

Computing the intersection of two ellipsoids based on a fast algebraic topology determination strategy

Xiao Chu^{a,b}, Kai Li^{a,b}, Xiaohong Jia^{a,b,*}, Jieyin Yang^{a,b}, Jiarui Kang^{a,b}

^a SKLMS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China

^b University of Chinese Academy of Sciences, China

ARTICLE INFO

Editor: Joshua A Levine

Keywords:

Ellipsoids

Intersection

Topology determination

Octree

ABSTRACT

Ellipsoids serve as the most commonly used geometric primitives and bounding volumes in computer-aided design and computer graphics, where an efficient and topologically stable intersection algorithm between two ellipsoids is highly required. Although there has been extensive research on intersections of two general quadrics, ellipsoids have their own specialty in both algebra and geometry which guides to new possibilities to break the bottleneck in intersection computation. In this paper, we use a topology-determination-based strategy in computing the intersection of ellipsoids. Firstly, the topology of the intersection curve is quickly determined using some algebraic discriminants without computing any point on the intersection curve; then an octree strategy is applied to efficiently compute at least one point on each intersection branch; finally, by tracing the branch, we get the complete intersection loci. Plenty of examples show that our algorithm is topologically stable when facing challenging cases including multi-branches, small loops, singular or tangent intersections, and is more efficient compared with existing algorithms.

1. Introduction

Ellipsoids are widely used as geometric primitives and bounding volumes in computer-aided design Wang et al. (2001), computer graphics Choi et al. (2008), etc. An efficient and topologically stable intersection algorithm between two ellipsoids is highly required in these applications. Ellipsoids are one special type of quadrics. There has been extensive theoretical research on the intersections of two general quadrics, including exploring classifications for the morphology of the intersection Tu et al. (2009) and designing algorithms for obtaining optimal parametric equations for the intersection curve Dupont et al. (2008a,b,c). Nevertheless, ellipsoids have their own specialty in both algebra and geometry, which has not been fully considered in existing work but can guide to new possibilities to break the bottleneck in intersection computation.

One challenge in intersection computation lies in the topology correctness of the intersection curve. Traditional methods, such as the subdivision method, the lattice method, and the tracing method, are difficult to capture all branches of the intersection curve. When there are small loops, contact points, or tangent intersections, these methods tend to lose some key components or feature in the intersection. On the other hand, there is also a trade-off in pursuing the topology correctness and the computation efficiency.

* Corresponding author at: SKLMS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China.

E-mail addresses: chuxiao18@mails.ucas.ac.cn (X. Chu), likai@amss.ac.cn (K. Li), xhjia@amss.ac.cn (X. Jia), yangjieyin17@mails.ucas.ac.cn (J. Yang), kangjiarui18@mails.ucas.ac.cn (J. Kang).

<https://doi.org/10.1016/j.cagd.2025.102442>

Received 10 March 2025; Accepted 15 April 2025

Available online 29 April 2025

0167-8396/© 2025 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

In this paper, we use a topology-determination-based strategy in computing the intersection of ellipsoids. Firstly, the topology of the intersection curve is quickly determined using some algebraic discriminants without computing any point on the intersection curve. These discriminants are inherited from some algebraic conditions in Shao and Chen (2023) to decide the intersection topology of two quadrics in $\mathbb{P}\mathbb{R}^3$, the 3-dimensional real projective space. Since ellipsoids are closed surfaces, their intersection curves lie in a bounded region, hence the topology does not change from $\mathbb{P}\mathbb{R}^3$ to \mathbb{R}^3 . Secondly, we use an octree strategy to efficiently compute at least one point on each intersection branch. This step is an important guarantee that we have computed all real branches in the intersection curve. Finally, by tracing each branch of the intersection, we get the complete intersection loci. The algorithm is topologically stable especially when facing challenging cases including multi-branches, small loops, singular or tangent intersections; and the predetermined topology and use of octree make the algorithm more efficient compared with existing algorithms. The main contribution of this paper is as follows.

- An algebraic technique is designed to quickly determine the intersection topology of two ellipsoids using several discriminants.
- An octree strategy is used based on the known number of topology branches to quickly locate all branches of the intersection curve between two ellipsoids.
- An efficient and topologically stable algorithm for computing the intersection curve of two ellipsoids is proposed, which can deal with challenging intersection topology including small loops, contacts, singularities, cusps and tangent curves.

The paper is organized as follows. In Section 2, we review some related works on ellipsoids and the intersection of surfaces. Section 3 introduces the topology determination of the intersection curve of two quadrics. Section 4 provides the algorithm for computing the intersection of two ellipsoids. Section 5 gives examples corresponding to all topology types of the intersection curve of ellipsoids. The paper is concluded in Section 6.

