Topology Guaranteed B-Spline Surface/Surface Intersection

JIEYIN YANG, KLMM, Chinese Academy Of Sciences; University of Chinese Academy of Sciences, CHINA
XIAOHONG JIA[*], KLMM, Chinese Academy Of Sciences; University of Chinese Academy of Sciences, CHINA
DONG-MING YAN, MAIS, Institute of Automation, Chinese Academy of Sciences; School of AI, University of Chinese Academy of Sciences, CHINA

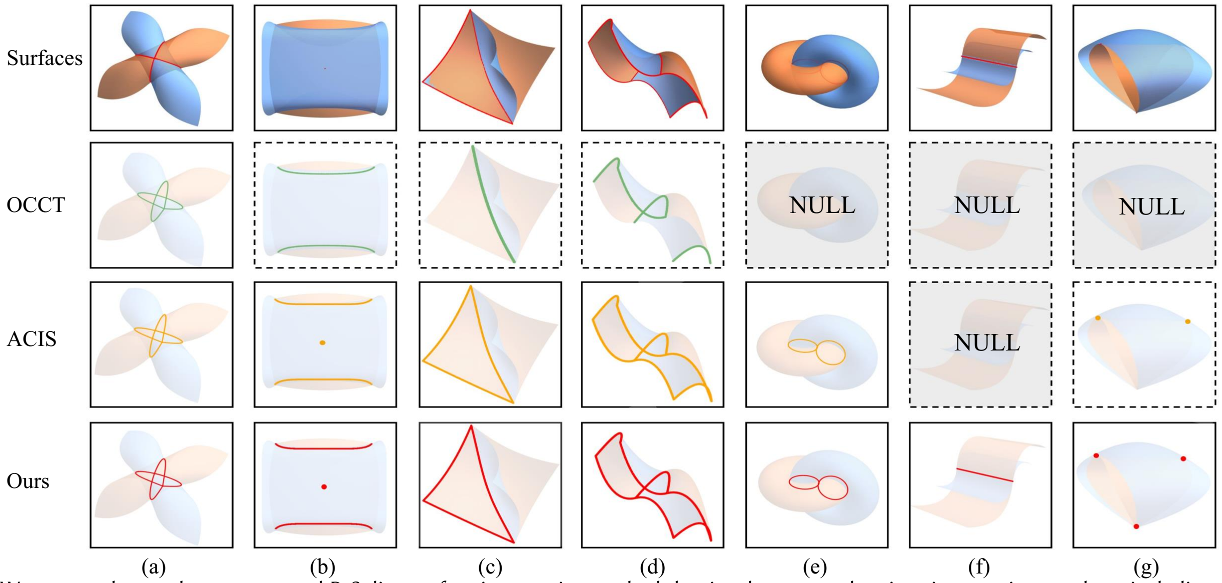


Fig. 1. We proposed a topology guaranteed B-Spline surface intersection method that is robust toward various intersection topology, including (a) cross intersections, (b) isolated contacts, (c)(d) intersections along boundaries, (e) contact in different intersection branches, (f) high-order contact along a whole curve and (g) multiple isolated contacts. The intersection loci of our method are presented in red curves, while the results of ACIS and OCCT are presented in yellow and green curves respectively. 'NULL' indicates that the result is an empty curve. The boxes with dashed boundary lines indicate wrong topology. Our method provides correct intersection topology in all cases, while ACIS and OCCT sometimes fail.

The surface/surface intersection technique serves as one of the most fundamental functions in modern Computer Aided Design (CAD) systems. Despite the long research history and successful applications of surface intersection algorithms in various CAD industrial software, challenges still exist in balancing computational efficiency, accuracy, as well as topology correctness. Specifically, most practical intersection algorithms fail to guarantee the correct topology of the intersection curve(s) when two surfaces are in near-critical positions, which brings instability to CAD systems. Even in one of the most successfully used commercial geometry engines ACIS, such complicated intersection topology can still be a tough nut to crack.

In this paper, we present a practical topology guaranteed algorithm for computing the intersection loci of two B-spline surfaces. Our algorithm well treats all types of common and complicated intersection topology with practical efficiency, including those intersections with multiple branches or cross singularities, contacts in several isolated singular points or high-order contacts along a curve, as well as intersections along boundary curves. We present representative examples of these hard topology situations that challenge not only the open-source geometry engine OCCT but also the commercial engine ACIS. We compare our algorithm in both efficiency and topology correctness on plenty of common and complicated models with the open-source intersection package in SISL, OCCT, and the commercial engine ACIS.

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

Additional Key Words and Phrases: Surface intersection, B-Spline surface, implicitization, boolean operation

ACM Reference Format:
Jieyin Yang, Xiaohong Jia, and Dong-Ming Yan. 2023. Topology Guaranteed B-Spline Surface/Surface Intersection. *ACM Trans. Graph.* 42, 6, Article 211 (December 2023), 16 pages. https://doi.org/10.1145/3618349

[*]Corresponding author.

Authors' addresses: Jieyin Yang, KLMM, Chinese Academy Of Sciences; University of Chinese Academy of Sciences, Beijing, CHINA, yangjieyin17@mails.ucas.ac.cn; Xiaohong Jia, KLMM, Chinese Academy Of Sciences; University of Chinese Academy of Sciences, Beijing, CHINA, xhjia@amss.ac.cn; Dong-Ming Yan, MAIS, Institute of Automation, Chinese Academy of Sciences; School of AI, University of Chinese Academy of Sciences, Beijing, CHINA, yandongming@gmail.com.

This work is licensed under a Creative Commons Attribution International 4.0 License.
© 2023 Copyright held by the owner/author(s).
0730-0301/2023/12-ART211
https://doi.org/10.1145/3618349

1 INTRODUCTION

Surface/Surface intersection is a fundamental task in Computer Aided Design (CAD) and geometric modeling systems. An efficient, robust, and topology guaranteed surface/surface intersection algorithm is highly desired in many modeling operations of CAD systems, such as Boolean operations, mesh generation, and rendering. Given the prevalent utilization of surface/surface intersection in CAD systems, research on this task extends beyond the single goal of efficiency. Consequently, an algorithm that not only fulfills the fundamental requirements but also exhibits robustness by adapting to diverse scenarios is highly required. Additionally, the algorithm must guarantee the topological correctness by effectively handling intricate situations, such as small loops and tangential situations.

There exists extensive research on surface intersection algorithms starting from the 1980s, during which several seminal algorithms gained significant attention, *i.e.* the algebraic method, the marching method, the lattice method, the subdivision method, and the hybrid method that combines two of these methods. Algebraic methods usually convert the surface intersection problem into solving algebraic equations, which can be challenging in certain complex scenarios. The marching method computes a series of starting points and then traces out the intersection locus using classic differential geometry [Bajaj et al. 1988a; Barnhill and Kersey 1990; Kriezis et al. 1992]. Nevertheless, a full computation of starting points on all branches, which directly determines the topology correctness of the intersection curve, is difficult to achieve. The lattice method decomposes the surface/surface intersection problem into curve/surface intersection problems by discretizing the parametric domain of the surface [Rossignac and Requicha 1987]. However, inadequate discretization may result in the loss of small loops and isolated points in the intersection curve. The subdivision method keeps dividing the surfaces into smaller patches and then converts the intersection problem into the intersection of these near-planar patches [Patrikalakis 1993], which may lead to the inaccurate calculation of tangential curves with insufficient number of subdivisions. Hybrid methods combine two of the above-discussed methods, for example, the algebraic method and the marching method, or the subdivision method and the marching method. Among all these methods, algebraic-related methods possess the advantage of preserving the intersection topology, which is difficult for the lattice or subdivision method. However, the algebraic method is generally inefficient and depends heavily on some other algebraic techniques, such as implicitization or solving polynomial systems.

Although extensive exploration has been conducted on the surface intersection problem, the existing methodologies can still be improved in terms of implementation. For example, some algebraic methods resort to surface implicitization or solving a system of polynomial equations, which involve two key steps in an intersection algorithm but are actually hard problems in algebraic geometry. Over the past decades, the implicitization and the polynomial system solving problems have been greatly developed in their own community; nevertheless, these developments were seldom brought back to the surface intersection literature [Hoffmann 1989; Sarraga 1983]. Additionally, records pertaining to the timing and topology of intersection examples are few, and a limited number of works [Jia et al. 2022; Lin et al. 2013] have performed comparative analyses to evaluate the performance of different methods.

The surface/surface intersection problem has not only been widely studied in academic research but rapidly developed in the industrial field. Many open-source and commercial software packages containing intersection functions have been developed and extensively employed in both research and industrial domains. Among these packages, ACIS [2023] is a renowned 3D modeling kernel that provides powerful capabilities for creating, manipulating, and analyzing geometric models. ACIS has established itself as one of the leading 3D modeling kernels in the CAD industry due to its robustness, versatility, and wide range of functionalities. The surface intersection function in ACIS has also become one of the industry standards due to its high robustness and efficiency. Nevertheless, when confronted with scenarios involving high-order contact or other complex intersection topology, the outcomes produced by ACIS are sometimes unsatisfactory as illustrated in Fig. 1.

In this paper, we present a practical topology guaranteed algorithm for computing the intersection of two B-Spline surfaces. The proposed algorithm demonstrates topology correctness not only in all common transversal examples of B-spline surface intersections as in ACIS but also in challenging intersection topologies, such as when the two surfaces are in high-order contact at some points or along a curve. We mainly adopt an algebraic routine in the algorithm by bringing novel techniques or improvements to state-of-the-art works in the key steps of implicitization, topology determination, and redundant curve clipping. Our main contributions are as follows:

(1) The fast moving plane technique is introduced to the traditional implicitization approach by Dixon matrices, which facilitates the intersection process even in the presence of base points.
(2) The topology of the intersection curve in the parametric domain is pre-computed to ensure the correctness of the adjacency relationships between critical points and boundary points, as well as considering the intersection along the boundaries of the parametric domain.
(3) A novel clipping method is proposed for the redundant part of the intersection curve caused by implicitization. The method is based on the inversion formula deduced from the Dixon matrix and successfully restricts the intersection curve to its required parametric domain.

2 RELATED WORK

Extensive research has been conducted on the problem of surface/surface intersection, and most of the proposed methods or approaches can be adapted to the B-Spline surface intersection problem [Bajaj et al. 1988b; Dokken 1997; Krishnan and Manocha 1997; Manocha and Canny 1991; Patrikalakis 1993; Sederberg and Meyers 1988; Shen et al. 2016; Ventura and Guedes Soares 2012]. These techniques can typically be classified into five categories: the algebraic method, the marching method, the lattice method, the subdivision method, and the hybrid method.

Algebraic methods. Given two B-Spline surfaces, algebraic methods typically convert one of them into its implicit equation and then substitute the parametric form of the other surface into this implicit equation. This process results in an algebraic curve with preserved algebraic characteristics [Manocha and Canny 1991; Sarraga 1983]. However, the implicitization of rational surfaces using the traditional algebraic method is known to be challenging [Busé et al. 2003]. Moreover, the parametric domain of the surface will be omitted after the implicitization, which leads to undesired intersection curves outside the parametric domain of surfaces during the intersection process. Sederberg and Chen [1995] were the first to discover the technique of moving surfaces for surface implicitization, where they mentioned the Dixon matrix as a possible candidate for implicitization. However, at that time, the Dixon matrix always failed to provide implicitization when the surface contained base points where the homogenous coordinates of the surface simultaneously vanish. Therefore, the technique they proposed was the first to use blending functions to construct a series of moving surfaces for implicitization, which was a different and novel technique. However, despite the disadvantage of the Dixon matrix of not being directly applicable to surfaces with base points, it remains the simplest method for implicitization, because it has much smaller matrix coefficients than other implicitization approaches.

Marching methods for surface intersection are widely used because of their generality and simplicity. Marching methods usually have two main steps. First, a series of starting points are determined on each branch of the intersection curves. Then, by using the local differential geometry of surfaces, the intersection locus is traced out from each starting point [Bajaj et al. 1988b; Barnhill and Kersey 1990; Patrikalakis 1993; Sederberg and Meyers 1988; Ventura and Guedes Soares 2012]. However, this approach also has some limitations, such as the necessity of identifying starting points, which can be challenging in the presence of small loops, and the determination of the tracing direction over singular points.

