Boolean Operation for CAD Models Using a Hybrid Representation

YINGYU YANG, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
XIAOHONG JIA*, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
BOLUN WANG, RWTH Aachen University, Germany
JIEYIN YANG, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
SHIQING XIN, Shandong University, China
DONG-MING YAN, State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China

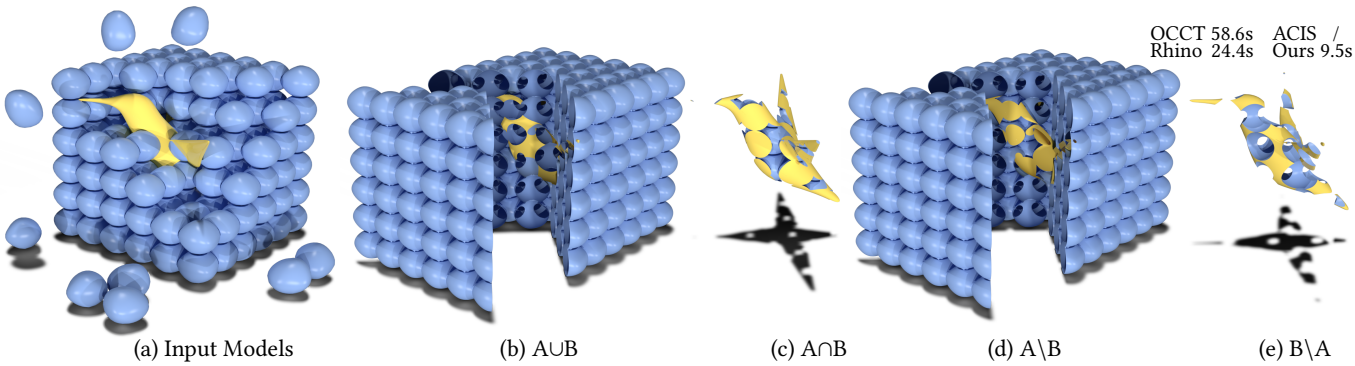


Fig. 1. We present a novel algorithm for fast Boolean operations on B-Rep models using a hybrid representation. This example first performs a Boolean union of 216 spheres, with the result denoted as A, taking input as shown in (a), followed by Boolean union (b), intersection (c), and subtraction (d)(e) operations with a bird model, denoted as B. The results are shown sequentially, with (b) and (d) being cross-sectional views. We recorded the total time, with our algorithm being the fastest. ACIS fails when performing Boolean operations between A and B.

Boolean operations for Boundary Representation (B-Rep) models are among the most commonly used functions in Computer Aided Design (CAD) systems. They are also one of the most delicate soft modules, with challenges arising from complex algorithmic flows and efficiency and accuracy issues, especially in extreme cases. Common issues encountered in processing complex models include low efficiency, missing results, and non-watertightness. In this paper, we propose a novel algorithm for efficient and accurate Boolean operations on B-Rep models. This is achieved by establishing a bijective mapping between B-Rep models and the corresponding triangle meshes with controllable approximation error, thus mapping B-Rep Boolean operations to mesh Boolean operations. By using conservative intersection detection on the mesh to locate all surface intersection curves and carefully handling degeneration and topology errors, we ensure that the results are consistently watertight and correct. We demonstrate the superior efficiency of the proposed method using the open-source geometry engine OCCT, the commercial engine ACIS, and the commercial software Rhino as benchmarks.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Shape modeling**; **Parametric curve and surface models**.

Additional Key Words and Phrases: Boolean operation, solid modeling, surface intersection

ACM Reference Format:
Yingyu Yang, Xiaohong Jia, Bolun Wang, Jieyin Yang, Shiqing Xin, and Dong-Ming Yan. 2025. Boolean Operation for CAD Models Using a Hybrid Representation. *ACM Trans. Graph.* 44, 4, Article 114 (August 2025), 17 pages. https://doi.org/10.1145/3730908

*Corresponding author.

Authors' Contact Information: Yingyu Yang, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China, yangyingyu23@mails.ucas.ac.cn; Xiaohong Jia, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China, xhjia@amss.ac.cn; Bolun Wang, Visual Computing Institute, RWTH Aachen University, Germany, bolun.wang@rwth-aachen.de; Jieyin Yang, State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China, yangjieyin17@mails.ucas.ac.cn; Shiqing Xin, School of Computer Science, Shandong University, China, xinshiqing@sdu.edu.cn; Dong-Ming Yan, State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China, yandongming@gmail.com.

ACM 1557-7368/2025/8-ART114
https://doi.org/10.1145/3730908

1 Introduction

Boolean operations on solid models are one of the most important modeling schemes widely used in modern geometric modeling software. However, Boolean operations in such software are usually not publicly documented, in which models are typically represented using Boundary Representation (B-Rep) [Braid 1974], encompassing curved surfaces and parametric boundary curves. Among the few open source geometric libraries that provide a reference implementation of B-Rep Boolean algorithms, Open CASCADE (OCCT) [2024] is by far the most widely adopted, with successful applications in fields such as construction, aerospace, and automotive industries. However, its efficiency is relatively limited, especially when complex models are processed. This inefficiency stems from the highly intricate nature of B-Rep Boolean workflows, which primarily involve two challenging steps: trimmed-surface intersection and topology classification. Robustness is also difficult to ensure, especially in cases involving tangency or small loops.

In contrast to B-Rep Booleans, mesh Boolean operations have demonstrated exceptional performance, delivering interactive rates and robust results, even when handling large-scale inputs. In fact, mesh Boolean and B-Rep Boolean produce the same topology in most cases, which can serve as a guide for B-Rep Boolean operations. However, to date, work using polyhedral approximation to attempt to solve B-Rep Boolean operations [Toriya et al. 1989; Yamaguchi and Tokieda 1984] either fails to handle models with NURBS surfaces or cannot guarantee correct results due to discretization errors.

To address the limitations of traditional B-Rep Boolean, this work leverages the efficiency of mesh Boolean techniques for B-Rep models. We use a hybrid representation of the solid models, that is, B-Rep models equipped with discrete mesh representations, to construct a bijective mapping between the B-Rep model and its corresponding triangle mesh. The input B-Rep model is first discretized quickly into a triangle mesh within a user-defined tolerance. Mesh intersection operations are then performed to obtain an initial intersection result. Subsequently, the intersection curves are mapped back to the B-Rep model through numerical optimization, with the mesh intersection curves updated simultaneously. Finally, the inside/outside classification results on the mesh are used to extract the corresponding B-Rep model. To ensure robustness, additional tests are introduced to handle failure cases and degeneracies.

We tested our algorithm on two datasets: 10,000 random and 100 complex B-Rep model pairs, achieving 2∼20 times accelerations compared to traditional methods. Our algorithm performs without failure on all test cases, even containing tangency and small intersection loops.

The main contributions of this paper are as follows.

- A fast algorithm for Boolean operations on B-Rep models using a hybrid representation that significantly improves the state-of-the-art performance by leveraging the high performance of recently developed interactive mesh Boolean techniques.
- A conservative intersection detection algorithm for parametric surfaces using Octree and normal cones, effectively avoiding the loss of small loops and tangency during mesh intersection.
- Carefully handling degeneration and topology errors to avoid solving failures, thus largely improving the robustness.

2 Related Work

Boolean operations in solid modeling have been studied extensively for decades, which can be broadly classified into two categories, i.e., mesh Boolean that operates on polyhedral meshes, and B-Rep Boolean that deals with B-Rep models.

2.1 Mesh Boolean

Mesh Boolean consists mainly of two steps: mesh arrangement and inside/outside classification. The approximate Boolean operations are highly efficient [Wang 2011; Zhao et al. 2011]. However, many scenarios require accurate Boolean results, where the key challenges are the precision of floating-point operations and the speed of intersection. Using binary space partition tree [Bernstein and Fussell 2009; Campen and Kobbelt 2010; Nehring-Wirxel et al. 2021] combined with plane-based geometry can accurately compute intersections. Hybrid representations [Pavic et al. 2010; Sheng et al. 2018] are used to combine plane-based and vertex-based geometry, or to integrate surface and volumetric data. The use of winding number vectors [Zhou et al. 2016] addresses the problem of not producing closed, self-intersection-free output without restriction to the number of input surfaces. Combining winding numbers with plane-based mesh arrangements that eliminate the need for (pre-)constructing a global acceleration structure [Trettner et al. 2022] achieves notable improvements in both efficiency and robustness. Using rational number representations or exact geometric predicates in point-based methods comes with a high computational cost. To solve this problem, Attene et al. [2020] introduced the concept of indirect geometric predicates and used efficient floating-point filtering and expansion arithmetic to provide accurate results with minimal overhead. Cherchi et al. [2020] applied indirect geometric predicates to the mesh arrangement problem, significantly improving the efficiency of intersection computation. Later, [Cherchi et al. 2022] improved floating-point filters and coupled them with a robust ray casting approach, achieving Boolean operations at interactive frame rates. Guo and Fu [2024] proposed indirect offset predicates to address the issue of higher failure rates in floating-point filtering caused by large coordinate terms, reducing the use of exact arithmetic and improving speed. Lévy [2024] introduced an exact mesh intersection algorithm that computes the Weiler model, enabling robust Boolean operations for arbitrary CSG expressions.

While current state-of-the-art mesh Boolean techniques are highly efficient, meshes alone are not sufficient to accurately represent geometric models in CAD, where B-Rep Boolean operations are necessary. The main challenge lies in the complexity and time consumption of parametric surface intersection algorithms.

2.2 B-Rep Boolean

B-Rep Boolean operations involve inputs and outputs as B-Rep models. The existing literature can generally be divided into two categories: methods based on parametric surface intersection and approaches using a polyhedral approximation. The first category directly computes intersection curves between parametric surfaces. The early methods [Braid 1975; Requicha and Voelcker 1985] could only handle models composed of planes or quadratic surfaces, for which accurate intersection curves could be computed. Casale and

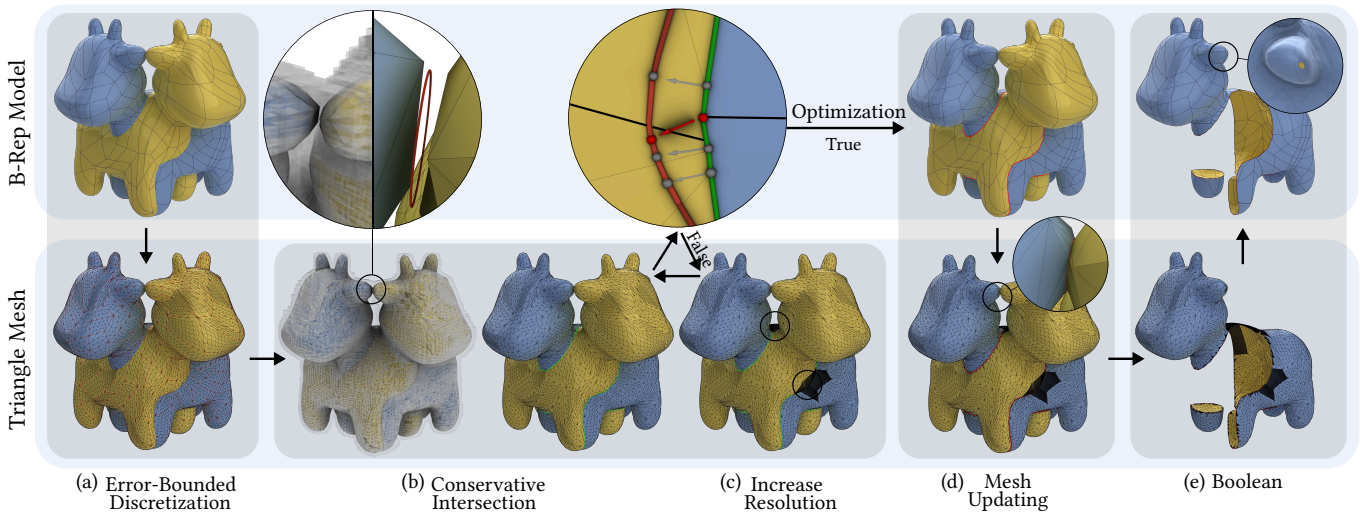


Fig. 2. Overview of our algorithm. We establish a bijection between the B-Rep model and the triangle mesh using hybrid representation. Our inputs are B-Rep models, which are mapped to a triangle mesh through error-bound discretization (a). Then, through the conservative intersection (b) between meshes, and by increasing the resolution (c) through iterating over erroneous regions (e.g. the green curve cannot converge to the accurate result by crossing boundaries since the red point lies on the boundary of an incorrect surface), we find all potential intersection curves between the B-Rep models. A numerical optimization method is then used to map the intersection back to the B-Rep models. After optimization, the intersection curves are used to update the mesh (d), maintaining the bijection between the intersection curve bounded patch on B-Rep models and its corresponding discretization. Finally, we perform the mesh's inside/outside classification and map the result patches back to the B-Rep model (e). Both Rhino and ACIS failed on this example.

Bobrow [1989] presented one of the first detailed descriptions of boundary evaluation for sculptured solids using the subdivision method. Krishnan et al. [2001] used a combination of curve-surface intersection and surface-surface intersection methods to improve accuracy. Later, [Fougerolle et al. 2005; Keyser et al. 2004] used both implicit and parametric representations to accurately express intersection curves using implicit equations and ensure precise inside/outside classification. Due to implicitization, they can only handle solids made of superquadrics or low-degree curved surfaces. There are also many algebraic methods that specifically focus on the surface intersection [Jia et al. 2022; Yang et al. 2023].

The efficiency of the B-Rep Boolean operations is often constrained by the intersection of parametric surfaces. To improve intersection speed, some studies discretize the input models into meshes to obtain approximate intersection lines. Yamaguchi and Tokieda [1984] first proposed repeatedly triangulating the input model to obtain the intersection lines, simplifying the algorithm and improving efficiency, but their algorithm could only handle planes and cylindrical surfaces. Using a combination of subdivision and polyhedral approximation methods, Toriya et al. [1989] proposed an algorithm applicable to free-form solids using the polyhedral approximation. However, it still requires performing an intersection between surfaces first, and topological issues may arise during approximation. Biermann et al. [2001] used triangle intersection and perturbation methods to provide approximate intersection curves, but this method could only be applied to subdivision surfaces and would locally change the input surfaces. García et al. [2011] based their method on triangle subdivision for clipping, but required that the input model be bounded by PN triangles.