2. Related works

2.1. Ellipsoids

Ellipsoids have been extensively used in a variety of applications as bounding volumes, including but not limited to robotics Lee et al. (2017), computer vision Shuai et al. (2016), virtual reality Wang (2014), molecular dynamics Ghossein and Lévesque (2013). For analyzing the positional relationship between two ellipsoids, Wang et al. (2001) proposes a simple algebraic condition based on the roots of a characteristic polynomial to determine whether two ellipsoids are separate, touching or overlapping. In Jia et al. (2011), this condition is restructured into simpler computations, allowing it to be applied for continuous collision detection between two moving ellipsoids. Later, Jia et al. (2020) presents the complete classification of arrangements of two ellipsoids, which can be viewed as the intersection of ellipsoids at a volumetric level, and proposes an efficient algorithm to determine the arrangement for two given ellipsoids. When it comes to the intersection computation of two ellipsoids, ellipsoids benefit from the properties of quadrics. Wang (2002) gives an overview of quadrics, including their algebraic background, intersection computation, and various applications in geometric modeling. Typically, methods for computing the intersection curve of two ellipsoids follow those used for quadrics.

2.2. Intersection of two quadrics

There is extensive literature on the computation of the intersection of two quadrics, especially on the topology determination of such intersection. We inherit the notation in Levin (1976) and use QSIC to represent the intersection curve of two quadrics. Levin (1976) proposes an algorithm for calculating the QSIC, proving that the intersection between any two quadric surfaces lies in a ruled quadric surface. The parametric representation of the QSIC is then provided by substituting the parametric representation of this ruled quadric into the equation of one of the input quadrics. Levin (1979) presents an improvement to the method in Levin (1976), but it is unable to provide structural information about the QSIC, such as reducibility and singularity.

To address these issues, a series of improved intersection algorithms have been proposed. Farouki et al. (1989) provides the condition for a degenerate intersection through the multivariate polynomial factorization. Wang et al. (2002) presents an algebraic method for classifying and parameterizing the QSIC based on a birational mapping between the QSIC and a plane cubic curve. After that, Wang et al. (2003) enhances Levin's method and provides an approach to classify the morphology of an irreducible QSIC. Dupont et al. (2008a,b,c) present exact and efficient algorithms for computing a near-optimal parameterization of QSICs. Tu et al. (2009) presents a method to classify QSIC using signature sequences, providing all 35 types of QSICs in three dimensional real projective space $\mathbb{P}\mathbb{R}^3$. Shao and Chen (2023) proposes a new approach for classifying the topology of the QSIC in $\mathbb{P}\mathbb{R}^3$ using a set of easily computable discriminants.

2.3. Intersection of two general surfaces

Classical surface intersection methods can be broadly categorized into four types: algebraic methods Sarraga (1983); Grandine and Klein (1997); Hass et al. (2007), subdivision methods Dokken et al. (1989); Park et al. (2020), lattice methods Rossignac and Requicha (1987), and tracing methods Bajaj et al. (1988); Barnhill and Kersey (1990); Krishnan and Manocha (1997); Ventura and Guedes Soares (2012). Algebraic methods guarantee the topological correctness of intersection curves, but their efficiency tends to be lower due to the need for implicitization and solving high-degree equation systems. Subdivision and lattice methods are more universal, but

Table 1
The classification of real root pattern by discriminants.

Classification label	Discriminants	Root pattern
(R1)	$D_4 > 0 \wedge D_3 > 0 \wedge D_2 > 0$	$\{1, 1, 1, 1\}$
(R2)	$D_4 > 0 \wedge (D_3 \leq 0 \vee D_2 \leq 0)$	$\{\}$
(R3)	$D_4 < 0$	$\{1, 1\}$
(R4)	$D_4 = 0 \wedge D_3 > 0$	$\{2, 1, 1\}$
(R5)	$D_4 = 0 \wedge D_3 < 0$	$\{2\}$
(R6)	$D_4 = 0 \wedge D_3 = 0 \wedge D_2 > 0 \wedge E = 0$	$\{2, 2\}$
(R7)	$D_4 = 0 \wedge D_3 = 0 \wedge D_2 > 0 \wedge E \neq 0$	$\{3, 1\}$
(R8)	$D_4 = 0 \wedge D_3 = 0 \wedge D_2 < 0$	$\{\}$
(R9)	$D_4 = 0 \wedge D_3 = 0 \wedge D_2 = 0$	$\{4\}$

may miss small loops and isolated intersection points. Tracing methods are currently the most widely used and efficient approach, relying on the local geometric properties of surfaces to trace out intersection curve. However, calculating the starting points remains a significant challenge. Therefore, guaranteeing both topological correctness and high efficiency in surface intersection algorithms is a highly difficult task Patrikalakis (1993); Kasik et al. (2005); Piegł (2005); Ye et al. (2024).