Lattice methods decompose the surface/surface intersection problem into multiple curve/surface intersection problems. In this approach, one of the surfaces is disintegrated into its isoparametric curves, which are then intersected with the other surface to determine their intersection points. The ultimate intersection curve of two surfaces is produced by linking the discrete intersection points [Rossignac and Requicha 1987]. Owing to the discretization of the lattice method, the inadequate sampling density of the isoparametric curves selected on the surface may result in missing isolated intersection points or small loops.

Subdivision methods are a recursive approach that involves subdividing the B-Spline surfaces into smaller patches until the intersection can be computed with sufficient accuracy. This approach consists of two iterative steps. First, each B-Spline surface is divided into a set of smaller patches by refining its control points. Then, an intersection test is performed between the patches of the two surfaces to determine possible intersections. The accuracy and efficiency of subdivision methods heavily depend on the choice of subdivision strategy and termination criteria [Lasser 1986; Sederberg and Meyers 1988]. [De Figueiredo 1996] utilized affine arithmetic to calculate the bounding box of small patches and then quickly discarded the pairs of patches with no box intersection. [Lin et al. 2013] accelerated the affine arithmetic intersection algorithm using a GPU, significantly enhancing its efficiency. Although subdivision methods can obtain intersection curves at relatively high speed, they may miss small loops and isolated intersection points, and the topological correctness near singular points is not guaranteed under this approach.

Hybrid methods pertain to the integration of two or more previously mentioned methods, with the objective of harnessing their distinct advantages. A common example is to use the subdivision method to find several starting points and then trace out the whole intersection curve using the marching method in order to enhance the efficiency of the intersection process [Barnhill and Kersey 1990; Dokken 1997]. Another representative approach involves utilizing algebraic methods to locate starting points, and subsequently employing marching methods to trace out the intersection curve from these calculated starting points [Krishnan and Manocha 1997]. The fusion of algebraic and tracing methods is frequently employed to strike a balance between the efficiency and topological accuracy of the intersection curve.

3 PRELIMINARIES

Before introducing the proposed intersection algorithm, we first introduce some preliminaries about base points, moving planes, and Dixon matrices.

3.1 Moving Planes

A *parametric surface* $\mathbf{P}$ in 3D space can be written in its homogeneous form

$$\mathbf{P}(u,v) = (a(u,v), b(u,v), c(u,v), d(u,v)),$$

where a, b, c, d are polynomials with $\gcd(a,b,c,d) = 1$, and $\left(\frac{a}{d}, \frac{b}{d}, \frac{c}{d}\right)$ give the surface coordinates in $\mathbb{R}^3$.

A parameter pair (u_0, v_0) is called a *base point* of $\mathbf{P}(u,v)$ if

$$a(u_0,v_0) = b(u_0,v_0) = c(u_0,v_0) = d(u_0,v_0) = 0. \tag{1}$$

A *moving plane* is a family of planes with parameters (u,v) in its coefficients in x, y, z, w written in the following form:

$$L(\mathbf{X}; u,v) := A(u,v)x + B(u,v)y + C(u,v)z + D(u,v)w = 0, \tag{2}$$

where $\mathbf{X} = (x,y,z,w) \in \mathbb{RP}^3$ is the homogeneous coordinate, and $A(u,v), B(u,v), C(u,v), D(u,v) \in \mathbb{R}[u,v]$ are bivariate polynomials. For each parameter value (u_0,v_0), $L(\mathbf{X}; u_0, v_0) = 0$ gives a plane.

A moving plane $L(\mathbf{X}; u,v) = 0$ is said to *follow* the parametric surface $\mathbf{P}(u,v)$ if the following equation holds:

$$L(\mathbf{P}(u,v); u,v) \equiv 0. \tag{3}$$

Geometrically, this indicates that for every parameter value (u_0,v_0), the plane $L(\mathbf{X}; u_0,v_0) = 0$ always passes through the point $\mathbf{P}(u_0,v_0)$ on the surface. For more basics on moving planes, readers can refer to [Sederberg and Chen 1995].

3.2 Dixon Matrices

Given three bivariate polynomials of degree (m,n):

$$f(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} a_{i,j}u^i v^j, g(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{i,j}u^i v^j, h(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{i,j}u^i v^j, \tag{4}$$

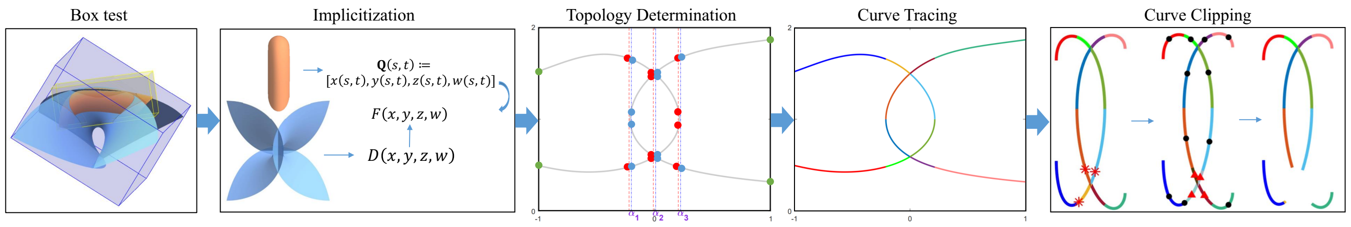


Fig. 2. Overview of the proposed algorithm. Surface $\mathcal{P}$ is represented in blue and Surface $\mathcal{Q}$ is represented in orange. First, a fast collision detection of the bounding boxes of the two surfaces is conducted. Then, if two bounding boxes collide with each other, Surface $\mathcal{P}$ is implicitized to $F(x, y, z, w)$ using its corresponding Dixon matrix $D(x, y, z, w)$. Subsequently, left helping points (red), right helping points (blue) and boundary points (green) are calculated to determine the topology of the intersection curve. These helping points then serve as the starting points for locus computation. After tracing the intersection curve in the parametric domain, we lift it to 3D space and employ the Dixon matrix to clip the 3D curve within the appropriate region.

the *Dixon matrix* $D(f, g, h)$ is a $2mn \times 2mn$ matrix given by the following equation [Dixon et al. 1908]:

$$\frac{1}{(u-\alpha)(v-\beta)}\begin{vmatrix} f(u,v) & g(u,v) & h(u,v) \\ f(u,\beta) & g(u,\beta) & h(u,\beta) \\ f(\alpha,\beta) & g(\alpha,\beta) & h(\alpha,\beta) \end{vmatrix} = (1, \alpha, \beta, \cdots, \alpha^{m-1}\beta^{2n-1}) D(f,g,h) \begin{pmatrix} 1 \\ u \\ v \\ \vdots \\ u^{2m-1}v^{n-1} \end{pmatrix}. \tag{5}$$

where α and β are two variables. The *Dixon resultant* of f, g, h is defined by the determinant $\det(D(f, g, h))$ of the Dixon matrix.

4 B-SPLINE SURFACE INTERSECTIONS

A *B-Spline surface* of bi-degree (m, n) is defined by $(k + 1) \times (l + 1)$ control points $\mathbf{p}_{i,j}$ and two knot vectors $\bar{\mathbf{U}} = \{u_0, u_1, \cdots, u_{m+k}\}$ and $\bar{\mathbf{V}} = \{v_0, v_1, \cdots, v_{n+l}\}$ [Gordon and Riesenfeld 1974]. The parametric representation of the B-Spline surface can be written as

$$\mathbf{R}(u, v) = \sum_{i=0}^{k}\sum_{j=0}^{l} N_{i,m}(u)N_{j,n}(v)\mathbf{p}_{i,j}, \tag{6}$$

where $N_{i,m}(u)$ and $N_{j,n}(v)$ are the spline basis functions that can be derived by the Cox-de Boor recursion formula [de Boor 1971].

Given two B-Spline surfaces $\mathcal{P}$ and $\mathcal{Q}$ of bi-degrees (m_1, n_1) and (m_2, n_2), with control points $\bar{\mathbf{P}}, \bar{\mathbf{Q}}$ and knot vectors $\bar{\mathbf{U}}_1,\bar{\mathbf{V}}_1$ and $\bar{\mathbf{U}}_2,\bar{\mathbf{V}}_2$:

$$\begin{aligned} &\bar{\mathbf{P}} = \{\mathbf{p}_{i,j} \in \mathbb{R}^3 \mid i = 0, 1, \cdots, k_1, \quad j = 0, 1, \cdots, l_1\}, \\ &\bar{\mathbf{U}}_1 = \{u_{1,0}, u_{1,1}, \cdots, u_{1,m_1+k_1}\} \quad \bar{\mathbf{V}}_1 = \{v_{1,0}, v_{1,1}, \cdots, v_{1,n_1+l_1}\}, \\ &\bar{\mathbf{Q}} = \{\mathbf{q}_{i,j} \in \mathbb{R}^3 \mid i = 0, 1, \cdots, k_2, \quad j = 0, 1, \cdots, l_2\}, \\ &\bar{\mathbf{U}}_2 = \{u_{2,0}, u_{2,1}, \cdots, u_{2,m_2+k_2}\} \quad \bar{\mathbf{V}}_2 = \{v_{2,0}, v_{2,1}, \cdots, v_{2,n_2+l_2}\}, \end{aligned} \tag{7}$$

the general outline of computing the intersection of $\mathcal{P}$ and $\mathcal{Q}$ is as follows, illustrated in Fig. 2.

(1) Fast decision of non-intersections using oriented bounding boxes (Sec. 4.1);
(2) Fast implicitization of the parametric surface with lower degree in each spline patch pairs (Sec. 4.2);
(3) Topology determination of the intersection curve in the parametric domain (Sec. 4.3);
(4) Computation of the intersection curve in 3D space (Sec. 4.4);
(5) Fast clipping of the intersection curve to adapt the parametric domain (Sec. 4.4).

Several key steps in the above outline, for example, implicitization, topology determination, and clipping are hard problems themselves and have been developed in their own research field over the decades. On one hand, these developments are seldom brought back to the surface intersection algorithm; on the other hand, more exploration is needed in these steps to give a detailed and practical algorithm of intersection computation.

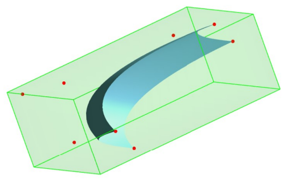
4.1 Fast Determination of Intersections

A fast and rough collision detection is performed between the two given spline surfaces using their bounding boxes constructed from control points. The separation of the two bounding boxes indicates the separation of the two surfaces. Otherwise, the surfaces are further split into Bézier patches and similar collision detection is taken on each pair of the Bézier patches from the two splines.

The *oriented bounding box* (OBB) [Assarsson and Moller 2000] of the control points of the B-Spline/Bézier surface is adopted, shown in Fig. 3(a). The OBB for a given point set is the smallest rectangular box that contains the point set with its axes not necessarily aligned with the global coordinate system. Chang et al. [2011] presented a fast algorithm for computing the OBB of the control points of B-Spline surfaces. Compared to the *axis-aligned bounding box* (AABB) in Fig. 3(b), OBB provides a tighter bound for the surface, hence is more efficient in discarding the non-intersection patches.

4.2 Fast Implicitization of Parametric Surfaces

Since we reduce the intersection of two B-spline surfaces into the intersections of patch pairs from the two splines, we next focus on computing the intersection of two Bezier patches. To do this, we plug the parametric form of one patch into the implicit form of the other. Next, we explore the implicitization technique of moving planes.



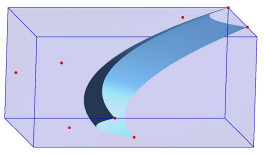


Fig. 3. The OBB (a) and AABB (b) of a spline surface. The control points are marked in red. The OBB is tighter than the AABB.

Given two parametric surfaces $\mathbf{P}(u,v)$ and $\mathbf{Q}(s,t)$ of bi-degree (m_1,n_1) and (m_2,n_2) with the following homogeneous parametrization:

$$\begin{aligned}\mathbf{P}(u,v) &= (a_1(u,v), b_1(u,v), c_1(u,v), d_1(u,v))\\ \mathbf{Q}(s,t) &= (a_2(s,t), b_2(s,t), c_2(s,t), d_2(s,t)),\end{aligned} \tag{8}$$

we convert one of the surfaces with the lower implicit degree[1], say $\mathbf{P}(u,v)$, into its implicit equation $F(x,y,z,w)=0$.