2.3 Software and engines

Many open-source and commercial modeling kernels containing Boolean operation functions have been developed, but few descriptions of the algorithms used are available. The Boolean operation functionality of OCCT [2024] is widely used for its ease of use and long-term maintenance. ACIS [2024] is a renowned commercial 3D modeling kernel that provides powerful capabilities to create, manipulate, and analyze geometric models. The B-Rep Boolean function in ACIS is one of the industry standards due to its high robustness and efficiency. Industrial software such as Rhino [2024] also has a B-Rep Boolean function, which focuses more on free-form shape modeling. The results and performance of OCCT, ACIS, and Rhino will be used as benchmarks in this paper.

Our algorithm follows the idea of polyhedral approximation, utilizing implicit predicates and octree from mesh methods while addressing the issue of loop detection in surface intersection. It shows high speed and maintains robustness in degenerated cases.

3 Overview

Our algorithm takes as input B-Rep models and specified Boolean operations, including intersection, union, and subtraction. A B-Rep model [Braid 1974] is a 3D solid bounded by its 2D boundaries, which are composed of adjacent bounded surfaces called faces. Each face is a piece of parametric surface, including planes, quadratic surfaces, and NURBS surfaces. The faces are surrounded by a set of closed loops that trim the surface, with each loop composed of edges that form a continuous boundary, with each edge shared by two adjacent faces. The points where several faces meet are called vertices. The output model, obtained by applying the Boolean

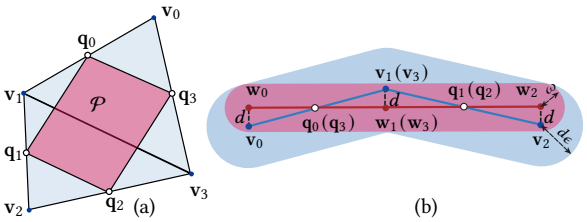


Fig. 3. (a) The four vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{v}_3$ are the four corners of the rational Bézier sub-patch, and $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$ and $\mathbf{q}_3$ are the four mid-points of the edges of $\square\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$. (b) is a side view of (a). By projecting $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{v}_3$ onto the red plane $\mathcal{P}$, we obtain the planar quadrangle $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$. The bounded red and blue regions in (b) are the ω envelope of $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ and d_ϵ envelope of $\square\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$. We then check the distance between the rational Bézier patch and $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ as explained in Theorem A.2.

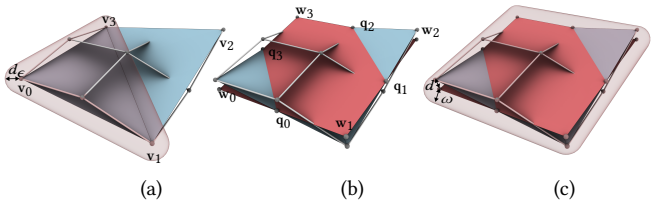


Fig. 4. M is composed of two triangles $\triangle\mathbf{v}_0\mathbf{v}_1\mathbf{v}_3$ and $\triangle\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$. One may directly select one of the two triangles as the convex polygon to compute control points - polygon distances as in (a). However, it may be reported exceeding d_ϵ while using our method (b) the rational Bézier patch (blue surface) is reported within d_ϵ (c), meaning our method leads to a smaller number of triangles in the final discretization. Thus, we apply the method (a) only when the polygon $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ is concave.

operation to the input models, is a B-Rep model enclosed by parts of the input surfaces.

Our hybrid representation maintains a bijective mapping between each B-Rep model and its discretization throughout the Boolean operation process. It allows us to compute Booleans of polygonal meshes as prior knowledge and then maps back the intersections to the B-Rep models to obtain the final Boolean operation results. Our method is accurate due to our conservative discretization scheme, normal estimation, and careful handling of degenerate cases like tangency, small loops, and coplanarity. The results generated by our method are ensured to be watertight B-Rep models corresponding to the specified Boolean operations. Our method outperforms the state-of-the-art methods in efficiency: compared to ACIS, Rhino, and OCCT, we approximately achieved a 2∼20 times acceleration on the 100 test cases, without failures.

Our algorithm proceeds in three steps. First, a bijective mapping between each B-Rep model and its triangular discretization is generated, and the intersection between the triangle meshes is performed, as described in Sec. 4.1 and Sec. 4.2. The discretization accuracy is guaranteed under a given surface-to-mesh distance tolerance d_ϵ using an iterative triangulation scheme. Then we leverage the fast mesh intersection methods described in [Cherchi et al. 2022].

Second, the intersection loops from the triangle meshes are mapped back to the B-Rep models. The bijectivity of the mappings ensures that each intersection loop can be mapped to either of the two associated B-Rep models. To obtain a unique and accurate representation of each loop, we apply a numerical optimization method to create a correspondence between the parametric representations of each loop on both B-Rep models. The failed areas will undergo local intersection handling or local mesh refinement until all the failures are resolved. The details of this process are described in Sec. 4.3 and Sec. 4.5.

Finally, the topology is updated to conduct the Boolean results. As described in Sec. 4.4, after obtaining the intersection curves, we re-triangulate the meshes to maintain validity and the bijectivity between each B-Rep patch bounded by the intersection loops and its corresponding discretization. We then perform an inside-outside classification on each patch of the resulting mesh to retain the parts according to the specific Boolean operation, thus the corresponding B-Rep patches can also be retained to finalize the B-Rep Boolean.

4 Methodology

In this section, we delve into the details on surface discretization, mesh intersection, optimization, Booleans and handling solving failures to ensure accuracy and efficiency of our method.

4.1 Surface discretization

Using meshes to locate intersection curves is much simpler and faster than computing intersection curves between algebraic surfaces. To ensure that all the intersections can be detected using meshes, we propose a new discretization method where the surface-to-mesh distance is guaranteed to be within an input error-bound d_ϵ; It also gives extra advantages that the mesh better reflects the surface shape while using fewer triangles in flatter regions, as shown in Fig. 7. Since the surfaces that make up the B-Rep model are sometimes trimmed, we first generate a mesh on the untrimmed surface and then trim it using the boundary curves.

4.1.1 Error-bounded triangulation. Our method first discretizes each closed B-Rep model composed of multiple surface patches into a triangle mesh. Then a bijective mapping between each surface patch and its discretization is constructed. The generated triangle mesh is a closed, watertight manifold under a given surface-to-mesh distance tolerance d_ϵ from the original B-Rep model.

We convert the surface patches of the B-Rep models into NURBS surfaces as inputs of this step. We denote the polygonal mesh M as a collection of $r+1$ planar convex polygons $M = \{P_i | i = 0, \ldots, r\}$. Assuming that a NURBS surface $\mathbf{p}(u, v)$ is defined on $u \in [u_0, u_1], v \in [v_0, v_1]$, we note that the NURBS surface can be partitioned into a collection of rational Bézier sub-patches [Piegl and Tiller 1996] $\mathbf{p}(u, v) = \bigcup_i \mathbf{p}_i(u_i, v_i)$, where each of the sub-patches is defined on $u_i \in [u_{0i}, u_{1i}] \subseteq [u_0, u_1]$, $v_i \in [v_{0i}, v_{1i}] \subseteq [v_0, v_1]$, and lies in the convex hull defined by its control net. The surface-to-mesh distance can be controlled by managing the distance from the control points of $\mathbf{p}_i(u_i, v_i)$ to P_i, as proved in Theorem A.1 in Appendix A.

For each sub-patch $\mathbf{p}(u, v)$ with control points $\mathbf{P} = \{\mathbf{p}_{i,j}\}$ where $i = 0, \ldots, m, j = 0, \ldots, n$, we create two triangles from the four corner control points, i.e. $\triangle\mathbf{p}_{00}\mathbf{p}_{0n}\mathbf{p}_{m0}$ and $\triangle\mathbf{p}_{0n}\mathbf{p}_{m0}\mathbf{p}_{mn}$, as an initial discretization of the surface. If the distance from the Bézier patch to these two triangles exceeds d_ϵ, it is subdivided into four sub-patches until the tolerance is fulfilled, while the mesh is updated throughout.

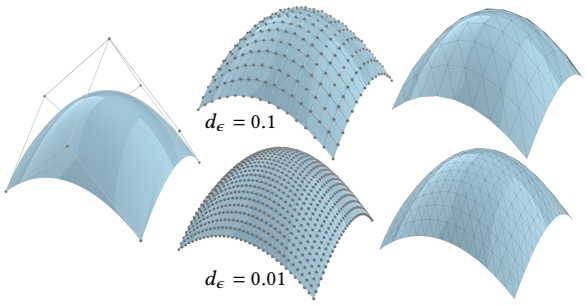


Fig. 5. The refinement results under different d_ϵ. On the left, the original surface is a bi-quadratic Bézier surface scaled to fit within a bounding box with a diagonal length of 1. When d_ϵ is set to 0.1, the subdivision stops after three iterations; when d_ϵ is set to 0.01, the subdivision stops after four iterations. The image shows the control polygons after subdivision and the final mesh results obtained through discretization.

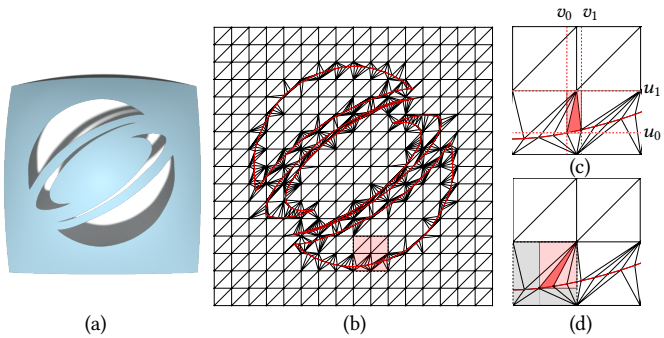


Fig. 6. Illustration of computing $d(T)$ of triangles containing sampling points on the boundaries. A trimmed surface (a) is triangulated using CDT (b). (c) and (d) are the close-ups of the red regions in (b). For the triangle in (c), we find the minimal rectangular region $(u_0, u_1) \times (v_0, v_1)$ covering it, compute the Bézier patch defined on it, and find the maximum distance between its control points and the triangle. In (d), the region is contained within a gray region defining a Bézier patch, thus the Bézier patch defined on the red region can be obtained by further subdividing the gray region.

The following method is used to check if the distance between the rational Bézier patch and the two triangles is within d_ϵ. As illustrated in Fig. 3, we project the four corners of the patch onto the plane passing through the midpoints of the quadrangular formed by the two triangles to obtain the planar polygon. If the polygon is convex, we can control the surface-to-mesh distance by controlling the distance from the control points to the polygon, as proved in Theorem A.2 in Appendix A. If the polygon is concave (only 10 out of 33,502 NURBS patches in our test dataset), we use the direct implementation of Theorem A.1 which chooses one of the two generated triangles of each patch as the convex planar polygon (taking Fig. 4 as a reference).

The vertices of the generated mesh are directly sampled from the refined NURBS patch as shown in Fig. 5, so the u-v parameters for each vertex can be assigned accordingly. Each iteration of our method increases the mesh density by 2×. Although it may generate more vertices than adaptive meshing, which only refines the regions exceeding the tolerance, one does not need to risk dealing with combinatorial problems caused by the unstructured discretization, which would also consume a significant amount of time.

Our method guarantees to terminate since the control net converges to the NURBS surface as the surface gets refined [Prautzsch and Kobbelt 1994], thus theoretically d_ϵ can be arbitrarily selected. In practice, we select d_ϵ as a value $10^{-2}d$ relative to the diagonal length d of the Axis-Aligned Bounding Box (AABB) of the B-Rep model as a balance of efficiency and accuracy.

4.1.2 Dealing with adjacency. Discretizing each patch by sampling regularly in its u-v domain requires techniques such as integer optimization [Bommes et al. 2009] to ensure watertightness between patches. We find it unnecessary; instead, we discretize each surface patch independently without considering its neighbors, re-sample the boundary curves, and reconstruct the triangulation around the boundaries.

For each surface, we first triangulate the rectangular u-v domain until reaching the given distance tolerance d_ϵ. Then, for each boundary curve, apply constrained Delaunay triangulation (CDT) in CGAL [2024] to retriangulate the two adjacent surfaces around the boundary, and remove the trimmed area as in [Diazzi et al. 2023], if it's a trimmed surface, generating a watertight mesh.

We extend our d_ϵ check to patches defined on triangular domains not aligned with the parametric curves to deal with the retriangulation around the NURBS boundaries. For a boundary triangle T, we find the minimal rectangular region in the parametric domain that covers its three vertices. Then using the NURBS subdivision technique, we can obtain a subpatch defined on the rectangular region. We compute the maximal distance $d(T)$ between the control points of the patch and the triangle as shown in Fig. 6. Thus, we re-define the discretization error bound $d_\epsilon(T)$ of a NURBS patch defined on a parametric triangle corresponding to the mesh triangle T as follows:

$$d_\epsilon(T) = \begin{cases} d_\epsilon & \text{if } T \text{ is an inner triangle,} \\ d(T) & \text{otherwise.} \end{cases}$$

An inner triangle is a triangle that does not contain sampling points on the boundaries. For convenience, $d_\epsilon(T)$ will be abbreviated as d_ϵ in the following text, with the understanding that a different value is used when dealing with boundary triangles. The distance bounds are then used in our octree search structure as explained in Sec. 4.2.1 for a conservative intersection detection.

The discretization results are bijective to the B-Rep models if the NURBS surfaces are regularly defined, such that we can use the mesh intersection results to provide proper initialization to solve the B-Rep intersections.