3. Preliminaries

We first review some previous results on classification of Quadratic Surface Intersection Curves (QSICs).

3.1. Characteristic polynomial of a quadric pencil

Consider two quadratic surfaces, or quadrics

$$\mathcal{Q}_A : f(X) := X^T A X = 0, \quad \mathcal{Q}_B : g(X) := X^T B X = 0, \quad (1)$$

where $X = (x, y, z, w)$ is homogeneous coordinate of a point $(x : w, y : w, z : w)$ in $\mathbb{P}\mathbb{R}^3$, and $A, B \in \mathbb{R}^{4 \times 4}$ are real symmetric matrices. The quadric pencil $\mathcal{Q}_\lambda(X) = X^T(\lambda A - B)X = 0$ always passes through the intersection curve C of \mathcal{Q}_A and \mathcal{Q}_B .

Define the *characteristic polynomial* of $\mathcal{Q}_\lambda(X)$ as $f(\lambda) = \det(\lambda A - B)$. Clearly, $f(\lambda)$ is a quartic polynomial. We first assume that $f(\lambda) \neq 0$. Without loss of generality, we can also assume that $\det(A) \neq 0$, otherwise we can find $\lambda_0 \in \mathbb{R}$ such that $\det(\lambda_0 A - B) \neq 0$, and replace the matrix A by $\lambda_0 A - B$ without changing the intersection curve.

The following lemma describes how to determine the real root pattern of a quartic polynomial.

Lemma 1 (Yang (1999)). *Let $f(\lambda) = a\lambda^4 + b\lambda^3 + c\lambda^2 + d\lambda + e$ be a quartic polynomial. Then the real root pattern of $f(\lambda) = 0$ can be determined by conditions in Table 1, where*

$$\begin{aligned} D_2 &= 3b^2 - 8ac, \\ D_3 &= b^2c^2 - 3b^3d - 6ab^2e + 14abcd - 4ac^3 + 16a^2ce - 18a^2d^2, \\ D_4 &= b^2c^2d^2 - 4b^2c^3e + 18b^3cde + 144ab^2ce^2 - 6ab^2d^2e - 4b^3d^3 - 27b^4e^2 - 80abc^2de + 18abcd^3 \\ &\quad - 192a^2bde^2 - 4ac^3d^2 - 128a^2c^2e^2 + 16ac^4e + 144a^2cd^2e - 27a^2d^4 + 256a^3e^3 \\ E &= 8a^2d + b^3 - 4abc, \end{aligned}$$






and the integer sequence in the third column represents the pattern of real roots. For example, $\{1, 1, 1, 1\}$ represents that $f(x)$ has four distinct real roots, $\{\}$ denotes that $f(x)$ does not have a real root, while $\{2, 1, 1\}$ means that $f(x)$ contains one double real root and two simple real roots.

3.2. Classification and determination of QSICs

Tu et al. (2009) first classifies QSICs into 35 types in $\mathbb{P}\mathbb{R}^3$, the three dimensional real projective space. Shao and Chen (2023) provides a set of discriminants to decide these 35 types of QSIC (see Table 7 in Shao and Chen (2023)) based on different real root patterns of the characteristic polynomial $f(\lambda) = 0$, labeled (R1) to (R9). To avoid a lengthy context, we show only case (R1) of Lemma 2 in their results.

Lemma 2 (Shao and Chen (2023)). **[Case (R1): $f(\lambda) = 0$ has four distinct real roots]** *Let $C(\mu, \lambda) = \det(\mu I - (\lambda A - B)) = \mu^4 + c_3(\lambda)\mu^3 + c_2(\lambda)\mu^2 + c_1(\lambda)\mu + c_0(\lambda)$ and $c_{31}(\lambda) := c_1(\lambda)c_3(\lambda)$. If $f(\lambda) = \det(\lambda A - B) = a\lambda^4 + b\lambda^3 + c\lambda^2 + d\lambda + e$ has four distinct real roots, then Table 2 gives a classification scheme for the intersection of \mathcal{Q}_A and \mathcal{Q}_B , where*

Table 2The classification of the QSIC when $f(\lambda) = 0$ has four distinct real roots Shao and Chen (2023).