In order to implicitize $\mathbf{P}(u,v)$, we define three moving planes

$$\begin{aligned}f &:= d_1(u,v)x - a_1(u,v)w\\ g &:= d_1(u,v)y - b_1(u,v)w\\ h &:= d_1(u,v)z - c_1(u,v)w.\end{aligned} \tag{9}$$

Obviously, f,g,h all follow surface $\mathbf{P}(u,v)$. We compute the Dixon matrix of polynomials f,g,h with respect to variables (u,v):

$$D(x,y,z,w) := Dix_{u,v}(f,g,h). \tag{10}$$

Proposition 1. *When the parametrization $\mathbf{P}(u,v)$ has no base points,* $\mathrm{rank(D)} = 2\mathrm{m_1n_1}$*, and the determinant of D gives the implicit equation of the surface [Sederberg et al. 1984]. Otherwise, if the parametrization contains base points,* $\mathrm{rank(D)} < 2\mathrm{m_1n_1}$*, and the implicit equation is a factor of the maximal nonzero minors of D [Manocha and Canny 1992].*

Remark 1. *Since the rank r of polynomial matrix D is computationally expensive, a practical alternative strategy is to assign some generic values of $\boldsymbol{X}_i = (x_i,y_i,z_i,w_i), i=1,\cdots,k$ to matrix D and compute $r_i = \mathrm{rank(D(}\boldsymbol{X}_\mathrm{i}))$. According to the fact that set $A := \{\mathbf{X}_0 = (x_0,y_0,z_0,w_0)\in\mathbb{R}^4 | \mathrm{rank(D(X_0))} < \mathrm{r}\}$ is a zero-measure set in $\mathbb{R}^4$, most values of r_i equal the rank of polynomial matrix r. Once rank r is determined, the implicit equation of the parametric surface can be computed. Here a fast algorithm for computing the Dixon matrix presented in [Chionh et al. 2002] is adopted.*

For a parameter value (u,v), we define the vector

$$\mathbf{U}(u,v) := [1,u,v,\cdots,u^{2m_1-1}v^{n_1-1}].$$

Theorem 4.1. *A point $\mathbf{X}_0 = (x_0,y_0,z_0,w_0)$ with $w_0 \neq 0$ is a point on the surface $\mathbf{P}(u,v)$,* i.e., *$\mathbf{P}(u^*,v^*) = \lambda\mathbf{X}_0$ for some nonzero constant λ and parameter value (u^*,v^*) if and only if $f(u^*,v^*;\mathbf{X}) = 0, g(u^*,v^*;\mathbf{X}) = 0, h(u^*,v^*;\mathbf{X}) = 0$ have one common solution $\mathbf{X} = \mathbf{X}_0$, which is also equivalent to $D(\mathbf{X}_0)\mathbf{U}(u^*,v^*) = 0$.*

[1]The implicit degree of a rational surface of bi-degree (m_1,n_1) is $2m_1n_1 - k$, where k is an integer contributed by base points of the parametric surface.

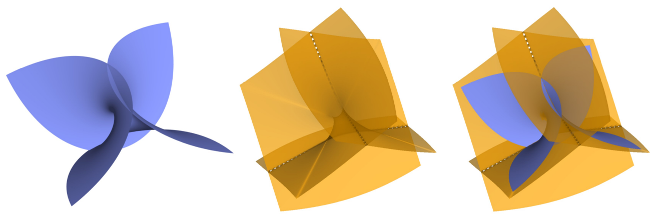


Fig. 4. Difference between a parametric surface with Eq. 11 within the domain $(u,v)\in[0,1]\times[0,1]$ and its corresponding implicit surface. (a) is the parametric surface; (b) is the implicit surface corresponding to (a); (c) is the difference between (a) and (b).

Proof. " $\Rightarrow$ ": If $\mathbf{P}(u^*,v^*) = \lambda\mathbf{X}_0$, we have $a_1(u^*,v^*) = \lambda x_0$, $b_1(u^*,v^*) = \lambda y_0$, $c_1(u^*,v^*) = \lambda z_0$, $d_1(u^*,v^*) = \lambda w_0$. Substituting X_0 into the equations, we obtain: $f(u^*,v^*;\mathbf{X_0}) = d_1(u^*,v^*)x_0 - a_1(u^*,v^*)w_0 = \lambda w_0\cdot x_0 - \lambda x_0\cdot w_0 = 0$. Similarly, $g(u^*,v^*;\mathbf{X_0}) = 0, h(u^*,v^*;\mathbf{X_0}) = 0$. So $\mathbf{X} = \mathbf{X}_0$ is a common solution of the three equations.

" $\Leftarrow$ ": If $f(u^*,v^*;\mathbf{X}) = 0, g(u^*,v^*;\mathbf{X}) = 0, h(u^*,v^*;\mathbf{X}) = 0$ have one common solution $\mathbf{X}_0 = (x_0,y_0,z_0,w_0)$, we have

$$\begin{aligned}d_1(u^*,v^*)x_0 &= a_1(u^*,v^*)w_0\\ d_1(u^*,v^*)y_0 &= b_1(u^*,v^*)w_0\\ d_1(u^*,v^*)z_0 &= c_1(u^*,v^*)w_0.\end{aligned}$$

Let $\lambda = d_1(u^*,v^*)/w_0$, we get $\mathbf{P}(u^*,v^*) = \lambda\mathbf{X}_0$. Here, we can guarantee that λ is a nonzero constant. If $\lambda = 0$, then $d_1(u^*,v^*) = 0$, we have $a_1(u^*,v^*) = b_1(u^*,v^*) = c_1(u^*,v^*) = 0$. Then $f(u^*,v^*;\mathbf{X}), g(u^*,v^*;\mathbf{X})$ and $h(u^*,v^*;\mathbf{X})$ are identical to 0, which is a contradiction to the uniqueness of solution $\mathbf{X}$. □

Remark 2. *The parametric surface here is given within a parametric domain $(u,v)\in[u_0,u_1]\times[v_0,v_1]$; however, implicitization itself has lost the restriction on this specific domain. See Fig. 4 for an illustration. As is proved in Thm. 4.1, Dixon matrices can help us to get the corresponding parameter value for any point $\mathbf{X}_0$ on the implicit surface by solving $D(\mathbf{X}_0)\mathbf{U} = 0$ for (u,v) as follows.*

- *If $D(\mathbf{X}_0)\mathbf{U} = 0$ has a solution $(u,v)\in[u_0,u_1]\times[v_0,v_1]$, then the point $\mathbf{X}_0$ is on the parametric surface within the parametric domain;*
- *If all real solutions of $D(\mathbf{X}_0)\mathbf{U} = 0$ are outside the region $[u_0,u_1]\times[v_0,v_1]$, then point $\mathbf{X}_0$ is outside the given parametric domain of the surface $\mathbf{P}(u,v)$.*

Example 1. *Consider a bi-cubic parametric surface $\mathbf{P}(u,v)\in\mathbb{RP}^3$ in $(u,v)\in[0,1]\times[0,1]$:*

$$\begin{aligned}\mathbf{P}(u,v) = [&-32u^3+96uv^2+48u^2-96uv-48v^2+6u+48v-11,\\ &96u^2v-32v^3-48u^2-96uv+48v^2+48u+6v-11,\\ &24u^2-24v^2-24u+24v,2].\end{aligned} \tag{11}$$

(1) Implicitization. The Dixon matrix $D(x,y,z,w)$ of $\mathbf{P}(u,v)$ is:

$$D(x,y,z,w) = \begin{pmatrix} 5184(4x-4y-7z) & 10368(11w+2y+4z) & \cdots & 0 & 0 \\ 1152(99w-32x+50y+92z) & -768(817w-32x+48y+196z) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}_{18\times 18}.$$

By Remark 1 we can quickly determine that $\operatorname{rank}(D(x,y,z,w)) = 9$. *Since $9 < 2\times 3\times 3 = 18$, according to Prop.1 the implicit equation of the surface is given by the 9×9 minor of the Dixon matrix $D(x,y,z,w)$.*

(2) Inversion formula.

For a given point $\mathbf{X} = (x_0, y_0, z_0, 1)$ satisfying $F(x_0,y_0,z_0,1) = 0$, we can determine whether this point is within the restricted parametric domain as following examples.

- $\mathbf{X}_1 := (-0.8660864938, -2.474813428, 1.846465226, 1)$. *By solving $D(\mathbf{X}_1)\mathbf{U} = 0$, we get the parametric value $(u,v) = (0.05514238183, 0.2901757920) \in [0,1]\times[0,1]$, which indicates that point $\mathbf{X}_1$ is within the parametric domain of surface $\mathbf{P}(u,v)$.*
- $\mathbf{X}_2 := (0.1882746188, -2.534024645, 2.683854645, 1)$. *By solving $D(\mathbf{X}_2)\mathbf{U} = 0$, we get the only parametric value $(u,v) = (1.003407729, 0.3274752700) \notin [0,1]\times[0,1]$, which indicates that point $\mathbf{X}_2$ is on the surface but outside the given parametric domain.*

An illustration of these two points with respect to the parametric surface is shown in Fig. 5.

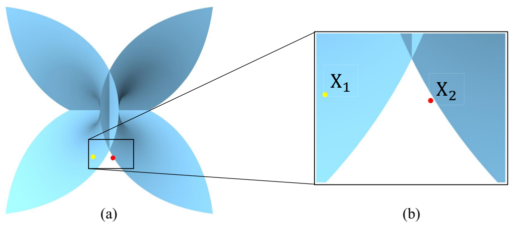


Fig. 5. Examples of different points on the same implicitized surface $F(x,y,z,w) = 0$. Figure (a) presents points X_1 and X_2 in Example 1; figure (b) illustrates the detail of the relative position of the points and the surface. The figures show that the yellow point is within the required domain of parametric surface $\mathbf{P}(u,v)$, while the red point lies outside the required domain of the parametric surface.

Remark 3. *When solving $D(\mathbf{X})\mathbf{U} = 0$, a practical way is to randomly select two linearly independent rows $\bar{D}$ of matrix $D(\mathbf{X})$ and solve for $\bar{D}(\mathbf{X})\mathbf{U} = 0$. This may result in redundant solutions, so we treat these solutions as follows. If all real solutions are outside region $[u_0,u_1]\times[v_0,v_1]$, the point $\mathbf{X}$ is not on the surface or not within the required parametric domain; if some solutions $(u_1,v_1),\cdots,(u_r,v_r)$ are within parametric domain $[u_0,u_1]\times[v_0,v_1]$, we compute $\mathbf{X}_i := \mathbf{P}(u_i,v_i), i = 1,\cdots,r$. If the distance from $\mathbf{X}$ to $\mathbf{X}_i$ is within a given threshold ϵ, we take these (u_i,v_i) as the parameter values for point $\mathbf{X}$.*

4.3 Topology Determination of the Intersection Curve in the Parametric Domain

Next we determine the topology of the intersection curve in the parametric domain of the surface $\mathbf{Q}(s,t)$. First, an implicit expression of the intersection curve in the parametric domain is computed; Then, the characteristic points of the intersection curve are computed; Finally, the connection of branches is determined, and the topology of the intersection curve is computed.

Let $F(X) = 0$ be the implicit equation of the parametric surface $\mathbf{P}(u,v)$. To compute the intersection of $\mathbf{P}(u,v)$ and $\mathbf{Q}(s,t)$, we substitute $X = \mathbf{Q}(s,t)$ into $F(X) = 0$:

$$\phi(s,t) := F(a_2(s,t), b_2(s,t), c_2(s,t), d_2(s,t)). \tag{12}$$

We next determine the topology of the curve $C := \{(s,t) \in \mathbb{R}^2 \mid \phi(s,t) = 0,\ (s,t) \in [s_0,s_1]\times[t_0,t_1]\}$.

