Inside/Outside Classification:

After triangle tessellation, each triangle is either completely inside or completely outside of the model. We utilize the high efficiency of OptiX intersection judgment 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 $\cos\theta$,

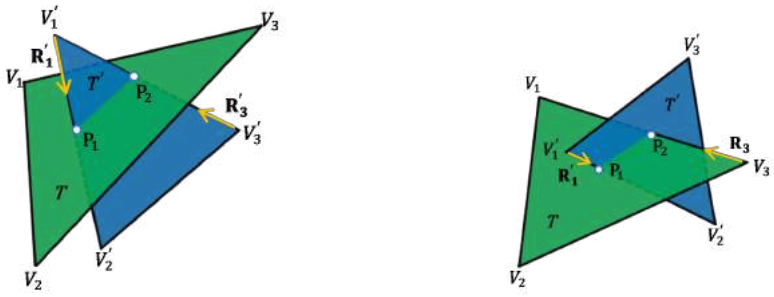

Fig. 3: The two most common triangle-triangle intersection cases. The triangles $T^{'}$ and T belong to two input models. (a) The rays $\mathbf{R_1^{'}}$ and $\mathbf{R_3^{'}}$ emitted from $T^{'}$ intersect triangle T at points P_1 and P_2, respectively. (b) The ray $\mathbf{R_1^{'}}$ emitted from triangle $T^{'}$ and the ray $\mathbf{R_3}$ emitted from triangle T intersect at points P_1 and P_2, respectively.

where θ is the angle between the direction vector $\mathbf{D}$ of the ray and the normal vector $\mathbf{n}$ of the nearest intersecting triangle (see Fig. 4). As ray tracing is performed very quickly on the GPU, the entire procedure is completed in a short amount of time.

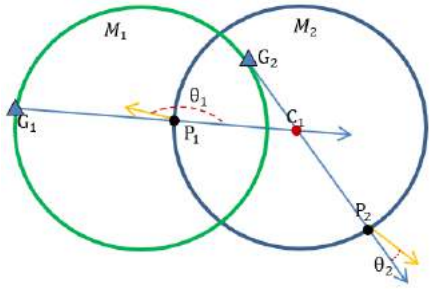

Fig. 4: M_1 and M_2 represent two models. The normal vectors (yellow arrows) of triangles in model M_2 are directed outwards, and C_1 is the centre of model M_2. The ray emitted by the triangle of the barycentre G_1 intersects M_2 at point P_1, and $\cos\theta_1 < 0$, so the triangle is judged to be 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.

Results and Discussion:

Three examples were used to compare the proposed algorithm with CGAL Nef Polyhedra [2], CGAL Corefine [2], Cork [1], QuickCSG [5], and Zhou's method [17] in LibIGL [9]. 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 1 and Fig. 5. Table 1 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