4.2 Mesh intersection

In this step, we detect intersections and compute intersection lines between the two triangle meshes. We refer to [Cherchi et al. 2022; Livesu 2019] as the mesh intersection computation method used in our paper. The results of this step will be used as hints for the computation of intersections of surfaces, which significantly accelerates the intersection calculation. As our discretization is bounded by the error d_ϵ, the intersections of the surfaces can be detected conservatively, thus improving the robustness and stability.

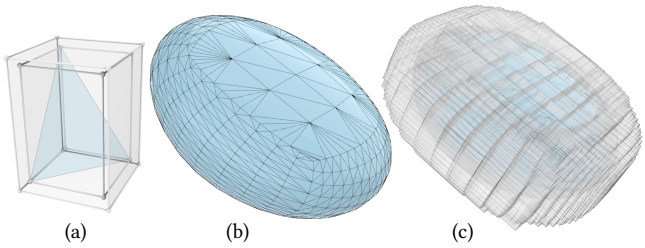


Fig. 7. Offset the Axis-Aligned Bounding Box (AABB). For each triangle, we first compute its AABB, then expand it by a distance of $d_\epsilon = 0.05d$ in both directions along each coordinate axis, where d is the diagonal length of the AABB. The mesh in (b) is obtained by discretizing a solid model. For each triangle in this mesh, we generate an offset AABB, resulting in (c), where the original solid model is fully contained within the offset AABB.

4.2.1 Conservative intersection detection. Since our method uses the triangle mesh of each surface to detect intersections, the detection results of the meshes may not match the surface intersection results. More specifically, there are five cases as shown in Fig. 8: Case I, both the meshes and surfaces intersect; Case II neither the meshes nor surfaces intersect; Case III the meshes miss intersections; Case IV the meshes detect intersections that do not exist in surfaces; and Case V both the meshes and surfaces intersect, but the intersected mesh patches do not correspond to the intersected surfaces. For Cases I and II, the detection of the mesh intersections provides valid results for the surface intersections. Case III and Case IV can be filtered by the distance threshold d_ϵ and our optimization-based intersection line computation, respectively. Case V is handled by further correction of curve topology and locally increasing discretization resolution. Here we explain how we eliminate Case III and leave Cases IV and V in Sec. 4.3 and 4.5, respectively.

Our discretization method ensures that the original surface is within a distance tolerance d_ϵ from the mesh. We denote M_A and M_B as the two triangle meshes of surfaces S_A and S_B, respectively, and $\mathbf{p}_{AB}$ is an intersection point of S_A and S_B. We can derive the following relation:

$$Dis(\triangle t_{AP}, \triangle t_{BP}) \leq Dis(\mathbf{p}_{AB}, \triangle t_{AP}) + Dis(\mathbf{p}_{AB}, \triangle t_{BP}) < 2d_\epsilon, \quad (1)$$

where $Dis(\cdot,\cdot)$ denotes the Euclidean distance between two triangles, $\triangle t_{AP}$ and $\triangle t_{BP}$ are two triangles in M_A and M_B, respectively, that are closest to $\mathbf{p}_{AB}$. Therefore, if there are two triangles $\triangle t_A \in M_A$ and $\triangle t_B \in M_B$ such that $Dis(\triangle t_A, \triangle t_B) < 2d_\epsilon$, this implies potential intersections of S_A and S_B.

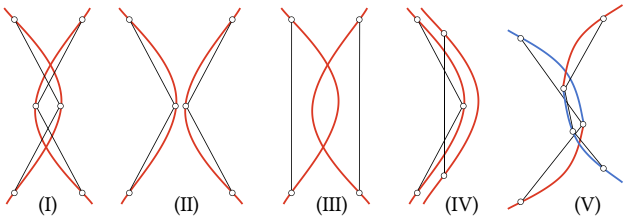


Fig. 8. 2D illustration of intersection classification. In the last case, the red and blue curves represent adjacent surfaces. The two blue surfaces do not intersect, but their meshes do.

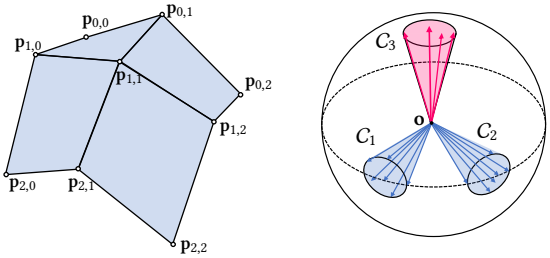


Fig. 9. The left figure shows a control net of a quadratic Bézier surface patch. The two circular half cones C_1 and C_2 cover its edge vectors $(\mathbf{p}_{i+1,j} - \mathbf{p}_{i,j})$ and $(\mathbf{p}_{i,j+1} - \mathbf{p}_{i,j})$. C_1 and C_2 take two vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ as their axes, and $\mathbf{o}$ as the common peak point. It can be proved that the two half cones cover the tangent vectors u and v of the rational Bézier patch [Khanteimouri and Campen 2023; Xu et al. 2013]. We hope to find the circular half cone C_3 that takes $\mathbf{a}_3$ as the axis and covers the normal vectors of the Bézier patch.

In our implementation, we use an octree to detect triangles that are closer than $2d_\epsilon$. For each triangle, we compute its AABB, offset the box by d_ϵ, and insert it into the tree, as shown in Fig. 7. For the triangles generated from the sampling points on the boundary curves, the offset distances become $d(T)$ as calculated in Fig. 6.

4.2.2 Redundant computation reduction. We use the normal vectors of the areas that have potential intersections to improve the detection accuracy. Given that within a small intersection loop, there must exist two points $\mathbf{p}_A \in S_A$ and $\mathbf{p}_B \in S_B$ such that the normal vector of S_A at $\mathbf{p}_A$ is collinear with that of S_B at $\mathbf{p}_B$ [Sederberg et al. 1989], we conservatively estimate the Gauss map of the corresponding region on each surface and check if two Gauss maps overlap. We use the control net of the rational Bézier patches of each surface to compute the region that covers the Gauss map of the surface. In Fig. 9, we propose a method to conservatively characterize the region of the normal vectors of the rational Bézier patch as a circular half cone. We have the following theorem:

Theorem 4.1. *The normal vectors of a Bézier patch are contained in a circular cone C_3 whose axis $\mathbf{a}_3$ is in the direction of $\mathbf{d} = \mathbf{a}_1 \times \mathbf{a}_2/\|\mathbf{a}_1 \times \mathbf{a}_2\|$, and the angle of C_3 is $\theta = \arccos(\mathbf{n} \cdot \mathbf{d})$, where $\mathbf{n}$ is the unit normal vector of a plane $\mathcal{P}$ that passes through $\mathbf{o}$ and is tangential to C_1 and C_2, and $(\mathbf{a}_1 \cdot \mathbf{n})(\mathbf{a}_2 \cdot \mathbf{n}) < 0$, if there exists a plane $\tilde{\mathcal{P}}$ passing through $\mathbf{o}$ such that the two half cones C_1 and C_2 are on the same side of $\tilde{\mathcal{P}}$, and $C_1 \cap C_2 = \mathbf{o}$. C_1, C_2, $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{o}$ are defined as in Fig. 9.*

The proof of Theorem 4.1 is given in Appendix B. Theorem 4.1 requires that the two complete cones extended by the two half-cones do not overlap. We note that this can always be achieved with our discretization technology, as long as the patch is regularly defined. Our method checks if $\mathbf{c}_1$ and $\mathbf{c}_2$ overlap after discretizing the patches, and refines the discretization if overlap occurs. Our Gauss map overlapping check reduces the computation and optimization for nonexistent intersections, thus reducing the number of invalid intersections detected by 40.7% and accelerating the subsequence optimization of intersections by an average of 34.9% on our dataset.

We employ a heuristic method to estimate the intersection points for the areas selected by the Gauss map overlapping checks. Assuming that $\triangle t_A \in M_A$ is a triangle that implies a potential intersection

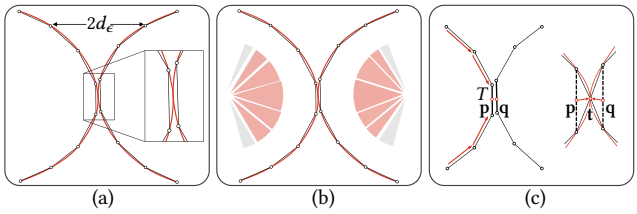


Fig. 10. 2D illustration of using conservative intersection check and Gauss map overlapping check. (a) Two surfaces contact at a tangency point but the meshes do not intersect. Triangles closer than $2d_\epsilon$ are filtered. (b) The red sectors represent the triangles whose corresponding patches overlap in the Gauss map estimations. (c) We locate the triangle T on the left surface nearest to the right one, calculate its centroid $\mathbf{p}$, and find the nearest point $\mathbf{q}$ on the right surface. These points are used as initial values of our numerical optimization method to solve the accurate intersection point $\mathbf{t}$. We finally update the meshes such that they meet at the accurate intersection point $\mathbf{r}$.

between S_A and S_B, we select the barycenter of $\triangle t_A$ and its closest point on M_B as the potential intersection points, as shown in Fig. 10. These two points are used as the seeds to find the accurate small intersection loop or a single tangential point if the intersection exists, which will be detailed in Section 4.3. Note that a point is regarded redundant if it falls in another accurately solved intersection curve. Such points will be directly removed and thus will not lead to significant time consumption.

4.2.3 Approximate parameter coordinates. After the mesh intersection step, each intersection curve is represented as a sequence of vertices. Owing to the implicit point representations used in [Cherchi et al. 2022], we can directly map each intersection point back to both NURBS surfaces corresponding to the meshes using barycentric coordinates by querying the triangles that intersect at that point. We note that the two points on the two surfaces mapped from the same intersection point generally do not coincide. We further optimize to solve this problem using the method mentioned below.

4.3 Intersection optimization

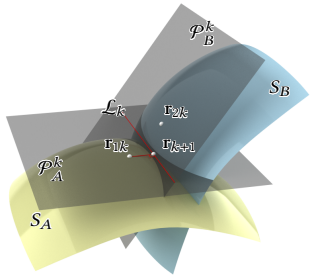
From the previous step, we obtained the intersection points of the meshes and their corresponding projections on the parametric domains. However, the NURBS surfaces may not intersect. Even if they do, each intersection point of the meshes might be projected to two different points on the two surfaces. Restricted by the triangulation resolution, small loops might be recognized as a single point (which should only occur at tangent points). We use a numeric optimization method to obtain more accurate intersection curves and distinguish tangent points and small loops.

We take advantage of two types of optimization in our intersection solving, as explained in Sec. 4.3.1 and Sec. 4.3.2. The first is Newton's root-finding method, which uses an algebraic way to find the optimal solutions for the intersection points. The second is a geometric optimization method. The termination condition for both methods during optimization is when the distance between the points is less than the given tolerance d_p.

4.3.1 Optimization using Newton's method. The approximate parameter coordinates obtained from the mesh intersection can be used as initial values for Newton's method to compute accurate intersection points on the parametric surfaces. For the cases where the intersection point is on the boundary curve, we restrict it to move along the curve. Details on the exact application of Newton's method are provided in Appendix C.

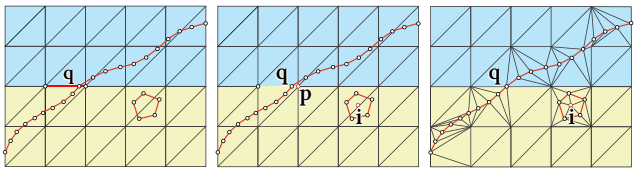
Due to the different scaling of the parameterizations of the two patches, Newton's iteration may provide inaccurate steps that excessively move one point out of the patch's parametric domain even if the intersection between two patches exists, causing solving failures. To improve solving accuracy, we use the undermentioned geometric method for more generic cases and use Newton's method only for tangential points and points gliding on boundary curves.

4.3.2 Optimization using a geometric method. Similar to Newton's method that linearizes the equations, we linearize the two surfaces geometrically as two tangent planes $\mathcal{P}_A^k$ in (u_k, v_k) and $\mathcal{P}_B^k$ in (s_k, t_k),i.e., $\mathcal{P}_A^k(u,v) = S_A(u_k,v_k) + \nabla S_A(u_k,v_k) \cdot (u,v)^T$, and $\mathcal{P}_B^k(s,t) = S_B(s_k,t_k) + \nabla S_B(s_k,t_k) \cdot (s,t)^T$. Then we project the point $\mathbf{r}_{1k}$ onto the intersection line $\mathcal{L}_k = \mathcal{P}_A^k \cap \mathcal{P}_B^k$ to obtain the optimal parametric locations of the two points for the iteration step. We note that this method is used only when the two tangent planes are not parallel. This method effectively reduces the likelihood of iterating outside the parametric domain when one of the points approaches the parametric boundary. In our experiments, it reduced such occurrences by 46.2%.



4.3.3 Choosing optimization methods for different cases. As Newton's method suffers from incorrect step lengths and the geometric method works only when the two tangent planes are not parallel, we choose different methods for different cases such that not only intersection loops but also tangent points and loops smaller than the mesh resolution can be correctly computed. If the two meshes do not intersect but are within the distance tolerance d_ϵ, there is a tangent point, or a small loop, or no intersection between the surfaces. We use Newton's method to solve such cases. For the cases where the intersection point is on an intersection loop between the two meshes, we use the geometric method. For both methods, if there is no solution in one of the two parametric domains, we regard it as a solving failure and rule out the aforementioned Case IV where the meshes detect intersections that do not exist between the surfaces. If the distance between the two points converges to zero during the optimization process, an accurate intersection point is found that lies on both S_A and S_B, as shown in Fig. 10 (c).