Further conditions for classifying QSIC corresponding to case (R1)					case
The eigenvalues of A are all positive or negative.					
$a < 0$	$r > 0$	$\mu_1 = \frac{-3q + \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a},$ $\mu_2 = \frac{-3q - \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a}$	$c_{31}(\mu_1) > 0 \wedge c_2(\mu_1) > 0$ or $c_{31}(\mu_2) > 0 \wedge c_2(\mu_2) > 0$		
	$r < 0$	$q > 0$	$\mu_1 = -\frac{b}{4a}, \mu_2 = \frac{-3q - \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a}$	$c_{31}(\mu_1) \leq 0 \vee c_2(\mu_1) \leq 0$ and $c_{31}(\mu_2) \leq 0 \vee c_2(\mu_2) \leq 0$	
		$q < 0$	$\mu_1 = \frac{-3q + \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a}, \mu_2 = -\frac{b}{4a}$		
		$r = 0$	$\mu_1 = -\sqrt{-\frac{p}{3}} - \frac{b}{4a}, \mu_2 = \sqrt{-\frac{p}{3}} - \frac{b}{4a}$		
$a > 0$	$r > 0$	$\mu_1 = -\frac{b}{4a}$	$c_{31}(\mu_1) > 0 \wedge c_2(\mu_1) > 0$		
	$r < 0$	$q > 0$	$\mu_1 = \frac{-3q + \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a}$	$c_{31}(\mu_1) \leq 0 \vee c_2(\mu_1) \leq 0$ and $c_{31}(\mu_2) \leq 0 \vee c_2(\mu_2) \leq 0$	
		$q < 0$	$\mu_1 = \frac{-3q - \sqrt{9q^2 - 32rp}}{4p} - \frac{b}{4a}$		
		$r = 0$	$\mu_1 = -\frac{q}{p} - \frac{b}{4a}$		

$$p = \frac{c}{a} - \frac{3b^2}{8a^2},$$

$$q = \frac{d}{a} - \frac{bc}{2a^2} + \frac{b^3}{8a^3},$$

$$r = -\frac{3b^4}{256a^4} + \frac{b^2c}{16a^3} - \frac{bd}{4a^2} + \frac{e}{a}.$$

For other situations (R2)-(R9) of the real root pattern of $f(\lambda) = 0$, Shao and Chen (2023) provides similar classification discriminants in their Theorem 3.2 to Theorem 3.10.

4. Intersection computation of two ellipsoids

The strategy of our algorithm for computing the intersection curve of two ellipsoids is to perform a topology determination before calculating the intersection. Once the topology of the intersection curve is known, we are left to search for one starting point on each real branch of the intersection curve and then use the tracing technique to finish the computation of the whole branch. The main steps of our algorithm as shown in Algorithm 5 are:

1. Determine the topology of the intersection curve by Algorithm 1.
2. Compute at least one starting point in each branch of the intersection curve.
3. Tracing all branches of the intersection curve from the starting points.

The advantages of this topology-based strategy are as follows. 1. Topology stability. The topology determination step avoids the loss of branches, especially those tiny loops or single contact points, which is a challenge in all existing algorithms; 2. Efficiency. Once the topology is determined, the number N of real intersection branches is known, then we have different processing according to N : if $N = 0$, the algorithm terminates; if $N = 1$, we need only compute one starting point; if $N = 2$, we compute two starting points. Note that the intersection of two ellipsoids can have at most two real branches. This saves a lot of computational cost.






















4.1. Intersection topology determination

Ellipsoids are a special type of quadric that are confined within a bounded region in \mathbb{R}^3 , so is their intersection curve. Therefore, although both the results of Tu et al. (2009) and Shao and Chen (2023) are in the real projective space $\mathbb{P}\mathbb{R}^3$, when it comes to the intersection curve of two ellipsoids, the results also apply to the affine space \mathbb{R}^3 .

We extract all possible intersection topology of two ellipsoids, i.e., those curves confined in a finite region in \mathbb{R}^3 , from Tu et al. (2009), and show these 21 topology and their corresponding real root pattern in the characteristic polynomial $f(\lambda) = 0$ in Table 3. Then for a given ellipsoid pair, we first use Lemma 1 to decide the real root pattern of $f(\lambda) = 0$, which leads to root classifications (R1)-(R9) in Table 3. Then we further use Lemma 2 to decide the intersection topology (Lemma 2 only presents the root class (R1), for other classes (R2)-(R9), readers can refer to Theorem 3.1- Theorem 3.10 in Shao and Chen (2023) for a similar determination idea). The above topology determination method is shown in Algorithm 1.

Table 3

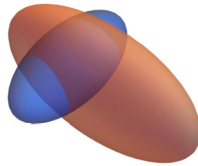
Topology classification of intersection curve between two ellipsoids. (R1) – (R9) refer to different root patterns in Lemma 1.