We first compute two types of characteristic points: critical points and boundary points. A *critical point* (s,t) of C is where the tangent line of C is parallel to the coordinate axes, *i.e.*,

$$\begin{aligned} &\frac{\partial\phi}{\partial s}(s,t) = 0 \ (t\text{-critical point}) \ \ \textit{or} \\ &\frac{\partial\phi}{\partial t}(s,t) = 0 \ (s\text{-critical point}). \end{aligned} \tag{13}$$

A *singular point* (s,t) of C is a special critical point that simultaneously satisfies

$$\frac{\partial\phi}{\partial s}(s,t) = 0 \qquad \textit{and} \qquad \frac{\partial\phi}{\partial t}(s,t) = 0. \tag{14}$$

A *boundary point* of C is an intersection point of curve C with the boundary lines $l := \{(s,t) \in \mathbb{R}^2 \mid s = s_0 \text{ or } s = s_1 \text{ or } t = t_0 \text{ or } t = t_1\}$ within domain $(s,t) \in [s_0,s_1]\times[t_0,t_1]$.

Since boundary points are easy to compute, here we only address the computation of s-critical points. To do this, we first compute the resultant of ϕ and $\frac{\partial\phi}{\partial t}$ with respect to t:

$$R(s) := \mathrm{Res}_t(\phi, \frac{\partial\phi}{\partial t}), \tag{15}$$

and calculate the real roots $A := \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ of $R(s)$ in the region $s \in [s_0,s_1]$. Here we use exact computations to return small intervals $I_i = [a_i,b_i]$ that isolate α_i for each i. Then, for the t-coordinate corresponding to $s = \alpha_i$, we proceed in the following two different situations.

(1) if $a_i \neq b_i$, then $I_i = [a_i,b_i]$ where a_i and b_i are distinct rational numbers. We solve $\phi_{left}(t) := \phi(a_i,t) = 0$ and $\phi_{right}(t) := \phi(b_i,t) = 0$ in $t \in [t_0,t_1]$ respectively.
(2) if $a_i = b_i$, denoted as q_i, then $I_i = [q_i,q_i]$. Given a small perturbation ϵ, we solve $\phi_{left}(t) := \phi(q_i-\epsilon,t) = 0$ and $\phi_{right}(t) := \phi(q_i+\epsilon,t) = 0$ in $t \in [t_0,t_1]$, respectively. The magnitude of ϵ can be adjusted based on the required level of precision. In our experiment, we set $\epsilon = 1e^{-6}$.

For each $i = 1,\cdots,n$, we record the roots of $\phi_{left}(t) = 0$ and $\phi_{right}(t) = 0$ as:

$$\begin{aligned} L_i &:= \{\underline{\beta} \in [t_0,t_1] \mid \phi_{left}(\underline{\beta}) = 0\} = \{\underline{\beta}_{i,1}, \underline{\beta}_{i,2}, \cdots, \underline{\beta}_{i,l_i}\} \\ R_i &:= \{\overline{\beta} \in [t_0,t_1] \mid \phi_{right}(\overline{\beta}) = 0\} = \{\overline{\beta}_{i,1}, \overline{\beta}_{i,2}, \cdots, \overline{\beta}_{i,r_i}\}, \end{aligned} \tag{16}$$

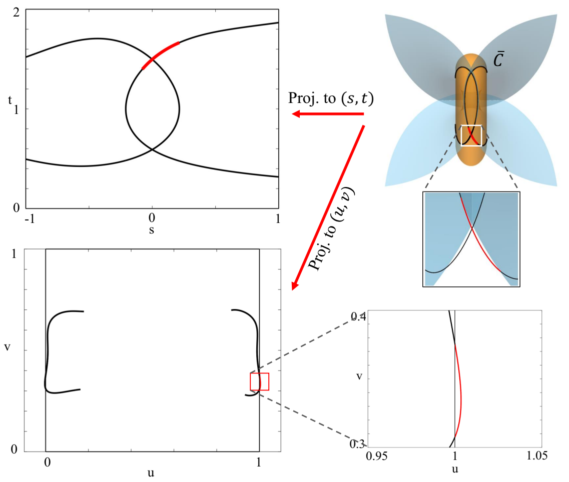


Fig. 6. The intersection curve $\overline{C}$ in 3D modeling space and its projection onto the parametric domain (s,t) and (u,v). The red part of $\overline{C}$ has shown its corresponding parts in the (s,t) domain and the (u,v) domain. In (s,t) domain, the red part is within the required region $(s,t) \in [-1,1] \times [0,2]$; however, in (u,v) domain, the red part is out of the required region $(u,v) \in [0,1] \times [0,1]$.

and let

$$L := \bigcup_{i=1}^{n} L_i \qquad R := \bigcup_{i=1}^{n} R_i. \tag{17}$$

Now two sets of points on the curve C are obtained:

$$\begin{aligned} P_l &:= \{(a_i, \underline{\beta}_{i,j}) \in A \times L \mid i = 1,2,\cdots,n,\ j = 1,2,\cdots,l_i\}, \\ P_r &:= \{(b_i, \overline{\beta}_{i,j}) \in A \times R \mid i = 1,2,\cdots,n,\ j = 1,2,\cdots,r_i\}, \end{aligned} \tag{18}$$

which include all s-critical points and those points on C that have the same s-coordinates as s-critical points. In the following context, we call P_l and P_r *left and right helping points*, respectively. According to [Jin and Cheng 2021], once these left and right helping points are computed, the topology of the intersection curve in the parametric domain (s,t) is determined. Refer to Fig. 14 and the appendix for the detailed example.

4.4 Tracing and Clipping the Intersection Curve

The advantage of the previous step in topology determination lies in that it greatly simplifies the locus tracing of the intersection curve $\phi(s,t) = 0$ in the parametric domain (s,t), especially in the neighborhood of singular points, where the standard tracing scheme [Barnhill et al. 1987] tends to lose its direction. Starting from our topology graph, tracing from a singular point, such as in Fig. 14(a) in the appendix, is decomposed by tracing from some left helping points or right helping points in a ϵ-neighborhood of this singular point along its related curve branches. For the choice of the tracing direction and step length, one can refer to [Chen et al. 1997] and the appendix.

Remark 4. *Since the tracing is along $C : \phi(s,t) = 0$, when C is mapped to the 3D modeling space, the 3D intersection locus $\bar{C}$ naturally falls within the required parametric region $(s,t) \in [s_0, s_1] \times [t_0, t_1]$.*

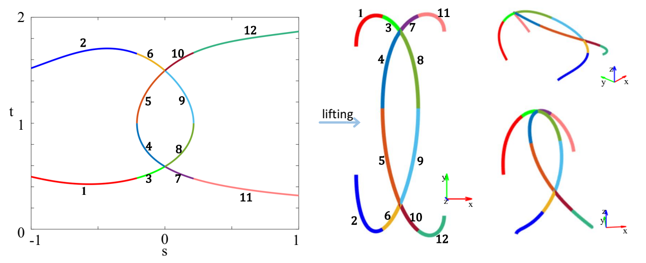


Fig. 7. Intersection curve lifting from 2D parametric domain (s,t) to 3D space. Left is the intersection curve C with different local branches. Right is the lifted intersection curve $\overline{C}$ with different viewpoints. Each local branch in $\overline{C}$ is lifted from the local branch in C with the corresponding number.

Unfortunately, some part of $\bar{C}$ may fall out of the required region $(u,v) \in [u_0,u_1] \times [v_0,v_1]$, since computing $\phi(s,t) = 0$ inherits the implicitization of the other surface $\mathbf{P}(u,v)$, and implicitization itself has lost the restriction of the required parametric domain in (u,v). See Fig. 6 for an illustration.

In the following, we clip the obtained 3D intersection curve $\overline{C}$ (Fig. 7) such that for every point $\mathbf{X}_0$ on $\overline{C}$, its corresponding parameter values on $\mathbf{P}(u,v)$ and $\mathbf{Q}(s,t)$ are within $[u_0,u_1] \times [v_0,v_1]$ and $[s_0,s_1] \times [t_0,t_1]$. The strategy consists of two parts: 1. Split the so far obtained $\overline{C}$ in Fig. 8(b) at its intersection points with the boundary curve of the surface $\mathbf{P}(u,v)$, as is presented in Fig. 8(c) and Fig. 8(d); 2. Clip $\overline{C}$ and select the part within $(u,v) \in [u_0,u_1] \times [v_0,v_1]$, as is presented in Fig. 8(e). The details are as follows.

(1) *Split $\bar{C}$*: We sample (u,v) on the four boundary lines of surface $\mathcal{P}$ in $[u_0,u_1] \times [v_0,v_1]$ with a step size of $1/N$, as is shown in Fig. 9(a), and then obtain a series of approximate intersection points $\mathbf{p}_i, i = 1,\cdots,l$ of $\bar{C}$ with these four boundary lines. These points are called splitting points, which are marked with red asterisks in Fig. 9(b). Split $\bar{C}$ at $\mathbf{p}_i, i = 1,\cdots,l$ into branches $C_1, C_2, \cdots, C_\gamma$. See Fig. 8(d).
(2) *Clip $\bar{C}$*: For each split branch $C_i, i = 1,2,\cdots,\gamma$, a random representative point $\mathbf{q}_i$ on C_i is selected as shown in Fig. 9(c). We decide whether to discard or keep C_i based on the behavior of $\mathbf{q}_i$: Compute the parameters (u,v) for $\mathbf{q}_i$ by solving $D(\mathbf{q}_i)\mathbf{U} = 0$ for (u,v). According to Remark 2, if there is one solution in $(u,v) \in [u_0,u_1] \times [v_0,v_1]$, the branch C_i is retained; otherwise, it is discarded. Then $\bar{C}$ is clipped within the (u,v) domain, see Fig. 9(d).

It is worth mentioning that after we split the original branches at the surface boundaries, a representative point of each split branch lies on the surface if and only if all points on the split branch lie on the surface. Hence, we focus only on those representative points.

5 EVALUATIONS AND COMPARISON

We implemented our algorithm and performed comparisons with the discrete intersection algorithm for mesh Booleans [Cherchi et al. 2022], the affine arithmetic-based intersection method with GPU acceleration [Lin et al. 2013], the intersection packages provided by two representative open-source software: OCCT [2023] and SISL

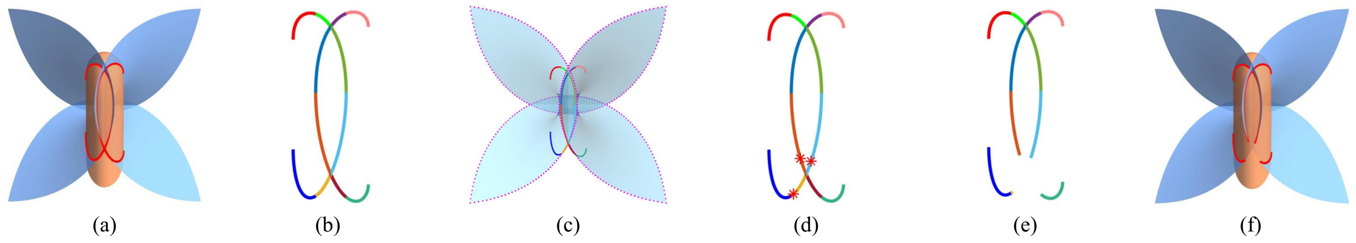


Fig. 8. Clip a 3D intersection curve to adapt the required parametric domain. (a) is the 3D intersection curve $\bar{C}$ before clipping; (b) presents different intersection branches in the parametric domain (s, t) by distinct colors; (c) illustrates the sample points of boundary curves of the surface $\mathcal{P}$; (d) shows the splitting points in red asterisks; (e) is the selected branches after clipping; (f) depicts the final intersection curve of two surfaces in the required parametric domain.