During optimization, we remove a point if it is too close to another point on the same loop. For cases where there is only one isolated intersection point in a potentially intersected region, or after removing all the points too close to each other, there is only one point left, the point being either a tangent point or a point on a small loop. Then we compare the normal vectors of the two surfaces



(a) (b) (c)

Fig. 11. Illustration of our mesh trimming in the parametric domain. The blue and yellow mesh patches are the discretizations of two adjacent surfaces from one B-Rep model. The red polyline is the inserted intersection curve, and point **q** is an intersection point on the boundary curve. We use CDT to ensure the polyline is the boundary of the trimmed triangle meshes. (a). We locate the constrained edge containing **q** (the red edge) and split it using **q**. (b). If an endpoint **p** of the split edge is too close to **q**, we merge **p** with **q** as shown in (c) to improve the mesh quality. If there are no other mesh vertices within an intersection loop, we insert one point **i** into it.

at the intersection point. If the two normal vectors are collinear, it is a tangent point. Otherwise, it is a point that belongs to a loop. For the latter case, we take advantage of the tracing method in [Bajaj et al. 1988] and use this point as a seed to find the whole loop.

4.3.4 Refinement. Since intersection points are obtained directly by optimizing the mesh intersection points, the resulting polylines may be uneven and sparse. We used a curvature-based refinement method to subdivide the intersection lines for further refinement. Specifically, for two consecutive points **p** and **q** on the optimized intersection line, each having parameter coordinates (u_0, v_0),(s_0, t_0) and (u_1, v_1), (s_1, t_1) respectively, we use the midpoints of their parameter domains as the initial values for optimization. The optimization results in an intersection **m** between **p** and **q**. We compute the distance from **m** to the line segment **pq** in 3D as the arc height h, calculate the angle α between the vectors **pm** and **mq**, and determine the chord length $l = \max(\|\mathbf{pm}\|, \|\mathbf{mq}\|)$. If the following conditions are met:

$$h < d_p \times 10^2, l < d_p \times 10^3, \alpha < \frac{\pi}{18},$$

then no further subdivision is performed and the refinement between **p** and **q** terminates. Otherwise, we continue to insert midpoints between **pm** and **mq**.

4.4 Booleans

Performing inside/outside classification on the surfaces of B-Rep models is challenging. Traditionally, for each patch, one may select a point on it, cast a ray from the point, and count the number of intersections between the ray and the models. However, numerical problems are easily encountered, leading to incorrect classifications and missing patches in the result. To avoid this issue, we leverage the inherent watertightness of the mesh Boolean. The results are then mapped back to the B-Rep model using the parametric and topology correspondence.

4.4.1 Mesh updating. As the intersections on the surfaces are relocated and refined during the optimization, the bijectivity is essentially broken. Each intersection curve is no longer mapped to the corresponding intersection curve between the two meshes, thus causing gaps or self-intersections. The distance threshold d_ϵ cannot be guaranteed either. Therefore, in this step, we trim and update the meshes using the intersection curves to maintain a correct topology, bijectivity with the corresponding surfaces, and the d_ϵ constraints. The intersection curves on the parametric surfaces are mapped to the meshes M_A and M_B by mapping each intersection point $\mathbf{r}$ to $\mathbf{r}_A \in M_A$ and $\mathbf{r}_B \in M_B$, respectively. Then we set $\mathbf{r}_A = \mathbf{r}_B = \mathbf{r}$, so that the two polylines in the meshes coincide with the intersection curve on the Bézier surfaces. Next, through CDT we obtain valid discretizations of the trimmed meshes, see Fig. 11. The generated trimmed meshes are still bijective to their corresponding surfaces. The triangulation can be totally operated in the parametric domain, it maps boundary curves to boundary curves, and contains no flipping triangles since the intersection curves are regular.

To improve remeshing quality, we remove a mesh vertex if it is too close to the intersection curve on the mesh. If there is no point within an intersection loop, we insert a point to ensure that the topology of the trimmed mesh patch aligns with the trimmed surface. For the newly generated boundary triangles around the intersection curve, we recalculate $d(T)$ to maintain controllable error.

4.4.2 Mesh and B-Rep Booleans.

Mesh Booleans. After trimming the meshes using the intersection curves, we directly apply a standard inside/outside classification step [Cherchi et al. 2022] to identify the triangles that need to be retained, thus completing the mesh Boolean operation.

B-Rep Booleans. The B-Rep Boolean operation results are restored as a collection of parameter surfaces and their boundary curves in the parametric domain. Instead of performing inside/outside classification on the B-Rep patches, we leverage the bijection to locate the surface patches retained using a mesh segmentation strategy.

Our algorithm segments the mesh Boolean results into patches along the boundary curves, which correspond to either the original boundary curves or the intersection curves. Starting from an inner triangle, i.e. not on the boundaries of each mesh patch, using it as a seed triangle for the patch, our algorithm expands the patch by including more neighboring inner triangles, until all the neighboring triangles of the patch are on the boundaries. The boundary curves can then be easily collected and mapped back to the parametric surfaces by fitting the curve in the parametric domain. All the patches are found if all the triangles in the mesh Boolean results are accessed. The B-Rep model Boolean operations are finalized by restoring the corresponding parametric surfaces and boundary curves of them.

4.4.3 Watertightness and correctness. The watertightness of our result is inherited from the mesh Boolean output, ensuring the mesh has no geometric gaps [Ye et al. 2024]. After mesh updating, the topology of the mesh matches that of the intended B-Rep model exactly. As a result, the topology of the mesh Boolean aligns with that of the expected B-Rep Boolean. The final B-Rep Boolean obtained via the bijection is therefore topologically correct.

4.5 Solving failures handling

Because our algorithm projects the intersections between two mesh models onto parametric surfaces to initialize optimization, if the

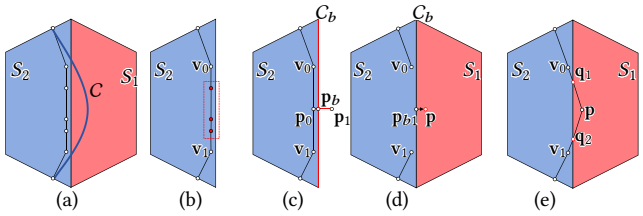


Fig. 12. Optimize across the boundaries. (a) C is the correct B-Rep intersection curve passing through surfaces S_1 and S_2. The intersection of the meshes is shifted onto S_2, completely bypassing S_1. (b) The red points represents the erroneous intersection points where optimization fails. They are bounded by two successfully optimized points $\mathbf{v}_0$ and $\mathbf{v}_1$ on S_2. (c) The midpoint $\mathbf{p}_0$ between $\mathbf{v}_0$ and $\mathbf{v}_1$ is inserted to replace the ones in the erroneous region. A full step takes $\mathbf{p}_0$ to an out-of-boundary location $\mathbf{p}_1$. We rescale the step to move $\mathbf{p}_0$ to $\mathbf{p}_b$ on the boundary curve C_b. In the next iteration (d), the point crosses the boundary by optimizing using the parameterization of S_1, thus achieving a smooth transition to the target position $\mathbf{p}$. The intersections $\mathbf{q}_1$ and $\mathbf{q}_2$ between the curve and C_b are solved in (e).

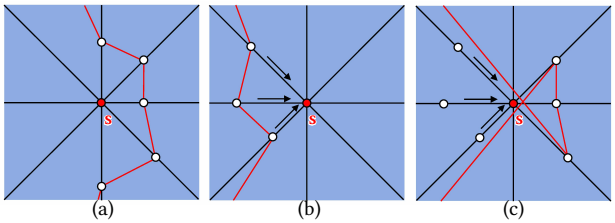


Fig. 13. The white dots in (a) are correct intersection points and $\mathbf{s}$ is a corner point. The black lines are boundary curves. In (b), the mesh intersection points lie on wrong boundary curves, with their Newton's directions indicated by arrows. In (c), crossing the corner point causes a topological error.

intersection appears on mesh patches that do not correspond to the two surfaces, the optimization often fails, as shown in Fig. 12(a). Even if the intersections are on correct discrete patches, the optimization may not find global minima. In other words, this problem occurs when insufficient discretization resolution cannot provide a proper initialization for optimization.

Our method first detects failure regions and then employs two strategies to solve the problem. After optimization, we collect the point pairs that cannot converge to a distance of 0 within their domains. Then for each erroneous region bounded by two successfully optimized points $\mathbf{v}_0$ and $\mathbf{v}_1$ on the same surface, i.e., all the optimization between $\mathbf{v}_0$ and $\mathbf{v}_1$ has failed, we apply the first strategy, which removes all the points between $\mathbf{v}_0$ and $\mathbf{v}_1$, replaces them with a new point, and restarts the optimization by allowing the points across the boundaries, as shown in Fig. 12. If such bound cannot be found or after crossing boundaries the optimization still fails, we increase the mesh resolution locally and re-optimize as the second strategy shown in Fig. 14. The above procedures are repeated if optimization failure persists. The algorithm is guaranteed to terminate since the mesh intersections converge to the spline surface intersections under refinement.

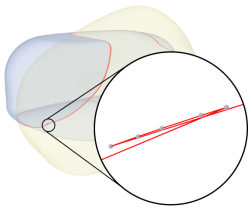
4.5.1 Optimize across boundaries. The first strategy applies to cases where on the same surface, there are two correct intersection points $\mathbf{v}_0$ and $\mathbf{v}_1$ on the two sides of the erroneous region, as shown in Fig. 12. The points in the erroneous region are removed and replaced with the midpoint $\mathbf{p}_0$ of $\mathbf{v}_0$ and $\mathbf{v}_1$. Then $\mathbf{p}_0$ is optimized using the geometric method described in Sec. 4.3.2. Instead of taking a full step length that takes the point to a position $\mathbf{p}_1$ outside the surface S_2 where the point is initially located, we truncate the step so that the point moves to $\mathbf{p}$ on the boundary curve C_b between S_2 and the neighboring surface S_1. In the next iteration, the optimization step of $\mathbf{p}$ is computed using the parameterization of S_1. In this way, we obtain a smooth transit of the point from S_2 to S_1 without reparametrization that would be necessary if we took the full steps [Marinov and Kobbelt 2004]. After obtaining the correct position of $\mathbf{p}$, we first solve the intersection points $\mathbf{q}_1$ and $\mathbf{q}_2$ on C_b using Newton's method, and then refine the curve as mentioned in Sec. 4.3.4 to obtain the final intersection curve.

We note that the first strategy only applies to the interior points but not to the boundary points that glide along the boundary curves. As shown in Fig. 13 (a), $\mathbf{s}$ is a corner point where more than two surfaces meet, and the white dots are the target positions of the boundary intersection points. If the initial positions indicated by the mesh intersections are given as in Fig. 13 (b), the points may glide toward $\mathbf{s}$ under optimization. However, after reaching $\mathbf{s}$, it is difficult to predict in which direction each vertex goes. The points may be directed to the curves close to the iterative directions. However, this may lead to topology errors, as illustrated in (c). Thus, we only use the first strategy in cases where the failure points are bounded by two successfully optimized points on the same surface. For other cases, we apply the second strategy as described below.

4.5.2 Local refinement. To address the issues mentioned above, we increase the mesh resolution of the parametric surfaces associated with the erroneous regions. As shown in Fig. 14, the erroneous region is bounded by two successfully optimized points $\mathbf{p}_f$ and $\mathbf{p}_b$ on boundary curves belonging to different patches. The intersection curve segment C_p between $\mathbf{p}_f$ and $\mathbf{p}_b$ is marked in green. The surfaces requiring refinement include those traversed by C_p (red regions) and the neighbors of a ring of them (orange regions).

We then compute the intersections between the meshes only in the refined regions. Since $\mathbf{p}_f$ and $\mathbf{p}_b$ are on the boundaries, from the new mesh intersection points we identify those lie on the same boundary curve as where $\mathbf{p}_f$ and $\mathbf{p}_b$ are located, and the curve bounded by them is used to replace the original. This process results in mesh intersection curves that provide more accurate estimations of surface intersections, as shown in Fig. 14 (b). Finally, the new segments are further optimized to obtain accurate intersection points.

4.5.3 Correction of reversed intersection. If the mesh resolution is not sufficient to maintain a one-to-one mapping between the mesh intersection and the B-Rep intersection curves, the surface intersection may exhibit a reverse sequence of points after convergence, see inset. The reversal of the intersection curve leads to topology errors within the curve and in the updated mesh after the process described



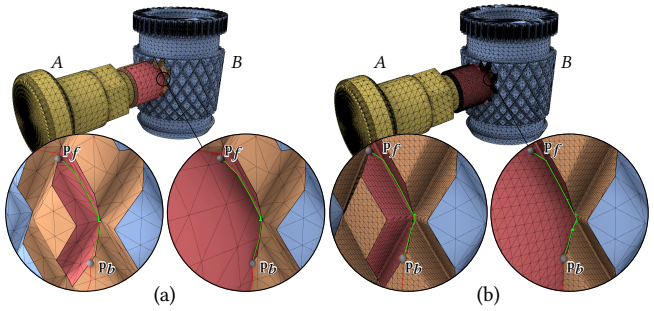


Fig. 14. Mesh refinement and local intersection correction. $\mathbf{p}_f$ and $\mathbf{p}_b$ are the nearest boundary intersection points around the erroneous region, forming the partial intersection curve C_p (green curve). Red regions represent surfaces traversed by C_p. Orange regions are the adjacent surfaces. Mesh refinement in these regions yields the corrected intersection curve in (b).

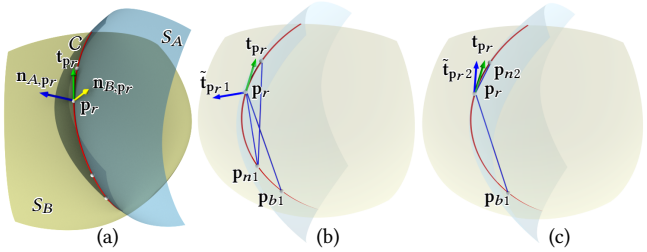


Fig. 15. Correcting reversed intersection points. $\mathbf{p}_r$ is a point on the intersection curve C between the two surfaces S_A and S_B. $\mathbf{n}_{A,\mathbf{p}_r}$ and $\mathbf{n}_{B,\mathbf{p}_r}$ are the normal vectors of S_A and S_B at $\mathbf{p}_r$, and $\mathbf{t}_{\mathbf{p}_r}$ is the tangent of C at $\mathbf{p}_r$. In (b), with points ordered incorrectly, a reversal is detected by comparing the discrete tangent vector $\tilde{\mathbf{t}}_{\mathbf{p}_r 1}$ with $\mathbf{t}_{\mathbf{p}_r}$. $\mathbf{p}_{n1}$ is then removed, and $\mathbf{p}_{n2}$ is connected to $\mathbf{p}_r$ (c). The process repeats at $\mathbf{p}_r$ until the correct next point is found, ensuring no reversal occurs.

in Sec. 4.4.1. To address this issue, our method first detects the reversed intersection point through tangential direction analysis, then removes the reversed point, and locally corrects the intersection curve.