The root pattern of $f(\lambda) = 0$	Topology	Case	The root pattern of $f(\lambda) = 0$	Topology	Case
(R1) {1, 1, 1, 1}		1	(R6) {2, 2}		24
		2			25
(R3) {1, 1}		3			30
(R4) {2, 1, 1}		5	(R7) {3, 1}		9
		6			19
		7			20
		13			21
		14			22
		15	(R9) {4}		27
		16			33
(R5) {2}		17			

Algorithm 1 Topology determination for the intersection of two ellipsoids.**Input:** $Q_A : f(X) := X^T A X = 0$, $Q_B : g(X) := X^T B X = 0$.**Output:** The topology of the intersection curve of Q_A and Q_B .Step1: Determine the root pattern (R1)-(R9) of the characteristic polynomial $f(\lambda) = 0$ by Lemma 1.

Step2: Go to Table 3 for the possible intersection topology types.

Step3: Determine the final intersection topology type by computing the discriminants in Lemma 2.

**Fig. 1.** Illustration of the ellipsoids in Example 4.1.**Example 4.1.** Given two ellipsoids as illustrated in Fig. 1:

$$Q_A : x^2 + \frac{y^2}{4} + z^2 - 1 = 0, \quad Q_B : x^2 - 2x + 9y^2 + \frac{9}{4}z^2 - 8 = 0. \quad (2)$$

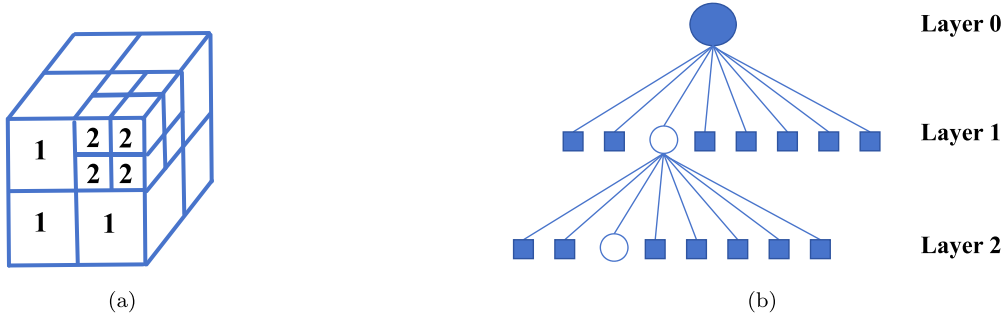


Fig. 2. (a) Illustration of divided sub-spaces, with the number on each box representing its level in the octree; (b) The tree representation for octree, where sub-nodes represent the eight sub-spaces corresponding to (a).

The coefficient matrices of Q_A and Q_B can be extracted as

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 9 & 0 & 0 \\ 0 & 0 & \frac{9}{4} & 0 \\ -1 & 0 & 0 & -8 \end{bmatrix}.$$

The characteristic polynomial corresponding to Q_A and Q_B is

$$f(\lambda) = \det(\lambda A - B) = -\frac{1}{4}\lambda^4 + \frac{189}{16}\lambda^3 - \frac{1737}{16}\lambda^2 + \frac{4293}{16}\lambda - \frac{729}{4}. \quad (3)$$

It is clear that

$$a = -\frac{1}{4}, \quad b = \frac{189}{16}, \quad c = -\frac{1737}{16}, \quad d = \frac{4293}{16}, \quad e = -\frac{729}{4}. \quad (4)$$

By Lemma 1, we obtain

$$D_2 = 201.48 > 0, \quad D_3 = 1.43 \times 10^5 > 0, \quad D_4 = 4.61 \times 10^8 > 0, \quad (5)$$

and the root pattern of the characteristic polynomial $f(\lambda)$ is determined as (R1).

Now we use Lemma 2 for further topology determination. By computing $C(\mu, \lambda) = \det(\mu I - (\lambda A - B)) = \mu^4 + c_3(\lambda)\mu^3 + c_2(\lambda)\mu^2 + c_1(\lambda)\mu + c_0(\lambda)$, we have

$$\begin{aligned} c_2(\lambda) &= -\frac{3}{4}\lambda^2 + \frac{131}{16}\lambda - \frac{135}{2}, \\ c_{31}(\lambda) &= c_1(\lambda)c_3(\lambda) = -\frac{25}{16}\lambda^4 + \frac{285}{8}\lambda^3 - \frac{20951}{64}\lambda^2 + \frac{68247}{64}\lambda - \frac{4131}{4}. \end{aligned}$$

Next, we calculate the discriminants in Table 2:

$$a = -0.25 < 0, \quad r = -9.77 \times 10^3 < 0, \quad q = -4.00 \times 10^3 < 0,$$

$$\mu_1 = 1.73, \quad \mu_2 = 11.81,$$

$$c_2(\mu_1) = -55.59 < 0, \quad c_{31}(\mu_2) = -5.82 \times 10^3 < 0.$$

Therefore, the topology of the intersection curve between Q_A and Q_B is case 1, i.e., two separate real loops, in Table 3.