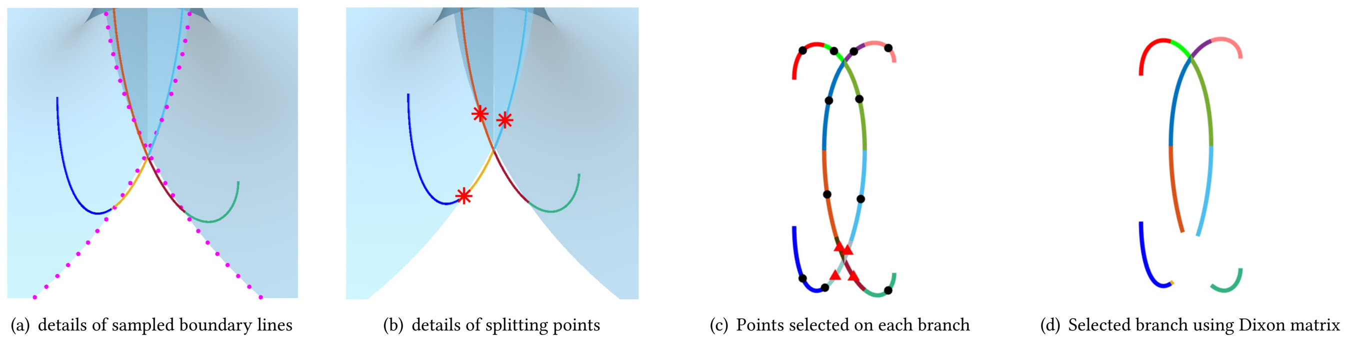


Fig. 9. Curve clipping detail. (a) marks the sample points on boundary lines in magenta; (b) shows the splitting points with red asterisks, which are composed of the points on the intersection branches that are close enough to the sample points in (a); (c) records the selected points (black point and red triangle) on each split branch, which are used to determine whether the branch needs to be abandoned. The black ones are the branches to be remained while the red ones are to be abandoned. (d) represents the final selected branches.

Table 1. The compared methods and their parameter settings in experiments.

Methods	Parameters
AA-GPU	maximum grid density=500
Mesh	grid density=50
OCCT	parameter tolerance=1e-7
SISL	geometric tolerance=1e-6
ACIS	fitting tolerance=1e-6
Ours	geometric tolerance=1e-6

[2021], and also with the commercial 3D modeler, ACIS [2023]. The parametric settings of these competitors are listed in Table 1. All experiments are executed on a PC with 3.20 GHz AMD Ryzen 7 5800H and 16 GB RAM.

5.1 Topology correctness

We first focus on the topology correctness of the intersection of two general B-spline surfaces, and compare our results with the open-source geometry kernel OCCT and the commercial engine ACIS. Plenty of experiments show that when the two surfaces intersect transversely, all three algorithms give the correct results. However, when the two surfaces are in certain tangential or special intersection topology, our algorithm performs most robustly in these challenging situations, while OCCT and even ACIS sometimes fail. Fig. 1 shows representative examples with different intersection topologies.

In Fig. 1(a), the intersection curve contains two loops that are crossing at two common points. Our algorithm, OCCT and ACIS all give the correct intersection result; in (b), the two surfaces intersect at two separate curve branches and are in contact at an isolated point. OCCT loses this isolated contact point and gives only the two separate intersection curves, while our algorithm and ACIS give the correct result. The omission of tangent points in the OCCT intersection algorithm stems from its reliance on a discrete method for the initial point search. By contrast, our proposed algorithm utilizes polynomial root solving to guarantee the inclusion of all tangent points; in (c), the two surfaces are very close to each other in the triangular domain, and the intersection curve is a curved triangle, two edges of which are the boundary curves of the two surfaces and the left edge is a regular intersection curve. OCCT loses the two common boundary curves, while both our algorithm and ACIS give the complete result; in (d), the two surfaces are again very close to each other, and their intersection is even more complex than (c), that is, the intersection contains two quadrilaterals sharing one edge. The two surfaces are in contact along the common edge of the two quadrilaterals, and intersect transversely at the other six curved edges of the two quadrilaterals, which are also the boundaries of the surfaces. OCCT returns three of the six edges of the two quadrilaterals, but has lost the contact curve in the middle and the other three boundary curves, while both our algorithm and ACIS give the complete result. Examples (c) and (d) illustrate the limitations of the OCCT intersection algorithm when computing intersection curves at surface boundaries. The omission of intersection curves along the boundaries might result from improper termination criteria during the tracing process. Our proposed algorithm addresses this issue by incorporating specific handling for boundary intersection curves, thereby preventing the occurrence of this problem; in (e), the intersection curve contains two loops that are contact at one common point. OCCT fails to return any intersection point, while both our algorithm and ACIS give the correct result; in (f), the two surfaces are in high-order contact along a line. Neither OCCT nor ACIS returns any intersection point, while ours correctly returns the contact line. The failures of OCCT in examples (e) and (f), as well as ACIS in example (f), demonstrate that when two surfaces are tangent or in high-order contact, their normals at the tangent curves are the same. Existing algorithms exhibit deficiencies in calculating the marching direction during the tracing process. In contrast, our proposed algorithm performs curve tracing on planar polynomial curves in the parametric domain, thereby enabling the easy determination of the tracing direction; in (g), the two surfaces are in contact at three isolated points. Neither OCCT nor ACIS provides a complete set of contact points, while our algorithm ensures the detection of all three points of contact. The reason for the missing intersection points in the OCCT intersection algorithm is analogous to example (b), both are attributed to the discrete algorithm of the initial point detection. As for ACIS, the cause of losing one of the intersection points may be linked to the fact that this particular point also serves as a self-intersection point for the surfaces. The ACIS algorithm demonstrates shortcomings in handling such intricate intersection points.

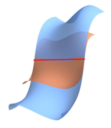
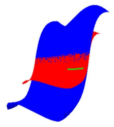
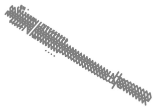
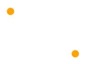
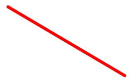
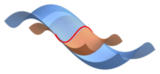
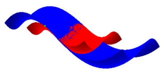
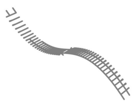
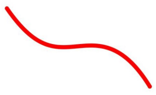
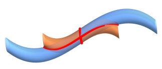
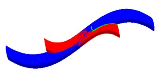
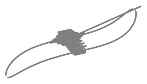
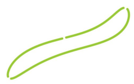
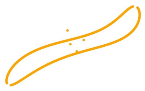
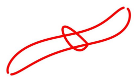
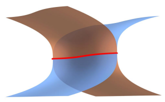
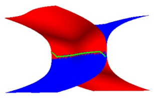
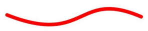
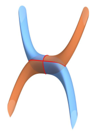
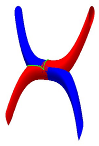
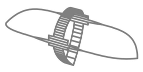
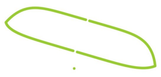
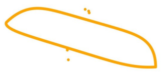
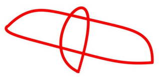
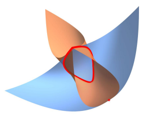
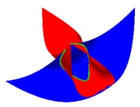
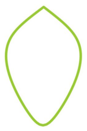
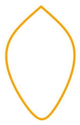
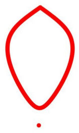
Overall speaking, our algorithm performs most robustly in all these challenging situations in topology. ACIS outperforms OCCT, and gives the correct results in most situations, but fails at examples with high-order contact curves or multiple isolated contact points. The topological robustness of our algorithm lies in our algebraic treatment of the implicitization and the topology analysis of the intersection curve. On one hand, although traditional implicitization methods such as resultants and Gröbner basis computation are not very practical in both efficiency and numerical computations, the technique of moving planes, which we have adopted, has overcome these difficulties and paved the way for the application of implicitization to many geometry problems. On the other hand, the topological analysis and clipping method ensure topological correctness, especially for those tangential intersections or isolated contact points, which are really difficult to capture through the discrete techniques adopted by most software.

5.2 Comprehensive performance comparison

We have performed plenty of experiments on bi-cubic or bi-quadratic B-Spline surfaces, and parts of them are shown in Table 2. Since for practical use in CAD systems, efficiency is one of the most important factors in evaluating the intersection algorithm, we compare our algorithm with efficient and practical state-of-the-art intersection methods, including the affine arithmetic-based intersection method with GPU acceleration (AA-GPU), the mesh intersection approach (Mesh Booleans), the open source intersection packages in OCCT and SISL, and the commercial 3D ACIS modeler (ACIS), from the perspective of the computation accuracy, topology correctness and efficiency. We claim that our algorithm has the most robust topology correctness and comparable practical efficiency with the existing methods or packages.

In Table 2, six examples of B-Spline surface intersections are provided, in which our method always gives the correct topology, while the other five algorithms either fail or give the undesired intersection topology. The results are illustrated for every method, where 'NULL' means that the computation result returns as no intersections. The intersection results show that tangential intersections or isolated singular points are very challenging for all competitors. Our algorithm is particularly designed based on topology determination and inversion formulas, so tangential intersections can always be treated well. The Mesh Booleans method can obtain relatively smooth intersection curves when dealing with the transversal intersection of surfaces. However, this method tends to give a 2D grid region instead of a 1D curve along the tangential intersection curve or the isolated point. A 2D grid-like intersection is generated because the triangular meshes corresponding to the two surfaces cannot guarantee to pass the tangential curve or isolated point simultaneously, resulting in significant errors in the intersection lines of the meshes near these tangential curves or isolated points. AA-GPU fails to get correct results in examples (1∼5) when the intersection is tangential or near boundaries, and gives partial results in example (6) when the intersection has an isolated point. Since the two surfaces are very close to each other near the tangential curves, the overestimation of the bounding boxes with the affine arithmetic may contribute to the very thick strips of intersection grids on parametric regions. Thus, the undesired results may be attributed to the phenomenon in which the required accuracy near the boundary and tangential curves exceed the maximum subdivision accuracy during the subdivision of the parametric regions of B-Spline surfaces. OCCT fails to give the intersection result in examples (1)(2)(4), and gives partial intersection results but loses a whole branch in examples (3)(5)(6); SISL fails on all the six examples; even ACIS only returns several intersection points in examples (1)(2), returns no result in example (4), loses a whole branch in examples (3)(5), and loses the isolated singular point in example (6). The loss of the tangential curve in OCCT, SISL, and ACIS may be caused by the fact that the normal vectors of the two surfaces are parallel at every point on the tangential curve, which results in the inability of the 3D tracing step to track the entire intersection curve. Meanwhile, the omission of the isolated point may be caused by the use of some discrete methods in the search for the starting point, which results in the isolated point being ignored during the discretization process.

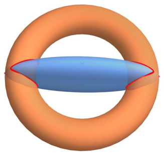
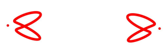
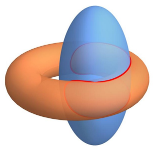
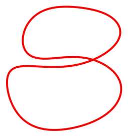
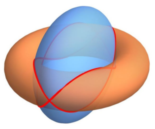
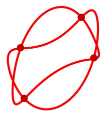
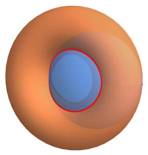
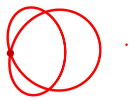
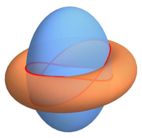
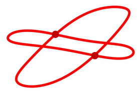
Table 2. Intersections of two B-Spline Surfaces. The first column shows the surfaces and their bi-degrees. The remaining columns represent the intersection results and the consuming times (in seconds) of different algorithms. AA-GPU refers to the affine arithmetic-based B-Spline surface intersection method with GPU acceleration. Mesh Booleans refers to the interactive and robust mesh boolean algorithm. OCCT refers to the intersection algorithm in the Open CASCADE Technology. SISL refers to the intersection algorithm in the SINTEF Spline Library. ACIS refers to the intersection algorithm in 3D ACIS Modeler. In the first column, the blue and orange surfaces depict the two surfaces to be intersected, and the red curve represents the intersection result of our method. The second column displays two surfaces in blue and red, with the intersection result of the AA method represented in green. N/A indicates that the program reported an error in this example, and the 'NULL' in the image indicates that the result obtained by this method is no intersection.

Surfaces	AA-GPU	Mesh Booleans	OCCT	SISL	ACIS	Ours
 example 1 (3,3)	 1.451	 0.011	NULL 0.003	NULL N/A	 0.003	 a tangent line 0.036
 example 2 (3,3)	 1.775	 0.038	NULL 0.003	NULL N/A	0.005	 a tangential curve 0.082
 example 3 (3,3)	 1.836	 0.008	 0.076	NULL 3.612	 0.074	 a loop and two curves 0.054
 example 4 (3,3)	 0.388	0.013	NULL 0.002	NULL N/A	NULL 0.003	 a tangential curve 0.019
 example 5 (3,3)	 0.532	 0.021	 0.040	NULL N/A	 0.054	 two loops 0.103
 example 6 (2,2)	 0.254	0.012	 0.023	NULL 0.288	 0.008	 a loop and a point 0.514

5.3 Intersection of Surfaces with Special Geometry

According to our method, given two parametric surfaces or spline surfaces, we transform one of them into its implicit expression. However, for specific surface types, such as quadrics, tori, cyclides, and surfaces of revolution, the implicit equations can be directly derived from their shape parameters, eliminating the need for the implicitization step. Consequently, our algorithm simplifies the intersection of these surfaces by directly deriving the implicit equation

Table 3. Intersections of torus and ellipsoid. The first column provides the parameters of the torus and the ellipsoid for each example; the second column displays two surfaces; the third column represents the intersection results; the remaining columns illustrate the time cost (in seconds) for both ACIS and our method. Given the absence of the ellipsoid class in ACIS, we represent the ellipsoids as piece-wise spline surfaces and intersect them with the torus. **Bold** fonts indicate better performance.