The correction algorithm is shown in Fig. 15. At the intersection point $\mathbf{p}_r$, with the previous point $\mathbf{p}_b$ and the next point $\mathbf{p}_n$, the tangent of the intersection curve at $\mathbf{p}_r$ can be approximated using the polyline segments as:

$$\tilde{\mathbf{t}}_{\mathbf{p}_r} = \frac{\mathbf{p}_b\mathbf{p}_r}{\|\mathbf{p}_b\mathbf{p}_r\|} + \frac{\mathbf{p}_r\mathbf{p}_n}{\|\mathbf{p}_r\mathbf{p}_n\|} \tag{2}$$

The point $\mathbf{p}_r$ lies on the surfaces $S_A = \mathbf{s}_1(u, v)$ and $S_B = \mathbf{s}_2(s, t)$, and after optimization, it has the accurate parameter coordinates (u_r, v_r) and (s_r, t_r). The accurate tangent $\mathbf{t}_{\mathbf{p}_r}$ of the intersection curve C at $\mathbf{p}_r$ is tangential to both S_A and S_B, and thus lies in the tangent planes of both S_A and S_B at $\mathbf{p}_r$. Therefore, we can accurately compute $\mathbf{t}_{\mathbf{p}_r}$ as:

$$\mathbf{t}_{\mathbf{p}_r} = \mathbf{n}_{A,\mathbf{p}_r} \times \mathbf{n}_{B,\mathbf{p}_r},$$

where $\mathbf{n}_{A,\mathbf{p}_r}$ and $\mathbf{n}_{B,\mathbf{p}_r}$ are normals of S_A and S_B at $\mathbf{p}_r$. If $\mathbf{p}_b$, $\mathbf{p}_r$, and $\mathbf{p}_n$ progress along the intersection curve in sequence, $\tilde{\mathbf{t}}_{\mathbf{p}_r}$ is close to $\mathbf{t}_{\mathbf{p}_r}$. Otherwise, it deviates significantly. If the angle between them is greater than 45° and less than 135°, we determine that a reverse has occurred at $\mathbf{p}_r$. Note that if the consecutive points $\mathbf{p}_b\mathbf{p}_r\mathbf{p}_n$ that exhibit reversal are collinear, $\tilde{\mathbf{t}}_{\mathbf{p}_r}$ is almost degenerate. In such a case, we directly detect the reversal, avoiding the angle comparisons.

For a reversal point $\mathbf{p}_r$, we remove its next point and reconnect the curve (Fig. 15(c)). The new tangent of the polyline at $\mathbf{p}_r$ is then calculated and compared with $\mathbf{t}_{\mathbf{p}_r}$. We repetitively remove the next point $\mathbf{p}_n$ and reconnect the curve until there is no reversal. If no such next point can be found, we remove $\mathbf{p}_r$ and restart the correction from $\mathbf{p}_b$ instead of $\mathbf{p}_r$.

4.5.4 Removing illegal intersections. The self-intersections that may occur during surface discretization and the new intersections that may arise during mesh updating can lead to topological inconsistencies between the mesh and the B-Rep model, although such occurrences are rare. We detect these illegal intersections and perform local refinement. Since the input B-Rep model has no self-intersections, these illegal intersections are eliminated.

4.5.5 Handling coplanarity. When conducting Boolean operations on B-Rep models, the coplanarity between the surfaces of the two models is a commonly encountered degenerative case. As our discretization method does not maintain coplanarity in triangle meshes because of floating-point error introduced in discretization, it may lead to incorrect Boolean operation results. Therefore, it is necessary to check coplanar planes and perform 2D Boolean operations before mesh discretizations. Two coplanar planes will be segmented into three parts after a Boolean operation in 2D, as illustrated in Fig. 16. The overlapping part is replaced by a trimmed common planar surface, and identical meshes are generated for both models in this part. The boundaries of the common surface are regarded as intersection curves between the two models, and thus the Boolean operations can be conducted.

5 RESULTS AND DISCUSSION

The proposed algorithm is implemented in C++. The mesh intersection module is partially based on [Cherchi et al. 2022], and the constrained Delaunay triangulation is based on CGAL [The CGAL Project 2024]. Our algorithm can work on all watertight manifold B-Rep models, regardless of the types of surfaces involved. All the results shown in the paper were performed on a Windows system with an Intel i9-13900 CPU and 64GB RAM. The distance tolerance in our paper is uniformly set to d_p and is specified as 10^{-7} in our experiments. The angular tolerance is uniformly set to 10^{-6} rad.

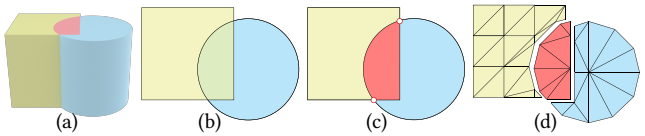


Fig. 16. The top surfaces of the two models in (a) are coplanar. (b) is a 2D schematic of the top surfaces. In (c), the intersection points of the 2D curves are calculated, and a 2D Boolean operation is performed, segmenting two surfaces into three parts. In (d), the intersection part is replaced by one trimmed common planar surface for both models, and identical meshes are generated in this part. The common part and the other two parts share identical sampling points on their boundaries.

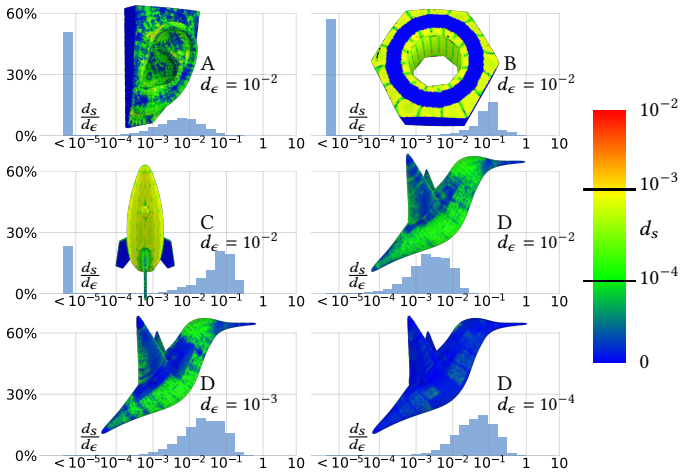


Fig. 17. Surface-to-mesh distance plots for models A, B, C, and D under varying d_ϵ values. 100k points were sampled uniformly on the models, and the histograms show the distribution of the ratios between the point-to-mesh distances and the surface-to-mesh distance threshold.

To test our method, we construct the following test sets: Set A consists of 10,000 pairs of models from the ABC dataset [Koch et al. 2019], each randomly rotated. Set B contains 100 pairs of complex models, either with a large number of facets or primarily composed of curved surfaces. For each pair of models $\{A, B\}$, we perform four operations A∪B, A∩B, A\B, and B\A and record the performance of our algorithm. *The source code and the dataset will be released publicly upon acceptance.*

5.1 Mesh accuracy validation

Our discretization method guarantees the surface-to-mesh distance. To validate it, we generate meshes for different models, with varying d_ϵ values, and uniformly sample 100k points on each model to calculate the surface-to-mesh distance. For each sampling point $\mathbf{s}$, its distance to the mesh is denoted as d_s. The input surface-to-mesh distance threshold d_ϵ is defined as in Equation 4.1.2. We record the values of d_s/d_ϵ in the histogram shown in Figure 17, and simultaneously visualize the distribution of d_s in a color map. The results show that our method guarantees the surface-to-mesh distance within d_ϵ.

It was observed that when $d_\epsilon = 10^{-2}$, the d_s values for Model D were less than $0.1d_\epsilon$. This model consists of 312 bicubic B-spline surfaces, and most of the surfaces have 7 to 9 knots in the u and v parameter directions. Since we first split the B-spline surfaces into Bézier surfaces at the knot vectors, the resolution of these surfaces is at least 7×7, and for most surfaces, the resolution of 7×7 is sufficient to ensure a maximum distance of $0.1d_\epsilon$. As d_ϵ decreases, the distance distribution becomes more dispersed, and the proportion of points with distances less than $0.1d_\epsilon$ decreases from 100% to 34.2%. It can also be observed that, despite the decrease in d_ϵ, our constraint on the surface-to-mesh distance remains conservative.

For the 100k samples on the four models at different d_ϵ, we recorded the maximum ratio between the point-to-mesh distance and d_ϵ, and the average distance of all sampled points, as shown in Table 1. No sampling point has a d_s greater than d_ϵ, and the average distance from all sampling points to the mesh is much smaller than d_ϵ. Therefore, our error-bounded discretization serves as the foundation for ensuring the conservative intersection detection discussed in Sec. 4.2.1.

Table 1. The input d_ϵ, the maximum and the average d_s/d_ϵ of the 100k sampling points on the four models shown in Fig. 17.

Model	d_ϵ	**Max** d_s/d_ϵ	**Avg.** d_s
A	10^{-2}	0.36	0.9×10^{-4}
B	10^{-2}	0.79	7.4×10^{-4}
C	10^{-2}	0.39	9.7×10^{-4}
D	10^{-2}	0.051	0.6×10^{-4}
D	10^{-3}	0.52	0.5×10^{-4}
D	10^{-4}	0.78	1.2×10^{-5}

5.2 Precision assessment of intersection curves

To evaluate the accuracy of our intersection curve calculations, we use ACIS [2024] as a benchmark and compare the distance between the intersection curves generated by our algorithm and those produced by ACIS. As mentioned in Sec. 4.3.4, we used d_p to control both the intersection point accuracy and the intersection curve precision. In our experiments, we set the test to ensure sufficient precision for the intersection points. Since we represent intersection curves as polylines during the Boolean operation, we calculated the distance between the curves generated by ACIS and the polylines produced by our algorithm.

We selected 100 pairs of models with valid ACIS Boolean operation results and calculated their intersection curves. For each curve, we uniformly sampled 10k points to compute the average and maximum distances from the ACIS curve to our polylines. In contrast, we projected our intersection points onto the ACIS curve and recorded the corresponding average and maximum distances. The results are summarized in Table 2, where the distance values are relative to the diagonal length of the AABB of the models.

Table 2. The maximum and the average distances from our intersection points to the ACIS intersection curves, as well as from 10k sampled points on the ACIS intersection curves to our intersection polylines, are recorded.

Distance	**Maximum**	**Average**
Ours to ACIS	4.06×10^{-7}	6.82×10^{-12}
ACIS to Ours	9.40×10^{-5}	1.37×10^{-7}

As shown in Table 2, the distance from ACIS curve to our polyline is larger than the distance from our polyline to ACIS curve because we approximate each curve using a polyline. However, the maximum distance, 9.40×10^{-5}, is less than the tolerance commonly used in CAD software to consider that two edges of the surface coincide. These results confirm the accuracy of our approach.

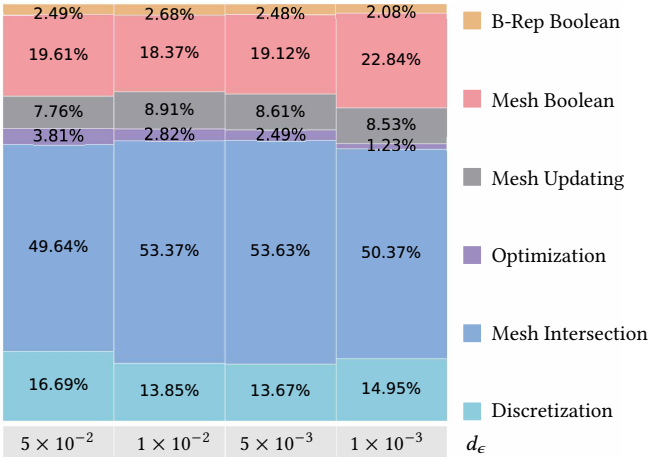


Fig. 18. The time distribution across different steps of the pipeline for various d_ϵ values based on Set B. The results show that Mesh Intersection consumes the most time, followed by Mesh Boolean.

Table 3. The total runtime and the number of failures for performing 4 operations A∪B, A∩B, A\B, and B\A on Set B under different d_ϵ values. Additionally, we recorded the total number of mesh triangles and the number of cases requiring enhancing mesh resolution as mentioned in Sec. 4.5.2.

d_ϵ	#Tri.	#Enhance Res.	Time(s)	#Fail
5×10^{-2}	13.2M	14	101.85	0
3×10^{-2}	14.0M	11	81.88	0
1×10^{-2}	15.8M	4	65.81	0
7×10^{-3}	17.4M	4	79.14	0
5×10^{-3}	20.0M	3	90.54	0
3×10^{-3}	27.2M	2	113.01	0
1×10^{-3}	60.2M	1	192.22	0

5.3 Efficiency analysis

We analyzed the time proportion of each part in the algorithm. Our algorithm is roughly divided into six parts, namely mesh discretization (Sec. 4.1), mesh intersection (Sec. 4.2), optimization (Sec. 4.3), mesh updating (Sec. 4.4.1), mesh Boolean and B-Rep Boolean (Sec. 4.4.2). Fig. 18 illustrates how each of these parts impacts the overall execution time at different d_ϵ values, where the time for each part represents the average time to perform intersection, union and subtraction operations on Set B.