4.2. Finding a point on each intersection branch

Once the intersection topology of the two ellipsoids is decided, we know the number of real branches of the intersection. Then our next goal is to find at least one point on each branch of the intersection curve. These points are computed sequentially by branches. Next, we show how to choose the starting points in each branch. In Table 4, all the starting points that we prefer, for all possible topologies, are colored red. During this process, we use the adaptive octree as shown in Fig. 2. See Meagher (1982) for details of the octree. The prior information from topological determination enables us to use the octree search until all the required starting points are located.

4.2.1. Detecting a starting point on an arbitrary intersection branch

We now show how to find a starting point on an arbitrary intersection branch of two ellipsoids. The strategy is to use an octree starting from a region B_0 that contains the whole intersection curve.

Table 4

Preferred starting points on every intersection branch. (For interpretation of the colors in the tables, the reader is referred to the web version of this article.)

$f(\lambda)$	Topology	Starting point(s)
(R1)		
(R3)		
(R4)		
(R5)		
$f(\lambda)$	Topology	Starting point(s)
(R6)		
(R7)		
(R9)		

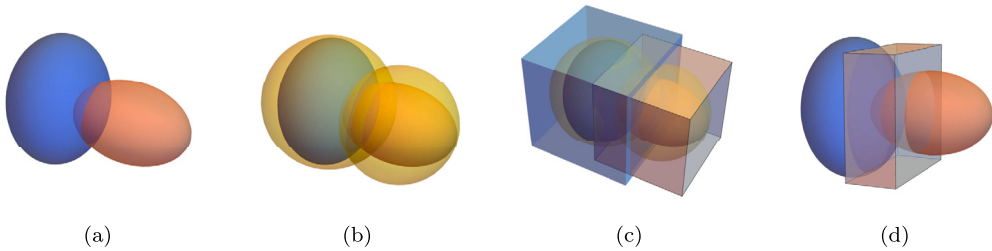


Fig. 3. (a) Two given ellipsoids; (b) Two spheres containing the two ellipsoids separately, with the sphere centers located at the ellipsoid center and radius as the length of the semi-major axis of the ellipsoid; (c) Two AABs containing the two spheres separately; (d) The final intersection region B_0 of two AABs.

The starting region B_0 is created in the following way. Firstly, we use a sphere to cover the corresponding ellipsoid. Then, two AABs (axis-aligned bounding boxes) are created to enclose each sphere separately. The overlap region of the two AABs is selected as the root node of the octree, represented by B_0 . Fig. 3 shows the process of creating B_0 (in gray) for two ellipsoids.

Next, we show how to find an intersection point in a given region, say B_0 . We construct an octree starting from the root node B_0 . When performing the depth-first search (DFS) in the octree, we focus only on the octants that contain portions of both surfaces Q_A and Q_B . The following proposition gives a sufficient condition for an octant containing a portion of a given ellipsoid with equation $f(x, y, z) = 0$.

Proposition 4.1. *Considering the eight vertices of the box of an octant, if there exists one vertex inside the ellipsoid and another vertex outside the ellipsoid, that is,*



Fig. 4. (a) An example of a box corresponding to Proposition 4.1; (b) An illustration of a box that does not satisfy Proposition 4.1 but still contains a portion of the ellipsoid.

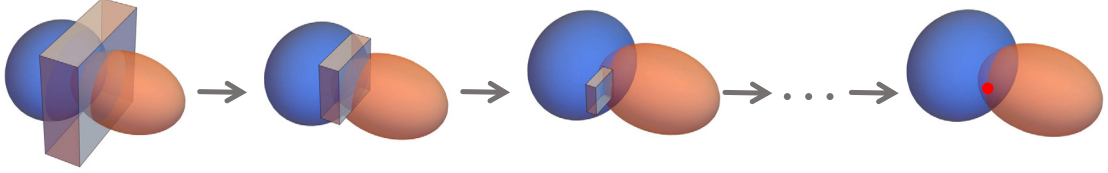


Fig. 5. Construction of an octree for finding a starting point.

$$\text{if } \exists P_i = (x_i, y_i, z_i), P_j = (x_j, y_j, z_j), \text{ s.t. } f(x_i, y_i, z_i) > 0 \text{ and } f(x_j, y_j, z_j) < 0, \quad (6)$$

where P_i and P_j are vertices of the octant, $1 \leq i, j \leq 8$, and $i \neq j$, then the octant must contain a portion on the ellipsoid.

Proposition 4.1 is only a sufficient condition for an octant containing a portion of the ellipsoid. For example, in Fig. 4(a), the point P_1 is outside the ellipsoid and the point P_2 is inside the ellipsoid. Therefore, the octant box contains a portion of the ellipsoid. However, in Fig. 4(b), $f(x_i, y_i, z_i) > 0$ for the eight vertices (or $f(x_i, y_i, z_i) < 0$ for the eight vertices), but it also contains a portion of the ellipsoid.