Paras 4	Surfaces	Intersection	Time cost	
			ACIS	Ours
example 1			**0.031**	0.039
example 2			**0.026**	0.058
example 3			0.168	**0.039**
example 4			0.078	**0.042**
example 5			0.154	**0.053**

from the shape parameters. This streamlined approach replaces the conventional implicitization step in our implementation.

Table 3 shows the comparison of our method with ACIS in the intersection of an ellipsoid and a torus, where the implicit equation of the ellipsoid is directly written down (note that here we prefer to select the implicit equation of the surface with lower implicit degree). A comparison is made in the geometry of the intersection curve and the time cost. The shape parameters of the torus and ellipsoid of the corresponding examples are listed in Table 4.

It is worth mentioning that when dealing with surface intersections, ACIS classifies all surface types into planes, spheres, cones, tori, and splines. Ellipsoids are converted to spline surfaces in ACIS before the intersection computation.

Our extensive testing of intersections between ellipsoids and tori indicates that our method always achieves the same correct intersection geometry as ACIS. Table 3 shows only several representative examples. In terms of efficiency, if the intersection curve contains cross singular points, which are marked with red dots, our method is usually faster than ACIS; if the intersection curve contains only simple loops, ACIS can be slightly faster than our method due to its highly-optimized implementation.

Table 4. The parameters of torus and ellipsoid. In the Torus column, 'Center' represents the center of the torus, 'Normal' the normal of torus, 'R' the major radius and 'r' the minor radius. In the Ellipsoid column, 'Center' represents the center of the ellipsoid, 'x-axis' and 'y-axis' define a Cartesian coordinates frame, 'a', 'b' and 'c' are the length of the semi-axes.

	Torus				Ellipsoid					
	Center	Normal	R	r	Center	x-axis	y-axis	a	b	c
example 1	[0,0,0]	[0,0,1]	4	1	[0,0,0]	[1,0,0]	[0,1,0]	1	5	2
example 2	[1,0,0]	[0,0,1]	3	1	[0,0,0]	[0,1,0]	[0,0,1]	3	4	2
example 3	[0,0,0]	[0,0,1]	3	1	[0,0,0]	[0,1,0]	[0,0,1]	4	3	2
example 4	[0,0,0]	[0,0,1]	2	1	[1,0,0]	[1,0,0]	[0,1,0]	2	2	1
example 5	[0,0,0]	[0,0,1]	3	1	[0,0,0]	[0,1,0]	[0,0,1]	3	4	2

5.4 Accelerating Tracing with TBB

During the process of tracing, the computation of each local branch (Fig. 7) is independent and uniquely determined by the starting point, the ending point, and the implicit equation, which have been calculated in the implicitization and topology determination steps. Consequently, in order to accelerate the tracing process, we employed the threading building blocks (TBB) library function, *parallel_for*, in the tracing step of local branches.

Table 5 illustrates the comparison of the average time cost of the above-mentioned examples before and after TBB acceleration. Based on the obtained results, it can be inferred that when the intersection curve has only a single branch within the parametric domain (*e.g.* examples (1)(2)(4) in B-Spline surfaces), the employment of TBB doesn't improve the efficiency of our algorithm. Nevertheless, in cases where the intersection curve has multiple local branches within the parametric domain, the utilization of TBB enables simultaneous tracing of these local branches. This capability leads to a substantial reduction in algorithmic computation time, up to 2-3 times faster than the original calculation time. See Fig. 10 for an illustration of the local branches in the parametric domain.

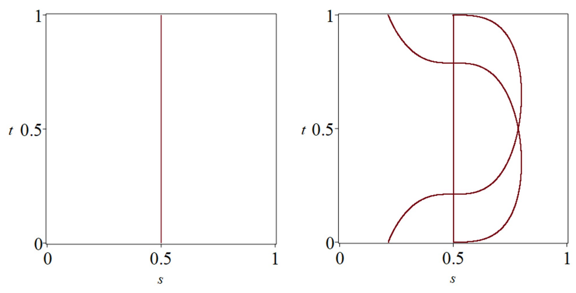


Fig. 10. Two examples of the intersection curve of B-Spline surfaces in the parametric domain (s,t). The figure on the left represents example (1), while the figure on the right represents example (5). Example (1) has only one local branch while example (5) has several local branches.

Table 5. Time cost comparison (in seconds) before and after employing TBB parallelism for the above-mentioned examples. The first six examples are B-spline surface intersection cases, while the last five examples are special surface intersection cases.

	B-Spline surface examples						Special surface examples				
	example 1	example 2	example 3	example 4	example 5	example 6	example 1	example 2	example 3	example 4	example 5
before	0.036	0.082	0.140	0.020	0.280	1.241	0.083	0.131	0.094	0.073	0.087
after	0.036	0.082	0.054	0.019	0.103	0.514	0.039	0.058	0.039	0.042	0.053

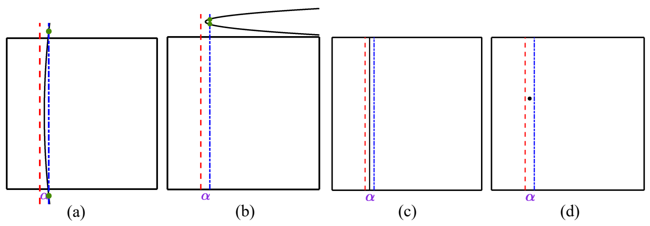


Fig. 11. Four cases when there are no left helping points and right helping points corresponding to root α. (a) represents that all roots are outside the parametric domain, but the intersection curve crosses through the domain; (b) indicates that all roots are outside the parametric domain, and the intersection curve does not pass through it; (c) represents that the intersection line is perpendicular to the s-axis; (d) shows that the intersection is an isolated point. The vertical dashed lines show the small isolation intervals for α, and the green points are the intersection points between the vertical lines and the curve. The black box represents the rectangular parametric domain $[s,t] \in [s_0,s_1] \times [t_0,t_1]$.

5.5 Special Cases in Topology Determination

In the topology determination step of our algorithm, it is possible to encounter situations where the left polynomial $\phi_{left}(t)$ and the right polynomial $\phi_{right}(t)$ corresponding to α have no roots in the parametric domain $t \in [t_0,t_1]$, which indicates there are no left helping points and right helping points corresponding to the root α. As is shown in Fig. 11, we categorize this situation into four distinct cases and discuss each of them individually.

(a) If the roots of $\phi_{left}(t)$ and $\phi_{right}(t)$ are outside the parametric domain $t \in [t_0,t_1]$ but the curve traverses the region $[s_0,s_1] \times [t_0,t_1]$, we can get the curve through boundary point tracing.
(b) If the roots of $\phi_{left}(t)$ and $\phi_{right}(t)$ are outside the parametric domain $t \in [t_0,t_1]$ and the curve does not fall within the parametric domain, we just ignore this curve.
(c) If the curve is a vertical line $s = \alpha$, we can detect it by substituting $s = \alpha$ into the polynomial $\phi(s,t)$ in Eq. 12 and determining if $\phi(\alpha,t)$ is a zero polynomial.
(d) If the intersection is an isolated point, we can find the point by substituting $s = \alpha$ into the polynomial $\phi(s,t)$ and solve $\phi(\alpha,t) = 0$ in the parametric domain $t \in [t_0,t_1]$.

To sum up, when $\phi_{left}(t)$ and $\phi_{right}(t)$ have no roots in the parametric domain $t \in [t_0,t_1]$, we substitute $s = \alpha$ into the polynomial $\phi(s,t)$ to get polynomial $\phi(\alpha,t)$. If $\phi(\alpha,t)$ is a zero polynomial, then we uniformly sample $s = \alpha$ in $t \in [t_0,t_1]$ to get the vertical intersection line in the parametric domain. If $\phi(\alpha,t)$ is a nonzero polynomial, we solve $\phi(\alpha,t) = 0$ in $t \in [t_0,t_1]$ to get the possible isolated intersection point.

Our algorithm also provides dedicated treatment for the intersection topology along the boundaries of the parametric domain. We first detect whether the boundary lines of the domain $(s,t) \in [s_0,s_1] \times [t_0,t_1]$ are intersection curves of the two surfaces as follows. We substitute $s = s_0$ into the polynomial $\phi(s,t)$ to get $\phi_{s_0}(t)$. If the $\phi_{s_0}(t)$ is a zero polynomial, it indicates that the boundary line $l_{s_0} := \{(s,t) \in \mathbb{R}^2 \mid s = s_0, t \in [t_0,t_1]\}$ corresponds to an intersection of the two surfaces. Similarly for the other boundary lines $s = s_1$, $t = t_0$, and $t = t_1$. Then we uniformly sample the boundary intersection lines and map them to the 3D space to get the corresponding boundary intersection curves in $\mathbb{R}^3$.

5.6 Applications

The intersection algorithm is crucial in the Boolean operations of solids. By computing the intersection curves of B-Spline patches in B-Rep solids, we can get different Boolean operation results such as union, intersection and subtraction. Fig. 12 shows the Boolean operation results of two hummingbirds using our intersection algorithm. We apply our algorithm to the Boolean operations of complex CAD models, and perform comparisons with OCCT and ACIS. When the adjacent surfaces in the CAD models intersect transversely, our

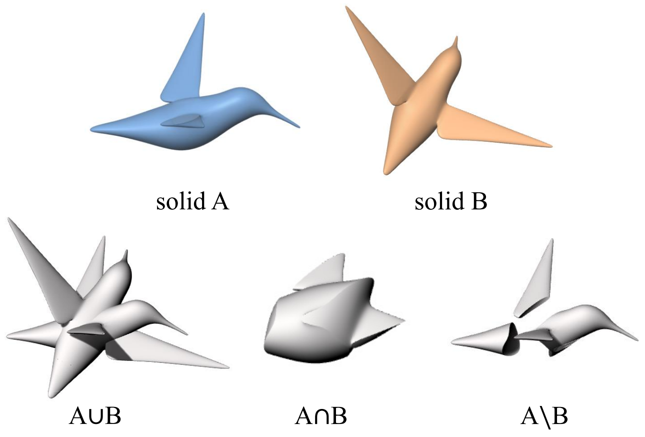


Fig. 12. Boolean operation results of two hummingbird models. The hummingbird model consists of 312 bi-cubic B-Spline patches, with each B-Spline patch containing 6×6 control points. Our method has detected 99 pairs of intersecting patches. Union, intersection and subtraction are performed after computing the intersection curves of two surfaces.