It can be seen that the time distribution across different parts is minimally affected by d_ϵ. The most time-consuming part is *Mesh Intersection* based on the implementation of [Cherchi et al. 2022], which accounts for approximately half of the total time, even when the mesh is relatively rough at $d_\epsilon = 0.05$. The next most time-consuming part is *Mesh Boolean*. Together, these two components take up more than 70% of the total time. Therefore, the time required for our algorithm to perform Boolean operations on B-Rep models does not exceed twice the time taken for Boolean operations on the corresponding triangle meshes.

Meanwhile, we recorded the total runtime under different d_ϵ values, along with the total number of mesh triangles and the cases requiring increased mesh resolution for repeated intersections as described in Sec. 4.5.2. These results are presented in Table 3, from which we can observe that when d_ϵ is relatively large, such as $5{\times}10^{-2}$, the generated mesh contains fewer triangles. In this case, the need for multiple instances of enhanced resolution arises, requiring additional mesh intersections, which prevents a reduction in the total runtime. Conversely, when d_ϵ is too small, such as 1×10^{-3}, the generated mesh contains more triangles. Although the need for enhanced resolution is significantly reduced, the increased time required for the initial mesh intersection causes the total runtime to rise noticeably.

We observe that when $d_\epsilon > 1 \times 10^{-2}$, the number of cases requiring resolution enhancement decreases significantly as d_ϵ decreases. However, when $d_\epsilon < 1 \times 10^{-2}$, this decrease becomes negligible, indicating that $d_\epsilon = 1 \times 10^{-2}$ is sufficiently accurate for most scenarios. Therefore, we set $d_\epsilon = 1 \times 10^{-2}$ in subsequent experiments and applications, achieving an average runtime of 0.1645s for each Boolean operation on each pair of models.

Table 4. The total runtime for union, intersection, and subtraction on 100 pairs of examples under different d_ϵ values.

d_ϵ	A∪B(s)	A∩B(s)	A\B(s)	B\A(s)
1×10^{-2}	16.49	16.53	16.37	16.42
5×10^{-3}	22.92	22.52	22.53	22.57
1×10^{-3}	48.43	47.86	47.99	48.21

We also recorded the execution time for union, intersection, and subtraction in Set B with different values of d_ϵ, as shown in Table 4. At the same d_ϵ, the runtime for the three operations is nearly identical because their differences in the pipeline are limited to labeling during the mesh Boolean stage and the surface recovery during the B-Rep Boolean stage. The former requires the same time for all operations, while the latter is proportional to the number of surfaces to be recovered. However, as shown in Fig. 18, the surface Boolean stage accounts for less than 3% of the total runtime, resulting in no significant runtime differences among the three operations.

5.4 Runtime comparison

We performed union, intersection, and subtraction operations on Set A and Set B, where subtraction was performed as both A\B and B\A, to test the four methods: OCCT, Rhino, ACIS, and our method.

The time distribution of single Boolean operations on Set A is shown in Fig. 19. Each model pair in Set A has varying complexity and number of faces, and is constructed by randomly selecting and rotating models from the ABC dataset. All pairs are ensured to intersect by construction. We construct the dataset this way to be closer to real-world Boolean operations. Compared to open-source OCCT, we achieve a 17× speedup. Compared to commercial ACIS and Rhino, we achieve speedups of 2.3×. Moreover, the time distribution of our method is more concentrated.

Set B consists of 100 pairs of complex models, designed to evaluate the robustness and stability in efficiency of our method on challenging cases. The total time, maximum time, average memory usage and maximum memory usage for Set B are recorded in Table 5. We also recorded the number of failed Boolean operations

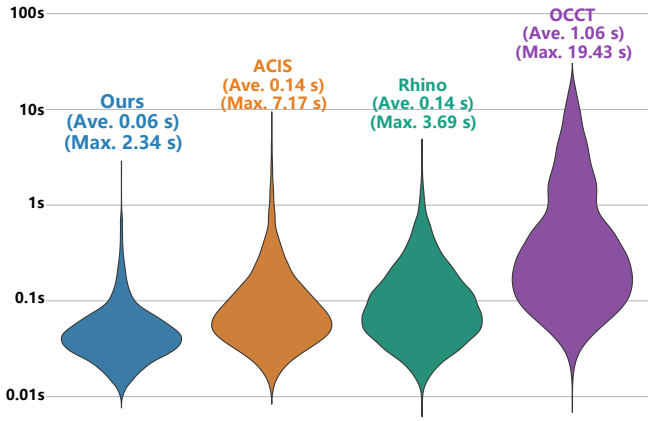


Fig. 19. Violin plots of the timing distributions for various methods on 10,000 pairs of models from the ABC dataset. The average and maximum time for a single Boolean operation are also recorded.

Table 5. The total runtime and memory usage for the four operations on Set B using OCCT, Rhino, ACIS, and our algorithm. Additionally, the longest runtime and highest memory usage for a single operation, as well as the number of failed cases, are recorded.

Method	Time(s)	Ave. Mem.(GB)	Max. Time(s)	Max. Mem.(GB)	#Fail
OCCT	1344.08	0.124	52.38	1.19	10
Rhino	224.02	0.044	8.99	0.21	8
ACIS	136.60	0.032	9.55	0.11	1
Ours	65.81	0.123	1.59	0.58	0

of each method. The histograms of memory usage and runtime distributions are shown in Figs. 21. Our method shows superior stability at runtime. The longest time for a single operation is only 1.59s, which is about $\frac{1}{6}$ for ACIS and Rhino, and about $\frac{1}{30}$ for OCCT. From Fig. 21, it can be seen that more than half of the examples are processed within 0.1s using our algorithm, and 94% of the examples are processed within 0.4s. In comparison, only 80.5%, 65.25%, and 31% of the examples are processed within the same time frame by ACIS, Rhino, and OCCT, respectively. Rhino and OCCT show a large deviation in runtime, with 7.75% and 38% of the examples requiring more than 1.6s to process, resulting in noticeable lag during usage.

In terms of memory usage on Set B, as shown in Table 5, our average memory usage is comparable to OCCT but higher than ACIS and Rhino. When processing complex models, our maximum memory usage is half that of OCCT. The higher memory usage is due to the fact that we store surface information, subdivided Bézier surface information, and mesh data simultaneously.

For the cases labeled "Fail" in Table 5, we selected two of those where OCCT and Rhino failed, as shown in Fig. 23. Only Boolean intersection results are presented, but errors also occur in union and subtraction operations. In the first case, the Boolean intersection should yield 28 closed solids. OCCT missed one narrow face, resulting in a non-watertight outcome with naked edges marked by red curves. Rhino produced only 27 solids, one missing in the area marked by a red circle. In the second case, the Boolean intersection should yield 54 closed solids. OCCT produced only 53 solids, missing one in the region marked by a red circle. Rhino returned an empty result. The case where ACIS failed is shown in Fig. 1. Our method can reliably identify all faces, no matter how small or intricate. More test examples are shown in Fig. 25.

To evaluate our capability to handle complex models composed of a large number of surfaces, we fitted the intricate mesh "cow" as shown in Fig. 2 and performed Boolean operations between them. The fitted cow model consists of 405 B-Spline patches, which generate numerous small faces after trimming. Our method successfully computed the intersection curves using the increasing mesh resolution approach described in Sec. 4.5.2 after detecting local errors during the optimization process and used conservative intersection detection to locate small loops near the ears. We recorded the total time required for Boolean intersection, union, and subtraction. OCCT took 180.3s, Rhino failed after 16.9s, and ACIS failed after 15.1s, while our method completed the operations in only 2.3s.

We also tested the efficiency of Boolean operations on multiple bodies, as shown in Fig. 1 and Fig. 22. In Fig. 22(a) and (b), we evaluated the time required for Boolean subtraction operations involving hundreds of bodies. All four methods produced correct results, with our algorithm achieving the shortest runtime. In Fig. 22(c), we tested the time required for Boolean union operations on multiple bodies. Fig. 22(d) shows an example in which the model is progressively subtracted, which is a common scenario of using Boolean operations in interactive design. As our method always maintains bijectivity in the hybrid representations, we don't need to remesh before the next Boolean operations. Once again, our algorithm demonstrated the fastest performance.

5.5 Handling coplanarity

To evaluate the capability to handle coplanarity, we tested our algorithm using several models with coplanar surfaces, as shown in Fig. 24. We also demonstrate in Fig. 20 our ability to handle models that share identical planar surfaces.

In Fig. 24(a), the two models have local coplanar regions. We first perform a 2D Boolean operation on the coplanar pair of trim surfaces to obtain the overlapping region (the red area). Then, we discretize both models and ensure that the generated meshes in the overlapping region are identical. Afterward, the intersection is computed for the remaining parts, resulting in the correct output. Fig. 24(b) is a challenging example. By rotating an elliptical cylinder along its axis by $\frac{\pi}{24}$ each time, 24 elliptical cylinders with coplanar top and bottom faces were generated. We perform a a Boolean union on every two models sequentially. During this process, the

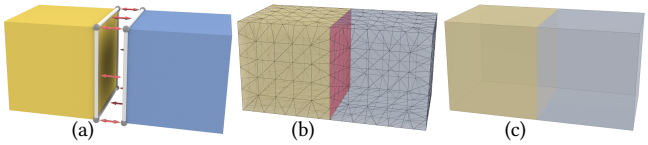
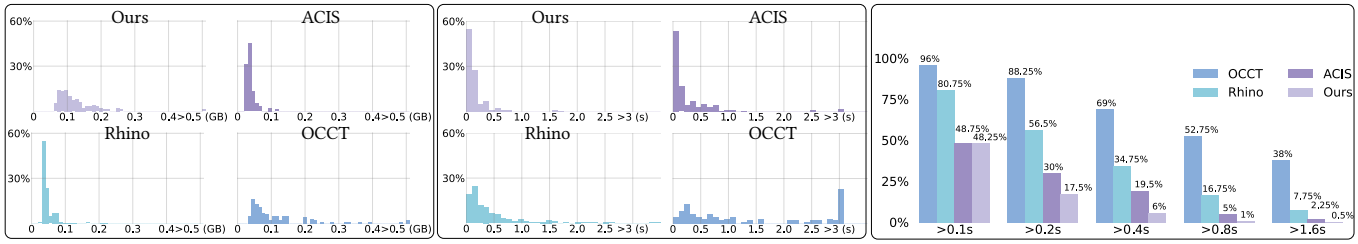
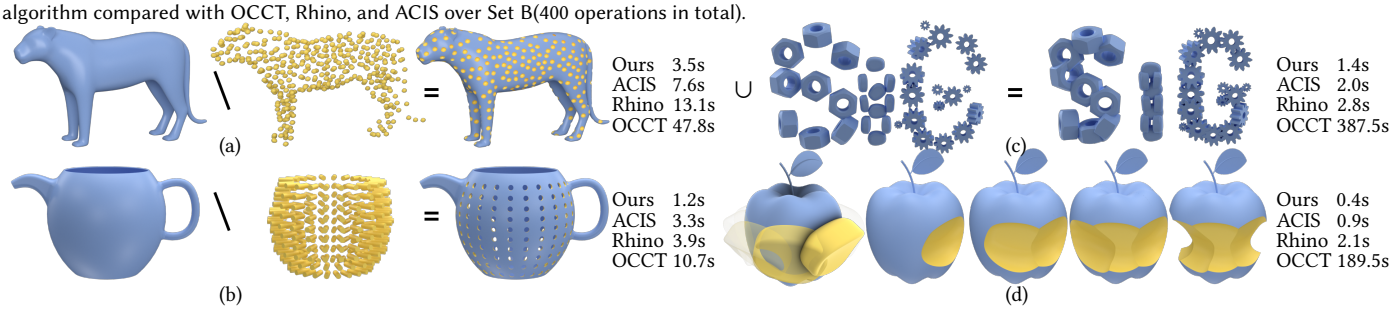


Fig. 20. In the 2D Boolean stage, we detect overlapping edges and vertices. Coincident edges and points are merged (a), resulting in identical mesh vertices at these locations (b), thus yielding exactly the same mesh in the coplanar regions and ensuring correct results after mesh Boolean (c).



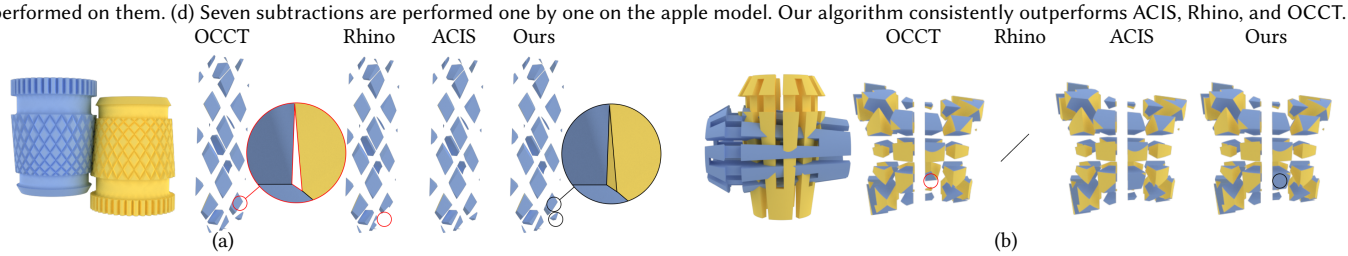
(a) Memory Usage(GB) (b) Time(s) (c) Time Distribution

Fig. 21. Histograms of memory usage and runtime for different methods, along with the percentage of Boolean operations exceeding a certain runtime for our algorithm compared with OCCT, Rhino, and ACIS over Set B(400 operations in total).