Definition 4.2. If an octant satisfies Proposition 4.1 for both ellipsoids Q_A with equation $f(x, y, z) = 0$ and Q_B with equation $g(x, y, z) = 0$, we call this octant an available octant.

Clearly, the octant box in Fig. 4(b) is not available, since the shown ellipsoid, which is one of the two surfaces, does not satisfy Proposition 4.1.

When DFS is performed on the octree, at first we subdivide only the available octant (if it exists) at every level until the size of the available octant meets the given threshold ϵ . Here, two situations can happen:

1. In every level of subdivision, there always exists an available octant (that allows for a further subdivision), until the size of one octant reaches ϵ .
2. In a certain level of subdivision, none of the eight octants is available, but the size of the octants has not reached ϵ .

Definition 4.3. Starting from an available octant, if the DFS ends with situation 1 in Definition 4.2, we call this octant a valid octant, otherwise an invalid octant.

Based on the above analysis, for an octant it can be

1. available and valid;
2. available but invalid;
3. not available.

Algorithm 2 summarizes the above analysis on finding a starting intersection point in a given region. Fig. 5 gives an illustration of this process. We start from the root node, subdivide it into eight leaf octants, pick one available octant, and repeat the process until the size of the octant reaches ϵ . If at a certain level the eight octants are not available but the size of the octants has not reached ϵ , then this routine fails to find a valid octant and suggests that the situation in Fig. 4(b) occurs. For such situations, we return to the first layer of the octree, choose another octant that we have given up on in the first round search, take this octant as the root node, and repeat the above process.

Remark 4.4. In the algorithm, a geometric tolerance ϵ is given. For an octant of size less than ϵ , if Equation (6) holds for both surfaces $f(x, y, z) = 0$ and $g(x, y, z) = 0$, then we stop further subdivision and approximately take the center point of the octant as an intersection point of these two surfaces under the tolerance ϵ . Then the maximum depth Dep_{max} where we stop the subdivision is $Dep_{max} = \lceil \log_2(\frac{d}{\epsilon}) \rceil$, where d is the size of the box at the root node.

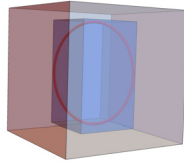


Fig. 6. Illustration of B_0 (the gray box), B_1 (the blue box) and $B_0 \setminus B_1$ (the difference of the gray box and the blue box). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Algorithm 2 Finding a starting point in region B_0 .

Input: $Q_A : f(X) := X^T A X = 0$, $Q_B : g(X) := X^T B X = 0$, Dep_{max}

Output: A starting point p_i on the intersection curve

$currentNode := B_0$, $M := 0$, $Flag F := 0$;

function $getStartingPoint(currentNode, M)$

$M \leftarrow M + 1$; /* M represents the octree level */

Divide $currentNode$ into eight octants, named $childNode[i]$, $i = 1, \dots, 8$;

if $M < Dep_{max}$ **then**

for $i = 1$ to 8 **do**

if $childNode[i]$ is available **then**

$currentNode \leftarrow childNode[i]$;

$getStartingPoint(currentNode, M)$; /* recursive call for DFS */

end if

end for

else if $M = Dep_{max}$ **then**

for $i = 1$ to 8 **do**

if $childNode[i]$ is available **then**

/* $childNode[i]$ is valid */

$F := 1$;

Find an intersection point as p_i and terminate the algorithm.

end if

end for

if $F = 0$ **then**

No intersection point is found and continue to search it.

end if

end if

end function

4.2.2. Detecting a starting point on the second intersection branch

Since the intersection of two ellipsoids can contain at most two real branches, when the topology determination indicates the number of intersection branches $N = 2$, we continue to find the next starting point located on the second branch. This point is searched within the region B_0 subtracting the first real branch. Let B_1 be the AABB enclosing the first real intersection branch. We first search for the second intersection starting point within $B_0 \setminus B_1$, if succeed, then we stop; if this search fails, we continue to search for the intersection starting point within B_1 . The illustration of B_0 , B_1 and $B_0 \setminus B_1$ is shown in Fig. 6.

Search for a starting point in $B_0 \setminus B_1$. We divide $B_0 \setminus B_1$ into six AABBs by the six supporting planes of B_1 , represented as $B_0 \setminus B_1 = \bigcup_{k=1}^6 \bar{B}_1^k$. Then Algorithm 2 is applied to each \bar{B}_1^k until a starting point is found, which we take as the starting point for the second branch. If the above process does not find the starting point, a further search in the interior of B_1 is required as follows.