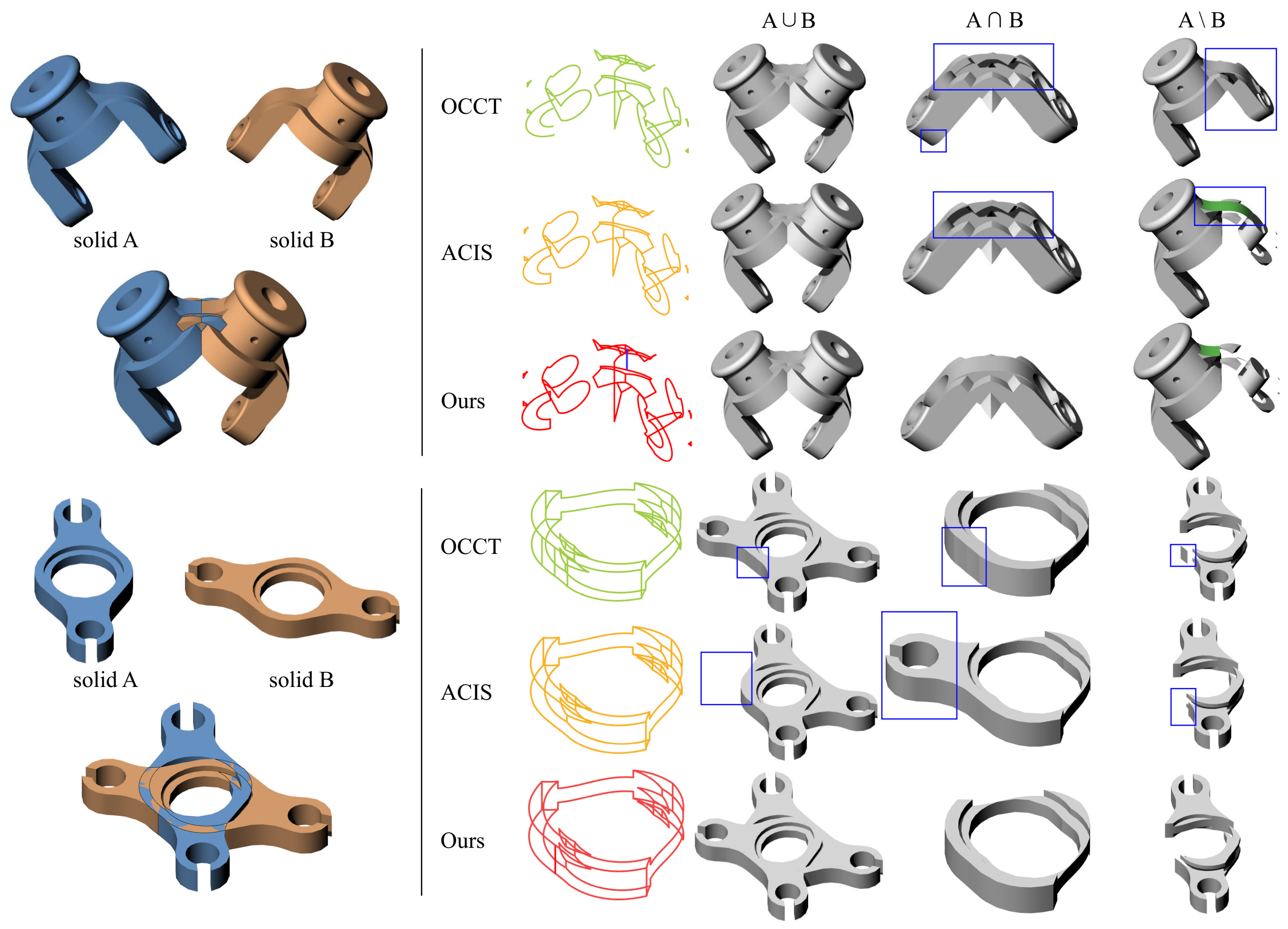


Fig. 13. Two examples of Boolean operations. We compared the results of Boolean operations based on the intersection curves generated by OCCT, ACIS, and our method. The intersection curves of OCCT and ACIS are represented in green and yellow respectively. Intersection curves of our method are represented in red and the tangential intersection in example 1 is highlighted in blue. Incorrect parts in Boolean operations are highlighted with the blue boxes.

algorithm gives the same Boolean operation results as OCCT and ACIS; nevertheless, when the adjacent surfaces are in high-order contact or intersect along their parametric boundaries, OCCT and ACIS sometimes provide incorrect results. Fig. 13 shows two such examples.

In the first example, we test the Boolean operations on two Cardan joint models. The intersection curves of OCCT are represented in green, where part of the intersection branch in the lower-left region and a tangential line in the middle region are missing. The miscalculation of the intersection curves leads to an error in the Boolean intersection performed by OCCT. The incorrect parts are highlighted within a blue box. Meanwhile, the incorrect Boolean subtraction results in OCCT arise from the inaccurate intersection calculation and the erroneous discarding of surface portions of solid A. The intersection curves of ACIS are represented in yellow. ACIS returns the transversal intersecting curves, but misses the tangential intersection curve, which is highlighted in blue in our corresponding intersection result. The missing tangential curve leads to the wrong omission of surfaces in the Boolean intersection results and the failure to subtract the portions of solid A that lie within solid B in Boolean subtraction results. The missing surface in the Boolean intersection is shown in the blue box, while the surface that is not properly trimmed in the Boolean subtraction is represented in green.

In the second example, we conduct Boolean operations on an additional pair of mechanical component models. OCCT mistakenly calculates two extra intersecting lines near the tangential intersection, which leads to inaccuracies of surfaces near the tangent region in both Boolean union and Boolean intersection results. The surplus intersections also lead to an erroneous trimming of a surface in solid A in the Boolean subtraction outcome. The intersection result of ACIS fails to capture a tangent intersection, subsequently impacting the surface classification process within the Boolean union and Boolean intersection outcomes, leading to a partial omission in the

Boolean union and an excessive portion in the Boolean intersection. The Boolean subtraction result of ACIS also leads to errors due to the same reason. Contrary to OCCT and ACIS, our algorithm accurately calculates the intersection curves of the tangential and boundary regions of surfaces, resulting in the correct Boolean union, intersection, and subtraction outcomes.

6 CONCLUSIONS AND FUTURE WORK

We present a topology guaranteed algorithm for computing the intersection of two B-Spline surfaces. The algorithm begins by performing fast collision detection using the oriented bounding boxes to eliminate non-intersection patch pairs. Subsequently, the implicitization for one Bézier surface within each Bézier patch pair is computed efficiently using the Dixon matrix of moving planes. Following this, a topology determination strategy is applied to the intersection curve in the parametric domain and the helping points on the curve are calculated for topology guaranteed tracing. Finally, a clipping method based on the inversion formula of the Dixon matrix is demonstrated to restrict the intersection curve within the parametric domain of two surfaces. We have illustrated the topological correctness of our method with various examples of B-Spline surface intersection with different topologies. Furthermore, a detailed comparison of the efficiency and topology correctness of our algorithm with the affine arithmetic-based method, mesh-to-mesh intersection algorithm and intersection commands in OCCT, SISL and ACIS is presented, which indicates that our method can maintain topology correctness even when two surfaces are under special relative position.

In the proposed method, we derive the implicit equation of the parametric surface from its Dixon matrix and then use it to determine the critical points of the intersection curves. However, the Dixon matrix contains all the necessary surface information required for performing intersection computations. Thus, our future research will focus on simplifying this process by directly computing the critical points of the intersection using the Dixon matrix, thereby enhancing the efficiency of our algorithm. Moreover, in our current work, we convert two B-Spline surfaces into multiple pairs of parametric surfaces to perform surface intersection within these pairs. However, this conversion step introduces an additional computational overhead. To address this limitation, future studies should explore ways to exploit the geometric properties of B-Spline surfaces. By making full use of these properties, we can further accelerate our algorithm and improve its performance.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2021YFB1715900), the National Natural Science Foundation of China for Outstanding Young Scholars (12022117), the National Natural Science Foundation of China (62172415), and the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-034).

REFERENCES

Ulf Assarsson and Tomas Moller. 2000. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools* 5, 1 (2000), 9–22.

C.L. Bajaj, C.M. Hoffmann, R.E. Lynch, and J.E.H. Hopcroft. 1988a. Tracing surface intersections. *Computer Aided Geometric Design* 5, 4 (1988), 285–307.

Chandrajit L Bajaj, Christoph M Hoffmann, Robert E Lynch, and JEH Hopcroft. 1988b. Tracing surface intersections. *Computer Aided Geometric Design* 5, 4 (1988), 285–307.

Robert E Barnhill, Gerald Farin, M Jordan, and Bruce R Piper. 1987. Surface/surface intersection. *Computer Aided Geometric Design* 4, 1-2 (1987), 3–16.

Robert E Barnhill and Scott N Kersey. 1990. A marching method for parametric surface/surface intersection. *Computer Aided Geometric Design* 7, 1-4 (1990), 257–280.

Laurent Busé, David Cox, and Carlos d'Andrea. 2003. Implicitization of surfaces in $\mathbb{P}^3$ in the presence of base points. *Journal of Algebra and its Applications* 2, 02 (2003), 189–214.

Chia-Tche Chang, Bastien Gorissen, and Samuel Melchior. 2011. Fast oriented bounding box optimization on the rotation group SO (3, $\mathbb{R}$). *ACM Transactions on Graphics (TOG)* 30, 5 (2011), 1–16.

Falai Chen, Yuyu Feng, and Kozak Jerenj. 1997. Tracing a planar algebraic curve. *Applied Mathematics-A Journal of Chinese Universities* 12, 1 (1997), 15–24.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. 2022. Interactive and Robust Mesh Booleans. *ACM Transactions on Graphics (TOG)* 41, 6 (2022).

Eng-Wee Chionh, Ming Zhang, and Ronald N Goldman. 2002. Fast computation of the Bezout and Dixon resultant matrices. *Journal of Symbolic Computation* 33, 1 (2002), 13–29.

Carl de Boor. 1971. *SUBROUTINE PACKAGE FOR CALCULATING WITH B-SPLINES.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Luiz Henrique De Figueiredo. 1996. Surface intersection using a ne arithmetic. In *Proceedings of graphics interface*, Vol. 96. Citeseer, 168–175.

SINTEF Digital. 2021. The SINTEF Spline Library. https://www.sintef.no/en/software/software-applied-mathematics/sisl/.

Albert L Dixon et al. 1908. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society* 6, 4969 (1908), 209236.

Tor Dokken. 1997. Aspects of intersection algorithms and approximation. *Doctor thesis* (1997).

William J Gordon and Richard F Riesenfeld. 1974. B-spline curves and surfaces. In *Computer Aided Geometric Design.* Elsevier, 95–126.

Christoph Martin Hoffmann. 1989. Geometric and Solid Modeling. (1989).

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

Kai Jin and Jinsan Cheng. 2021. On the complexity of computing the topology of real algebraic space curves. *Journal of Systems Science and Complexity* 34 (2021), 809–826.

George A Kriezis, Nicholas M Patrikalakis, and F-E Wolter. 1992. Topological and differential-equation methods for surface intersections. *Computer-Aided Design* 24, 1 (1992), 41–55.

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

Dieter Lasser. 1986. Intersection of parametric surfaces in the Bernstein-Bezier representation. *Computer-Aided Design* 18, 4 (1986), 186–192.

Hongwei Lin, Yang Qin, Hongwei Liao, and Yunyang Xiong. 2013. Affine arithmetic-based B-Spline surface intersection with GPU acceleration. *IEEE transactions on visualization and computer graphics* 20, 2 (2013), 172–181.

Dinesh Manocha and John Canny. 1991. A new approach for surface intersection. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications.* 209–219.

Dinesh Manocha and John F Canny. 1992. Implicit representation of rational parametric surfaces. *Journal of Symbolic Computation* 13, 5 (1992), 485–510.

Nicholas M Patrikalakis. 1993. Surface-to-surface intersections. *IEEE Computer Graphics and Applications* 13, 1 (1993), 89–95.

Jaroslaw R Rossignac and Aristides AG Requicha. 1987. Piecewise-circular curves for geometric modeling. *IBM Journal of Research and Development* 31, 3 (1987), 296–313.

Ramon F Sarraga. 1983. Algebraic methods for intersections of quadric surfaces in GMSOLID. *Computer Vision, Graphics, and Image Processing* 22, 2 (1983), 222–238.

Open Cascade SAS. 2023. Open CASCADE Technology. https://dev.opencascade.org/.

Thomas W Sederberg, David C Anderson, and Ronald N Goldman. 1984. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing* 28, 1 (1984), 72–84.

Thomas W Sederberg and Falai Chen. 1995. Implicitization using moving curves and surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.* 301–308.

Thomas W Sederberg and Ray J Meyers. 1988. Loop detection in surface patch intersections. *Computer Aided Geometric Design* 5, 2 (1988), 161–171.

Jingjing Shen, Laurent Busé, Pierre Alliez, and Neil Dodgson. 2016. A line/trimmed NURBS surface intersection algorithm using matrix representations. *Computer Aided Geometric Design* 48 (2016), 1–16.

Spatial Team. 2023. 3D ACIS Modeler. https://www.spatial.com/products/3d-acis-modeling.