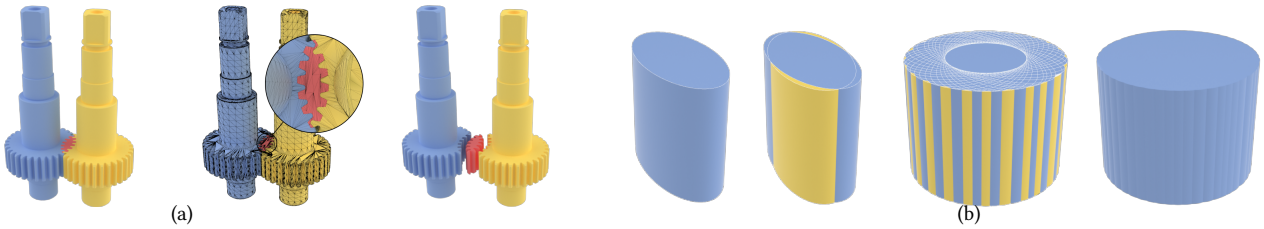
(a) (b) (c) (d)

Fig. 22. Our algorithm remains efficient when performing Boolean operations on multiple models. In (a), the model subtracts 400 balls, each consisting of 6 B-spline surfaces. In (b), the model subtracts 234 cylinders. (c) The letters S, I, and G consist of 9, 10, and 17 bodies, respectively, with Boolean union operations performed on them. (d) Seven subtractions are performed one by one on the apple model. Our algorithm consistently outperforms ACIS, Rhino, and OCCT.



(a) (b)

Fig. 23. Two examples where OCCT and Rhino failed. The symbol '/' indicates a failure return. Only the results of the Boolean intersection are shown as examples. The errors are highlighted with red circles.



(a) (b)

Fig. 24. (a) We perform a 2D Boolean operation on the coplanar parts, generating a boundary-consistent mesh in the coplanar region and obtaining correct Boolean operation results. (b) Boolean union of 24 coplanar elliptical cylinders.

coplanar portions produce a large number of fragments, making it challenging to construct the map between the coplanar solids and the coplanar mesh for our algorithm. Ultimately, our algorithm successfully obtains the correct Boolean union result.

5.6 Detecting small loops

We tested the effectiveness of our conservative intersection detection described in Sec. 4.2.1 on detecting small loops. As shown in Fig. 26, we performed Boolean intersection operations on two pairs of models with extremely small intersection loops. In our algorithm, due to the minimal size of the contact regions, the meshes of these two pairs of models did not intersect directly. However, when using the octree to determine potential mesh intersections, we identified regions where surface intersections might exist based on the normal estimation and distance evaluation shown in Fig. 9, ultimately obtaining the intersection curves. For these two examples in Fig. 26, the Boolean intersection results in Rhino and OCCT are both empty sets. ACIS successfully computed the intersection for only one of the cases. In contrast, our algorithm consistently produced all the intersection results.

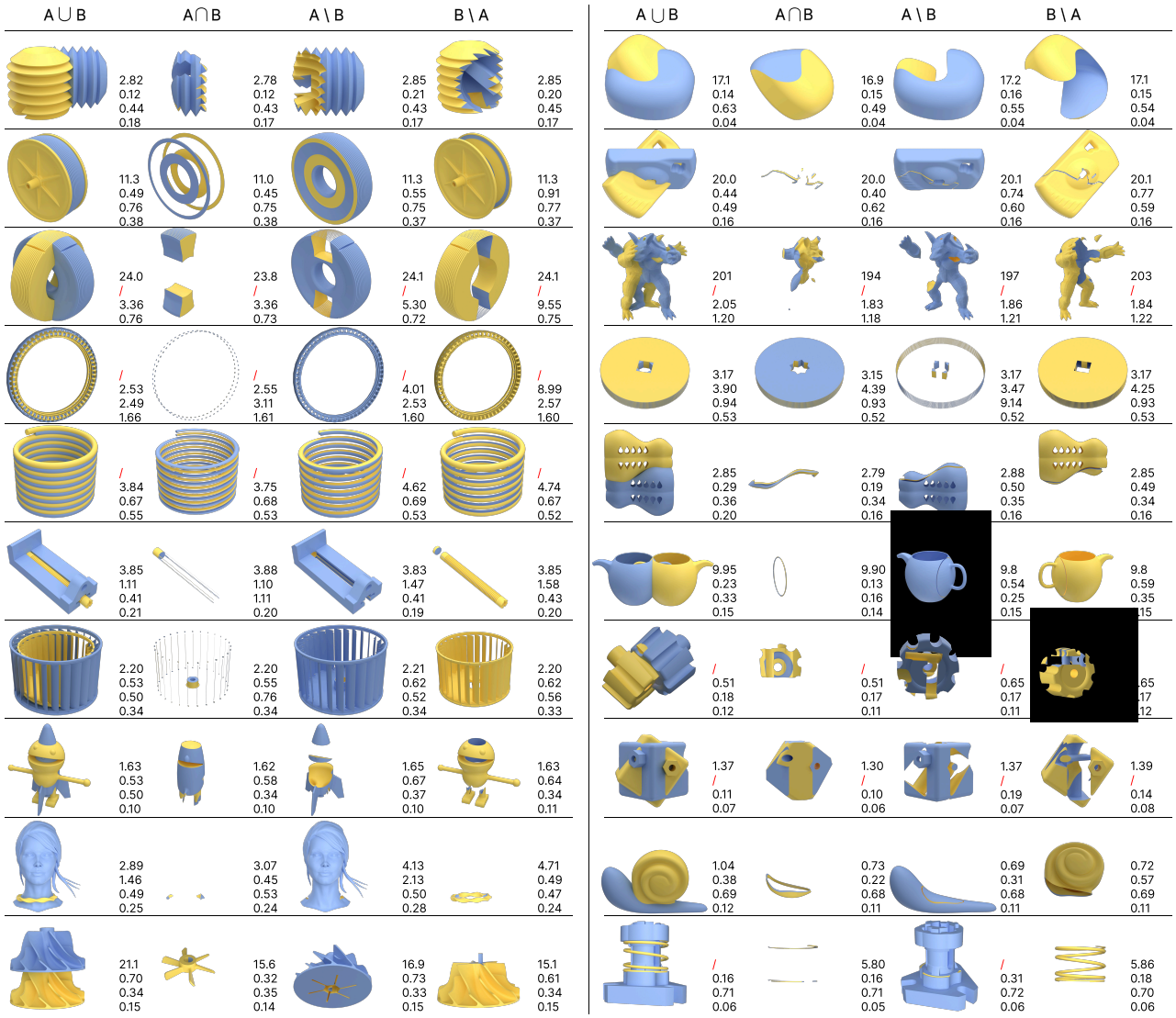


Fig. 25. 20 examples where competitors encounter errors or low efficiency. The four numbers listed in the right-bottom of each figure are the runtime in seconds (*s*) for OCCT, Rhino, ACIS and our method. '/' represents incorrect results, including missing facets, extra facets, or returning a failure.

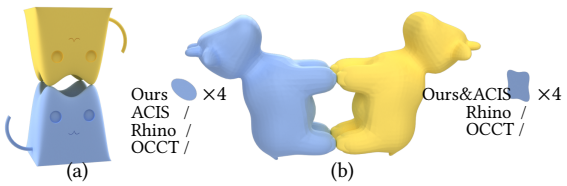
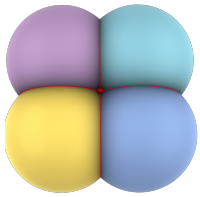


Fig. 26. Small Loop Detection. Both examples contain extremely small intersection loops. In (a), only our algorithm identifies all intersections, while in (b), both our algorithm and ACIS find all intersections. / indicates an empty set is returned. Our conservative intersection detection ensures that all intersection loops are reliably detected and handled.

6 Conclusion and Future Work

We propose a novel algorithm for performing Boolean operations on B-Rep models using hybrid representation. To the best of our knowledge, no published methods can achieve the same level of efficiency and stability as our approach. Our algorithm achieves over one order of magnitude acceleration compared to the only open-source geometry engine, OCCT. Additionally, it provides a speedup of 2 ∼ 3× when compared to the commercial engine ACIS and commercial software Rhino, with significant improvement in robustness. In cases where our competitors produce incorrect results, our algorithm is able to return the correct results.

There are still limitations in our method, which can be left as future work. For Boolean operations between multiple bodies, our



algorithm processes between every two models and currently cannot handle them simultaneously. The limitation lies in the intersection optimization step of our pipeline, which needs to account for the common intersection points of multiple bodies; see inset. Furthermore, handling multiple bodies introduces more complicated degenerations, which will need to be carefully addressed in future work. Another limitation is that we have not yet achieved interactive rates for B-Rep Boolean operations, although mesh Boolean operations can achieve interactive speeds for models with fewer than 200k triangles. This is because our models, on average, generate more than 150k triangles and require additional processing on the B-Rep models. In addition, we can only guarantee the correctness of Boolean operations when there are no small loops whose size and distance are both smaller than the mesh resolution. We create one seed point for each pair of non-intersected triangles within d_ϵ, and thus we may miss intersections if in a very rare case where there are multiple small loops in the corresponding region. Using local subdivision can detect all loops, but it may take too many levels of subdivision to do so [Krishnan and Manocha 1997].

Acknowledgments

The work is supported by the Key Project of the National Natural Science Foundation of China (12494550, 12494551, and 12494553), the Beijing Natural Science Foundation (Z240002), the NSFC-FDCT Joint Project (62461160259), and the Strategic Priority Research Program of the Chinese Academy of Sciences (XDB0640000 and XDB0640200).

References

Robert McNeel & Associates. 2024. Rhinoceros (Version 8.0). https://www.rhino3d.com/.

Marco Attene. 2020. Indirect Predicates for Geometric Constructions. *Comput. Aided Des.* 126 (2020), 102856.

Chandrajit L. Bajaj, Christoph M. Hoffmann, Robert E. Lynch, and John E. Hopcroft. 1988. Tracing surface intersections. *Comput. Aided Geom. Des.* 5, 4 (1988), 285–307.

Gilbert Bernstein and Don Fussell. 2009. Fast, Exact, Linear Booleans. *Comput. Graph. Forum* 28, 5 (2009), 1269–1278.

Henning Biermann, Daniel Kristjansson, and Denis Zorin. 2001. Approximate Boolean operations on free-form solids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001*. 185–194.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. (TOG)* 28, 3 (2009), 77:1–77:10.

IC Braid. 1974. *Designing with Volumes*. Ph. D. Dissertation. Cambridge University.

IC Braid. 1975. The synthesis of solids bounded by many faces. *Commun. ACM* 18, 4 (1975), 209–216.

Marcel Campen and Leif Kobbelt. 2010. Exact and Robust (Self-)Intersections for Polygonal Meshes. *Comput. Graph. Forum* 29, 2 (2010), 397–406.

M.S. Casale and J.E. Bobrow. 1989. A set operation algorithm for sculptured solids modeled with trimmed patches. *Comput. Aided Geom. Des.* 6, 3 (1989), 235–247.

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and Robust Mesh Arrangements Using Floating-Point Arithmetic. *ACM Trans. Graph. (TOG)* 39, 6 (2020), 250:1–250:16.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. 2022. Interactive and Robust Mesh Booleans. *ACM Trans. Graph. (TOG)* 41, 6 (2022), 248:1–248:14.

Lorenzo Diazzi, Daniele Panozzo, Amir Vaxman, and Marco Attene. 2023. Constrained Delaunay Tetrahedrization: A Robust and Practical Approach. *ACM Trans. Graph. (TOG)* 42, 6 (2023), 181:1–181:15.

Yohan D Fougerolle, Andrei Gribok, Sebti Foufou, Frédéric Truchetet, and Mongi A Abidi. 2005. Boolean operations with implicit and parametric representation of primitives using R-functions. *IEEE Trans. Vis. Comput. Graph.* 11, 5 (2005), 529–539.

ÁL García, J Ruiz de Miras, and FR Feito. 2011. Evaluation of Boolean operations between free-form solids using extended simplicial chains and PN triangles. *The Visual Computer* 27, 6–8 (2011).

Jia-Peng Guo and Xiao-Ming Fu. 2024. Exact and Efficient Intersection Resolution for Mesh Arrangements. *ACM Trans. Graph. (TOG)* 43, 6 (2024), 165:1–165:14.

Xiaohong Jia, Kai Li, and Jinsan Cheng. 2022. Computing the Intersection of Two Rational Surfaces Using Matrix Representations. *Comput. Aided Des.* 150 (2022), 103303.

John Keyser, Tim Culver, Mark Foskey, Shankar Krishnan, and Dinesh Manocha. 2004. ESOLID—a system for exact boundary evaluation. *Comput. Aided Des.* 36, 2 (2004), 175–193.

Payam Khanteimouri and Marcel Campen. 2023. 3D Bézier Guarding: Boundary-Conforming Curved Tetrahedral Meshing. *ACM Trans. Graph. (TOG)* 42, 6 (2023), 182:1–182:19.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 9601–9611.

Shankar Krishnan and Dinesh Manocha. 1997. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Trans. Graph. (TOG)* 16, 1 (1997), 74–106.

S Krishnan, D Manocha, M Gopi, T Culver, and J Keyser. 2001. BOOLE: A Boundary Evaluation System for Boolean Combinations of Sculptured Solids. *International Journal of Computational Geometry & Applications* 11, 01 (2001), 105–144.

Marco Livesu. 2019. Cinolib: A Generic Programming Header Only C++ Library for Processing Polygonal and Polyhedral Meshes. *Transactions on Computational Science XXXIV* (2019), 64–76.

Bruno Lévy. 2024. Exact predicates, exact constructions and combinatorics for mesh CSG. arXiv:arXiv:2405.12949

Martin Marinov and Leif Kobbelt. 2004. Optimization techniques for approximation with subdivision surfaces. *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications*, 113–122.

Julius Nehring-Wirxel, Philip Trettner, and Leif Kobbelt. 2021. Fast Exact Booleans for Iterated CSG using Octree-Embedded BSPs. *Comput. Aided Des.* 135 (2021), 103015.

Darko Pavic, Marcel Campen, and Leif Kobbelt. 2010. Hybrid Booleans. *Comput. Graph. Forum* 29, 1 (2010), 75–87.

Les Piegl and Wayne Tiller. 1996. *The NURBS book*. Springer Science & Business Media.

Hartmut Prautzsch and Leif Kobbelt. 1994. Convergence of subdivision and degree elevation. *Advances in Computational Mathematics* 2, 1 (1994), 143–154.

Aristides A G Requicha and Herbert B Voelcker. 1985. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proc. IEEE* 73, 1 (1985), 30–44.