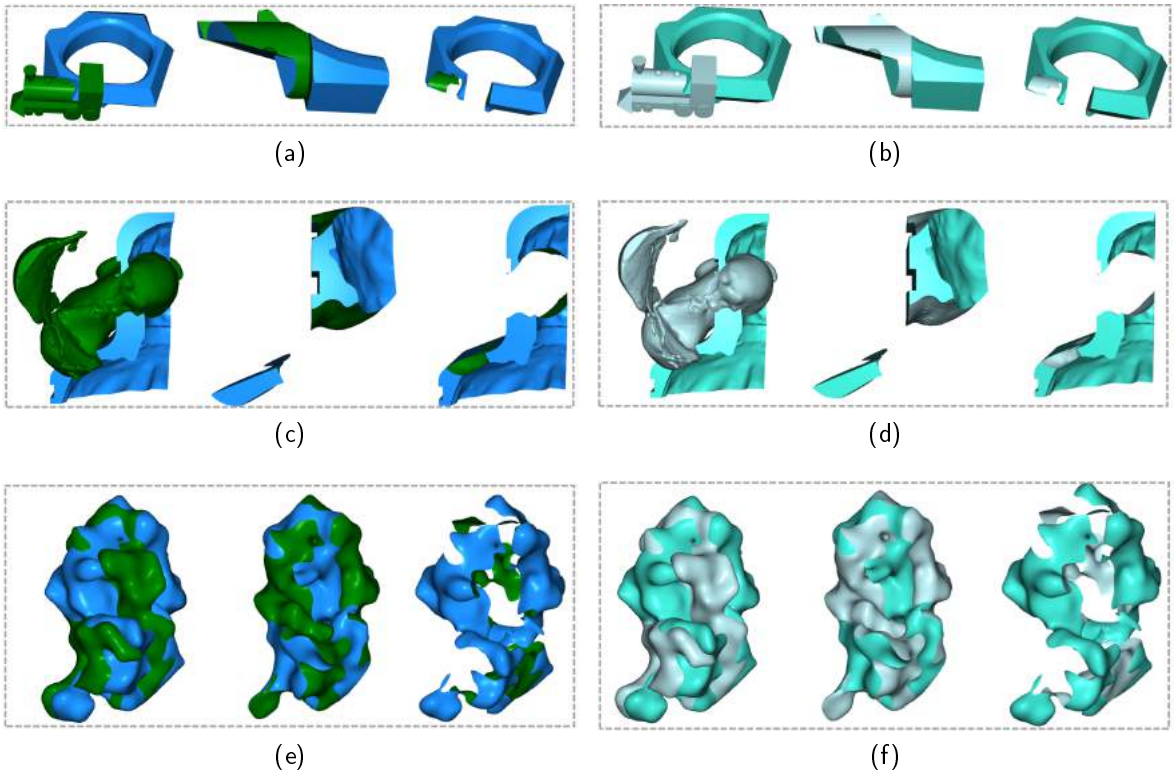

Fig. 5: The outputs of the six examples in Table 1 are the union, intersection, and difference sets. The model is from the Thingi10K Dataset. (a), (c) and (e)are the outputs of the proposed method. (b), (d) and (f) are the outputs of Zhou's method in LibIGL.

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.

Table 1: Computation time statistics (seconds) .

Example	*Model**	Face Num	Intersect Face Num	Ours	QuickCSG	LibIGL	Cork	CGAL Nef	CGAL Core
1	74890 ∪ 104738	15k	0.4K	0.02	0.02	3.51	1.10	26.15	1.18
2	57937 ∪ 441711	147k	2K	0.05	0.22	131.8	9.23	571.6	18.23
3	461112 ∪ 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.

Acknowledgement:
This research is funded by the funding of Natural Science Foundation of China (No: 61662006).

References:

[1] Gilbert Bernstein. Cork boolean library, https://github. com/gilbo/cork. 2013.

[2] CE Board. Cgal, computational geometry algorithms library, http://www.cgal.org.

[3] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[4] L Paul Chew. Constrained delaunay triangulations. In *Proceedings of the third annual symposium on Computational geometry*, pages 215–222, 1987.

[5] Matthijs Douze, Jean-Sébastien Franco, and Bruno Raffin. Quickcsg: Fast arbitrary boolean combinations of n solids. *arXiv preprint arXiv:1706.01558*, 2017. https://doi.org/10.48550/arXiv.1706.01558

[6] Francisco R Feito, Carlos J Ogáyar, Rafael Jesús Segura, and ML Rivero. Fast and accurate evaluation of regularized boolean operations on triangulated solids. *Computer-Aided Design*, 45(3):705–716, 2013. https://doi.org/10.1016/j.cad.2012.11.004

[7] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime ray tracing on gpu with bvh-based packet traversal. In *2007 IEEE Symposium on Interactive Ray Tracing*, pages 113–118. IEEE, 2007. https://doi.org/10.1109/RT.2007.4342598

[8] Bruno Heidelberger, Matthias Teschner, and Markus H Gross. Volumetric collision detection for derformable objects. *CS technical report*, 395, 2003. https://doi.org/10.3929/ethz-a-006665865

[9] Alec Jacobson, Daniele Panozzo, C Schüller, Olga Diamanti, Qingnan Zhou, N Pietroni, et al. libigl: A simple c++ geometry processing library. *Google Scholar*, 2013.

[10] Mark W Jones, J Andreas Baerentzen, and Milos Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on visualization and Computer Graphics*, 12(4):581–599, 2006. https://doi:10.1109/TVCG.2006.56

[11] B Kornberger. C++ constrained delaunay triangulation fade2d, https://www.geom.at/fade2d/html/index.html.

[12] T Möller and B Trumbore. Fast, minimum storage ray-triangle intersection. j graph tools; 2 (1): 21–8, 1997. https://doi.org/10.1080/10867651.1997.10487468

[13] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *Acm transactions on graphics (tog)*, 29(4):1–13, 2010. https://doi.org/10.1145/1778765.1778803

[14] Bin Sheng, Ping Li, Hongbo Fu, Lizhuang Ma, and Enhua Wu. Efficient non-incremental constructive solid geometry evaluation for triangular meshes. *Graphical Models*, 97:1–16, 2018. https://doi.org/10.1016/j.gmod.2018.03.001

[15] Charlie CL Wang. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE transactions on visualization and computer graphics*, 17(6):836–849, 2010. https://doi:10.1109/TVCG.2010.106

[16] Hanli Zhao, Charlie CL Wang, Yong Chen, and Xiaogang Jin. Parallel and efficient boolean on polygonal solids. *The Visual Computer*, 27:507–517, 2011. https://doi.org/10.1007/s00371-011-0571-1

[17] Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, 35(4):1–15, 2016. https://doi.org/10.1145/2897824.2925901