Manuel Ventura and C Guedes Soares. 2012. Surface intersection in geometric modeling of ship hulls. *Journal of Marine Science and Technology* 17 (2012), 114–124.

A EXAMPLE FOR TOPOLOGY DETERMINATION

See the following example for an illustration of the topology determination of curves within the parametric domain.

EXAMPLE 2. *Consider the intersection of the cubic surface* $\mathbf{P}(u,v)$ *given in Example 1 with another cubic surface* $\mathbf{Q}(s,t)$*, in the domain* $(u,v) \in [0,1]\times[0,1], (s,t) \in [-1,1]\times[0,2]$*:*

$$\mathbf{Q}(s,t) := [2st^2+2s, -s^2t^3+s^2-3t^3+3, 2s^3t+6t, s^2t^2+s^2+t^2+1].$$

Substituting $X = \mathbf{Q}(s,t)$ *into the implicit equation* $F(x,y,z,w) = 0$ *computed in Example 1, we get*

$$\begin{aligned}\phi(s,t) &= (s^{18}+21s^{16}+192s^{14}+1000s^{12}+3258s^{10}+6858s^8+9288s^6\\&+7776s^4+3645s^2+729)t^{24}+(3s^{18}+51s^{16}+396s^{14}+1884s^{12}\\&+6090s^{10}+13626s^8+20412s^6+19116s^4+9963s^2+2187)t^{22}\\&+\cdots+(s^{18}+9s^{16}+60s^{14}+252s^{12}+822s^{10}+1926s^8+3356s^6\\&+3996s^4+2673s^2+729).\end{aligned}$$

Compute the resultant $R(s) := \mathrm{Res}_t(\phi, \frac{\partial\phi}{\partial t})$*. By solving* $R(s)=0$ *we get three real roots* $\alpha_1,\alpha_2,\alpha_3$ *in* $s\in[-1,1]$*, as shown in Fig. 14. Here,* α_2 *is rational while* α_1,α_3 *are irrational. Hence their corresponding isolation intervals* $I_i, i=1,2,3$ *are:*

$$\begin{aligned}\alpha_1 &\in [\frac{-942370651714277}{4503599627370496}, \frac{-15077930427428431}{72057594037927936}],\\ \alpha_2 &\in [0,0],\\ \alpha_3 &\in [\frac{3860892779556277}{18014398509481984}, \frac{15443571118225109}{72057594037927936}].\end{aligned}$$

From these isolated intervals, we get the following sets of left helping points and the right helping points in $\mathbb{R}^2$*:*

$$\begin{aligned}P_l := \{P_l^1 &:= \{[-0.2092483191, 0.4824088532], [-0.2092483191, 1.655782000]\},\\ P_l^2 &:= \{[-0.000001, 0.5915023911], [-0.000001, 0.5915169779],\\ &[-0.000001, 1.496790140], [-0.000001, 1.496826202]\},\\ P_l^3 &:= \{[0.2143226030, 0.4773526276], [0.2143226030, 0.9999999967],\\ &[0.2143226030, 1.000000003], [0.2143226030, 1.665001610]\}\},\\ P_r := \{P_r^1 &:= \{[-0.2092483191, 0.4824088532], [-0.2092483191, 0.9999999994],\\ &[-0.2092483191, 1.000000001], [-0.2092483191, 1.655782000]\},\\ P_r^2 &:= \{[0.000001, 0.5915016720], [0.000001, 0.5915176969],\\ &[0.000001, 1.496790139], [0.000001, 1.496826203]\},\\ P_r^3 &:= \{[0.2143226030, 0.4773526276], [0.2143226030, 1.665001610]\}\}.\end{aligned}$$

From these helping points, we immediately get the topology graph of the intersection curve in the parametric domain (s,t)*, as shown in Fig. 14.*

B TRACING PROCEDURE

We fix the starting point as $\mathbf{p} := (b_i, \overline{\beta}_{i,j})$, and the ending point as $\mathbf{q} := (a_{i+1}, \underline{\beta}_{i+1,j})$. Tracing is started from point $\mathbf{p}$ and stopped when meeting point $\mathbf{q}$ or the boundaries.

Two main steps are executed alternatively to generate a set of points on curve C:

$$\mathcal{P} := \{\mathbf{p}_0 = \mathbf{p}, \mathbf{p}_1, \cdots, \mathbf{p}_k\} \tag{19}$$

- Initial estimation. Denote $\mathbf{p}_l = (s,t)$, and the curve C satisfies $\phi(s,t)=0$. The initial estimation of next point is $(s+\Delta s^{(0)}, t+\Delta t^{(0)})$ that

$$\begin{pmatrix}\Delta s^{(0)}\\ \Delta t^{(0)}\end{pmatrix} = sign \cdot \frac{h}{\sqrt{\phi_s^2(s,t)+\phi_t^2(s,t)}}\begin{pmatrix}-\phi_t(s,t)\\ \phi_s(s,t)\end{pmatrix}$$
$$sign = \begin{cases}-1 & -\phi_t(s,t) < 0\\ 1 & otherwise\end{cases} \tag{20}$$

where h is a given step length.

- Iteration with Newton method. Suppose we have $(\Delta s^{(j)}, \Delta t^{(j)})$, then

$$\begin{pmatrix}\Delta s^{(j+1)}\\ \Delta t^{(j+1)}\end{pmatrix} = \begin{pmatrix}\cos\omega^{(j)} & \sin\omega^{(j)}\\ -\sin\omega^{(j)} & \cos\omega^{(j)}\end{pmatrix}\begin{pmatrix}\Delta s^{(j)}\\ \Delta t^{(j)}\end{pmatrix}$$
$$w^{(j)} := \frac{\phi}{\sqrt{(-\Delta t^{(j)}\phi_s + \Delta s^{(j)}\phi_t}} \tag{21}$$

The two steps repeat recursively until the distance of $(\Delta s^{(j)}, \Delta t^{(j)})$ and $(\Delta s^{(j+1)}, \Delta t^{(j+1)})$ are smaller than a given ϵ in l_2 norm. Then the next point of $\mathbf{p}_l$ is defined as $\mathbf{p}_{l+1} = (s+\Delta s^{(j)}, t+\Delta t^{(j)})$. We set $\mathbf{p}_{l+1}$ as starting point and repeat the above two steps.

The whole tracing step from $\mathbf{p}$ ends when the points are stepping to $\mathbf{q}$ or the parameter boundaries.

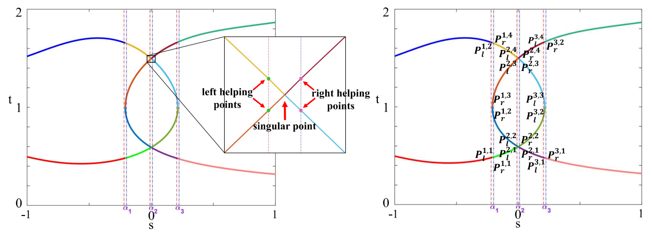


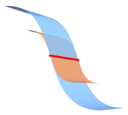
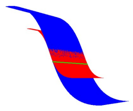
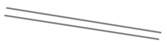
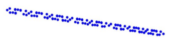
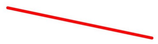
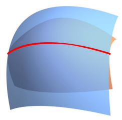
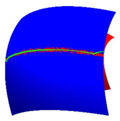
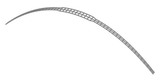
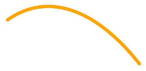
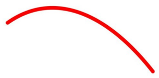
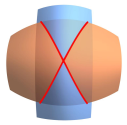
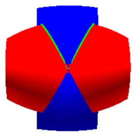
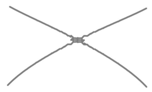
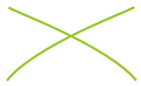
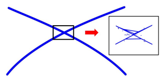
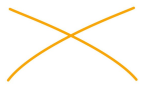
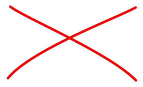
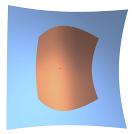
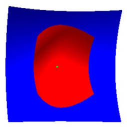
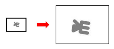
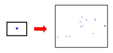
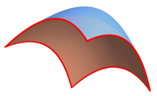
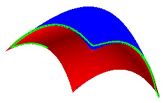
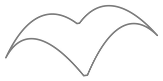
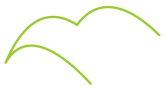
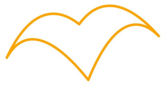
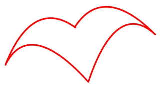
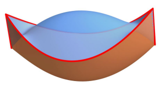
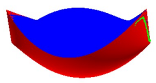
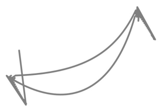
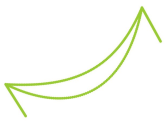
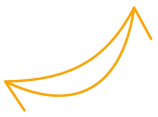
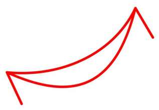
Fig. 14. The image of the curve $\phi(s,t)=0$ in parametric domain $[-1,1]\times[0,2]$. In the left figure, $\alpha_i, i=1,2,3$ are the s-coordinates of s-critical points in the domain $[-1,1]\times[-\infty,+\infty]$. The vertical dashed lines show the small isolation intervals for $\alpha_i, i=1,2,3$. The intersection points of the vertical dashed lines with the curve $\phi(s,t)=0$ give all left helping points $P_l^{i,j}$ and right helping points $P_r^{i',j'}$, marked in the right figure. Note that around the singular point of $\phi(s,t)=0$ shown in the left figure, there can be more than one left helping point or right helping point.

C ADDITIONAL EXPERIMENTAL RESULTS

More experimental results about B-Spline surface intersection are listed in Table 6.

Based on the obtained results, it can be inferred that AA-GPU demonstrates efficient computation of the interior of tangential curves with lower orders; however, its performance deteriorates at the boundary of the curves in examples (1)(2); when the intersection curves are situated at the boundaries of the surfaces in examples (5)(6), or when they possess cross point as depicted in example (3), AA-GPU tends to yield incomplete outcomes. The Mesh Booleans method has a poor performance near cross points or cusps in examples (3)(5)(6); when facing tangential situations in examples (1)(2)(4), it obtains two paralleled lines or a 2D grid region. OCCT tends to neglect isolated intersection points in example (4) and fails to

Table 6. Comparison of the intersections of two B-Spline Surfaces. The first column shows the surfaces and their bi-degrees. The remaining columns represent the intersection results and the consuming times (in seconds) of different algorithms. AA-GPU refers to the affine arithmetic-based B-Spline surface intersection method with GPU acceleration. Mesh Booleans refers to the interactive and robust mesh boolean algorithm. OCCT refers to the intersection algorithm in the Open CASCADE Technology. SISL refers to the intersection algorithm in the SINTEF Spline Library. ACIS refers to the intersection algorithm in 3D ACIS Modeler. N/A indicates that the program reported an error in this example, and the 'NULL' indicates that the result obtained by this method is no intersection.

Surfaces	AA-GPU	Mesh Booleans	OCCT	SISL	ACIS	Ours
 example (1) (3,1)	 1.355	 0.012	NULL 0.280	 0.001	NULL 0.012	 a tangent line 0.006
 example (2) (2,2)	 0.427	 0.016	NULL 0.003	NULL 0.260	 0.016	 a tangential curve 0.015
 example (3) (2,2)	 0.042	 0.013	 0.026	 0.686	 0.008	 two intersect curves 0.027
 example (4) (2,2)	 0.004	 0.008	NULL 0.002	 0.205	 0.001	 a tangential point 0.014
 example (5) (2,2)	 0.226	 0.008	 0.043	NULL 2.82	 0.001	 four boundary curves 0.014
 example (6) (2,2)	 0.173	 0.017	 0.030	NULL N/A	 0.046	 four boundary curves 0.026

accurately capture certain boundary intersections in example (5); additionally, it fails to address the tangential conditions in examples (1)(2). SISL gets a bunch of intersection points near tangential line and tangential point in examples (1)(4), and it obtains unsatisfactory result near cross point in example (3); besides, it returns errors in the cases that two surfaces intersect at their boundaries in examples (5)(6). ACIS can accurately calculate the intersections in most quadratic cases, but still miss the tangential line in example (1). Among all experiments, our method can guarantee the topology of intersections and calculate them within an acceptable time, which is comparable to the computation time of ACIS in correct cases.