Open Cascade SAS. 2024. Open CASCADE Technology. https://dev.opencascade.org/.

Thomas W Sederberg, H N Christiansen, and S Katz. 1989. Improved test for closed loops in surface intersections. *Comput. Aided Des.* 21, 8 (1989), 505–508.

Bin Sheng, Bowen Liu, Ping Li, Hongbo Fu, Lizhuang Ma, and Enhua Wu. 2018. Accelerated robust Boolean operations based on hybrid representations. *Comput. Aided Geom. Des.* 62 (2018), 133–153.

Spatial Team. 2024. 3D ACIS Modeler. https://www.spatial.com/products/3d-acis-modeling.

The CGAL Project. 2024. *CGAL User and Reference Manual* (6.0.1 ed.). https://doc.cgal.org/6.0.1/Manual/packages.html

Hiroshi Toriya, Teiji Takamura, Toshiaki Satoh, and Hiroaki Chiyokura. 1989. Boolean Operations of Solids with Free-From Surfarces Through Polyhedral Approximation. *New Advances in Computer Graphics*, 405–420.

Philip Trettner, Julius Nehring-Wirxel, and Leif Kobbelt. 2022. EMBER: exact mesh booleans via efficient & robust local arrangements. *ACM Trans. Graph. (TOG)* 41, 4 (2022), 39:1–39:15.

Charlie C.L. Wang. 2011. Approximate Boolean Operations on Large Polyhedral Solids with Partial Mesh Reconstruction. *IEEE Trans. Vis. Comput. Graph.* 17, 6 (2011), 836–849.

Gang Xu, Bernard Mourrain, Régis Duvigneau, and André Galligo. 2013. Optimal analysis-aware parameterization of computational domain in 3D isogeometric analysis. *Comput. Aided Des.* 45, 4 (2013), 812–821.

Fujio Yamaguchi and Toshiya Tokieda. 1984. A Unified Algorithm for Boolean Shape Operations. *IEEE Computer Graphics and Applications* 4, 6 (1984), 24–37.

Jieyin Yang, Xiaohong Jia, and Dong-Ming Yan. 2023. Topology Guaranteed B-Spline Surface/Surface Intersection. *ACM Trans. Graph. (TOG)* 42, 6 (2023), 211:1–211:16.

Yuanyan Ye, Yubo Wang, Juan Cao, and Zhonggui Chen. 2024. Watertight surface reconstruction method for CAD models based on optimal transport. *Computational Visual Media* 10, 5 (2024), 859–872.

Hanli Zhao, Charlie C. L. Wang, Yong Chen, and Xiaogang Jin. 2011. Parallel and efficient Boolean on polygonal solids. *The Visual Computer* 27, 6–8 (2011), 507–517.

Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph. (TOG)* 35, 4 (2016), 39:1–39:15.

A Theorems in Discretization

Theorem A.1. *The surface-to-mesh distance from $\mathbf{p}(u,v)$ to M is within a tolerance d_ϵ, if for each sub-patch $\mathbf{p}_i(u_i,v_i) \subseteq \mathbf{p}(u,v)$, there exists a planar convex polygon $P_i \in M$ such that the distances between the control points of $\mathbf{p}_i(u,v)$ and P_i are within d_ϵ.*

Proof. The proof of the above theorem is straightforward. The d_ϵ-envelope of the planar convex polygon P_i is a bounded convex space. The control points of $\mathbf{p}_i(u_i,v_i)$ lie in the envelope, thus the convex hull of the control points lies in the envelope, thus the patch $\mathbf{p}_i(u_i,v_i)$ is also contained in the envelope, due to the convexity of the convex hull and the envelope. □

Theorem A.2. *The surface-to-mesh distance from a rational Bézier sub-patch whose four corners are $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{v}_3$ to the triangle mesh M consists of two triangles $\triangle\mathbf{v}_0\mathbf{v}_1\mathbf{v}_3$ and $\triangle\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$ is within a tolerance d_ϵ, if the distances between the control points of the rational Bézier sub-patch and the planar quadrangle $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ is within ω, $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ is convex, and $\omega + d < d_\epsilon$, where $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$, d and ω are defined as in Fig. 3.*

Proof. According to Theorem A.1, the distance between a rational Bézier patch and $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ is no larger than ω, if the distances between the control points and $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ are within ω. For any point $\mathbf{w}$ on $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$, the distance between $\mathbf{w}$ and M is within d. As the distance between a point on the rational Bézier patch and $\square\mathbf{w}_0\mathbf{w}_1\mathbf{w}_2\mathbf{w}_3$ does not exceed ω, the distance between $\mathbf{w}$ and M does not exceed $\omega + d < d_\epsilon$. □

B Proof of Theorem 4.1

As shown in Fig. 27 (a), we assume that at a point $\mathbf{q} = \mathbf{p}(u_i, v_i)$ of a Bézier patch $\mathbf{p}(u,v)$, the two partial derivatives are

$$\frac{\partial \mathbf{p}(u,v)}{\partial v}|_{u=u_i} = \mathbf{v}_1 \in C_1, \frac{\partial \mathbf{p}(u,v)}{\partial u}|_{v=v_i} = \mathbf{v}_2 \in C_2.$$

As the two half cones C_1 and C_2 are on the same side of the plane $\tilde{\mathcal{P}}$, and $C_1 \cap C_2 = \mathbf{o}$, for $\forall \mathbf{v}_1 \in C_1$ and $\forall \mathbf{v}_2 \in C_2$, $\mathbf{v}_1 \times \mathbf{v}_2 \neq (0,0,0)^T$, thus the normal vector at $\mathbf{q}$ is $\mathbf{n}_i = \frac{\mathbf{v}_1 \times \mathbf{v}_2}{\|\mathbf{v}_1 \times \mathbf{v}_2\|}$. It can be observed that:

(1) As shown in Fig. 27 (a), the normal vector is determined by the plane $\mathcal{P}$ expanded by the two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, so it can be determined by the two vectors $\mathbf{v}_1', \mathbf{v}_2'$ in the plane and on the boundary of C_1 and C_2, respectively. This means that any normal vector of $\mathbf{p}(u,v)$ can be obtained by two vectors on the boundaries of C_1 and C_2.

(2) Let $\mathbf{v}_1$ be fixed, change $\mathbf{v}_2$, then $\mathcal{P}$ rotates against $\mathbf{v}_1$. The Gaussian image of the normal vectors is a segment of a great circle on a unit sphere. The two endpoints of the segment correspond to two extreme cases in which the planes $\mathcal{P}_1$ and $\mathcal{P}_2$ are tangential to C_2 at $\mathbf{v}_2^1$ and $\mathbf{v}_2^2$, respectively, as shown in Fig. 27 (b).

(3) For each $\mathbf{v}_1$, we find $\mathbf{v}_1'$ that is symmetrical to $\mathbf{v}_1$ w.r.t. the plane $span(\mathbf{a}_1, \mathbf{a}_2)$. Let $\mathbf{v}_1$ or $\mathbf{v}_1'$ be fixed, then the corresponding two Gauss maps are symmetrical to $\mathbf{d} = \mathbf{a}_1 \times \mathbf{a}_2 / \|\mathbf{a}_1 \times \mathbf{a}_2\|$, since the endpoints of one circular Gauss map are symmetrical to the other one w.r.t. $\mathbf{d}$. See Fig. 27 (c).

(4) In Fig. 27 (c), the deviation from $\mathbf{n}_1'$ to $\mathbf{d}$ is greater than that from $\mathbf{n}_1$ to $\mathbf{d}$. To prove this, we assume that $\mathbf{a}_1$ and $\mathbf{a}_2$ are unit vectors, $\mathbf{v}_1 = \mathbf{a}_1 + t_1\mathbf{d}$, $\mathbf{v}_1' = \mathbf{a}_1 - t_1\mathbf{d}$, $\mathbf{v}_2 = \mathbf{a}_2 + t_2\mathbf{d}$, $\mathbf{v}_2' = \mathbf{a}_2 + t_3\mathbf{d}$,

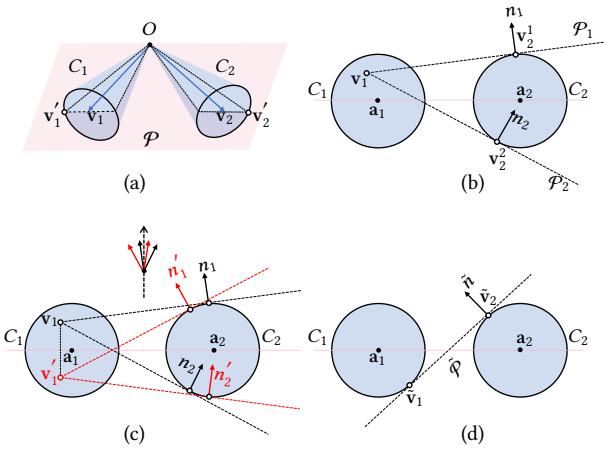


Fig. 27. The proof of Theorem 4.1. (a) shows that the normal vector orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$ totally depends on the plane $\mathcal{P}$ passing through $\mathbf{v}_1$ and $\mathbf{v}_2$. (b), (c) and (d) are the 2d illustrations of (a). For more detailed explanations please see the complete proof in Appendix B.

where $t_3, t_2 > 0, t_1 > 0$, and the angle between $\mathbf{a}_1$ and $\mathbf{a}_2$ is not larger than $\pi/2$. It is easy to see that $t_3 > t_2$. We note that the above assumptions do not lose generality, since for any $\mathbf{v}_1$, we can temporarily take $\mathbf{a}_1$ as the bisector of $\mathbf{v}_1$ and $\mathbf{v}_1'$, and if the angle between $\mathbf{a}_1$ and $\mathbf{a}_2$ is larger than $\pi/2$, we take the opposite half-cone of C_1 to ensure the assumption. Then $\mathbf{n}_1$ is in the direction of $\mathbf{v}_1 \times \mathbf{v}_2 = \mathbf{a}_1 \times \mathbf{a}_2 + t_1(\mathbf{d} \times \mathbf{a}_2) + t_2(\mathbf{a}_1 \times \mathbf{d}) = \mathbf{a}_1 \times \mathbf{a}_2 + t_1\mathbf{r}_1 + t_2\mathbf{r}_2$, where $\mathbf{r}_1$ and $\mathbf{r}_2$ are unit vectors orthogonal to $\mathbf{d}$. Similarly, $\mathbf{n}_1'$ is in the direction of $\mathbf{v}_1' \times \mathbf{v}_2' = \mathbf{a}_1 \times \mathbf{a}_2 - t_1\mathbf{r}_1 + t_3\mathbf{r}_2$. If $l_1 = \|t_1\mathbf{r}_1 + t_2\mathbf{r}_2\| < l_2 = \| - t_1\mathbf{r}_1 + t_3\mathbf{r}_2\|$, then the deviation from $\mathbf{n}_1'$ to $\mathbf{d}$ is larger than that from $\mathbf{n}_1$ to $\mathbf{d}$. We compute $l_2^2 - l_1^2 = (t_3 - t_2)(t_3 + t_2 - 2t_1\mathbf{r}_1\mathbf{r}_2)$. Since the angle between $\mathbf{a}_1$ and $\mathbf{a}_2$ is no larger than $\pi/2$, it is easy to check the angle between $\mathbf{r}_1$ and $\mathbf{r}_2$ is in $[\pi/2, \pi)$. Thus, $l_2^2 - l_1^2 > 0$, $\mathbf{n}_1'$ has a larger deviation from $\mathbf{d}$. As $\mathbf{n}_1'$ and $\mathbf{n}_2$ are symmetrical w.r.t. $\mathbf{d}$, the deviation from $\mathbf{n}_2$ to $\mathbf{d}$ is larger than that from $\mathbf{n}_1$ to $\mathbf{d}$.

Using (2) and (3), we can find $\mathbf{v}_1$ on the lower part of C_1's boundary, i.e., $\mathbf{v}_1 \cdot \mathbf{d} < 0$, such that the angle between $\mathbf{n}$ and $\mathbf{d}$ reaches maximal, as shown in Fig. 27 (d). For a fixed $\mathbf{v}_1$, since the Gauss map is a segment of a great circle, the maximal bias from the Gauss map to $\mathbf{d}$ happens on one of the endpoints, that is, when $\mathcal{P}$ and C_2 are tangential at $\mathbf{v}_2$. Using (4), we know that when $\mathbf{v}_2 \cdot \mathbf{d} > 0$ the deviation between $\mathcal{P}$ from the horizontal plane reaches maximum. Then similarly, with a fixed $\mathbf{v}_2$, $\mathbf{v}_1$ should also be on a tangential line. Thus, Theorem 4.1 is proved. □

C Implementation of the Newton's Method

We treat the parametric coordinates (u_0, v_0) and (s_0, t_0) of the intersection point $\mathbf{r}_0$ on the surfaces $S_A = \mathbf{s}_1(u,v)$ and $S_B = \mathbf{s}_2(s,t)$ as variables, and solve for the zero of $\mathbf{D}(u,v,s,t) = \mathbf{s}_1(u,v) - \mathbf{s}_2(s,t)$. The iteration equation is $\nabla\mathbf{D}\Delta_k = \mathbf{s}_2(s_k,t_k) - \mathbf{s}_1(u_k,v_k)$. We update the variables in the direction of the steepest descent of $\mathbf{D}$, which lies in the space spanned by the rows of $\nabla\mathbf{D}$. Let $\Delta_k = \nabla\mathbf{D}^T\mathbf{a}_k$, where $\mathbf{a}_k = (a_0, a_1, a_2)^T$ is a coefficient vector. The equation becomes: $\nabla\mathbf{D}\nabla\mathbf{D}^T\mathbf{a}_k = \mathbf{s}_2(s_k,t_k) - \mathbf{s}_1(u_k,v_k)$. The variables $\mathbf{x} = (u,v,s,t)^T$ in each iteration are updated as $\mathbf{x}_{k+1} = \mathbf{x}_k + \nabla\mathbf{D}^T\mathbf{a}_k$.