Search for a starting point in the interior of B_1 . By the tracing method, we can get a sequence of real points $P_i, i = 1, \dots, k$ on the first intersection branch. We divide the point sequence into $C_0 := \{P_i\}, i = 1, \dots, \lfloor k/2 \rfloor$ and $C_1 := \{P_i\}, i = \lfloor k/2 \rfloor + 1, \dots, k$. Construct an AABB for C_0 and C_1 separately, and let them be B_{seg}^1 and B_{seg}^2 as shown in Fig. 7 (a). Then we use Algorithm 2 to search for a starting point in $B_1 \setminus \{B_{seg}^1 \cup B_{seg}^2\}$. If fails, we further partition C_0 and C_1 to four point sequences, and find four AABBs bounding these point sequences as shown in Fig. 7 (b), and repeat the above process until a starting point is finally found.

Algorithm 3 provides the detailed process about searching for the starting point with the given partition number M_{div} in region $B_1 \setminus B_{seg}$, $B_{seg} = \{B_{seg}^j\}_{j=1}^{M_{div}}$. The complete outline of finding a starting point for the second intersection branch, including the process mentioned above, is provided in Algorithm 4.

4.3. Computing the intersection curve of two ellipsoids

After getting a starting point on each branch, we computing the whole intersection curve by the tracing method, where the tracing direction is given by the local differential geometry Bajaj et al. (1988) of the curve. See the tracing methods in Mortenson (1997); Barnhill and Kersey (1990); Abdel-Malek and Yeh (1996). A sequence of ordered intersection points is generated using the tracing method and connected by line segments to construct the intersection curve. During the tracing process, we adjust the tracing step

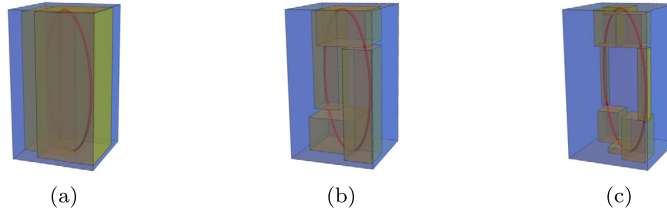


Fig. 7. Illustration of the sequences of AABBs (the yellow boxes) constructed from the corresponding point sequences. (a) Two point sequences; (b) Four point sequences; (c) Eight point sequences.

Algorithm 3 Finding a starting point in the interior of B_1 .

Input: $\mathcal{Q}_A : f(X) := X^T A X = 0$, $\mathcal{Q}_B : g(X) := X^T B X = 0$, M_{div} (number of partitions)

Output: A starting point \mathbf{p}_2 in the interior of B_1

```

currentNode :=  $B_1$ ,  $M := 0$ ;
function getInteriorStartingPoint(currentNode,  $M$ ,  $M_{div}$ )
     $M \leftarrow M + 1$ ; /*  $M$  represents the octree level */
    Divide currentNode into eight octants, named  $childNode[i]$ ,  $i = 1, \dots, 8$ ;
    for  $i = 1$  to 8 do
        if  $childNode[i] \cap \{B_{seg}^j\}_{j=1}^{2^{M_{div}}} \neq \emptyset$  then
            currentNode  $\leftarrow childNode[i]$ ;
            getInteriorStartingPoint(currentNode,  $M$ ,  $M_{div}$ ). /* recursive call for finding box in  $B_1 \setminus B_{seg}$  */
        else
            getStartingPoint( $childNode[i]$ ,  $M$ ). /* call Algorithm 2 for initial  $B_v = childNode[i]$  */
        end if
    end for
end function

```

Algorithm 4 Finding a starting point for the second intersection branch.

Input: $\mathcal{Q}_A : f(X) := X^T A X = 0$, $\mathcal{Q}_B : g(X) := X^T B X = 0$, Dep_{max} , B_0 , B_1

Output: A starting point \mathbf{p}_2 for the second intersection branch

```

1: Compute  $B_0 \setminus B_1$ , which contains six AABBs  $\{\bar{B}_1^k\}_{k=1}^6$ ;
2:  $k := 1$ , Flag  $F := 0$ ;
3: while  $k \leq 6$  do
4:   getStartingPoint( $\bar{B}_1^k$ , 0);
5:   if  $F = 1$  then
6:     Find an intersection point as  $\mathbf{p}_2$  and terminate the algorithm.
7:   else
8:      $k \leftarrow k + 1$ ;
9:   end if
10: end while
11:  $M_{div} := 1$ ;
12: while  $F = 0$  do
13:   getInteriorStartingPoint( $B_1$ , 0,  $M_{div}$ );
14:   if  $F = 1$  then
15:     Find an intersection point as  $\mathbf{p}_2$  and terminate the algorithm.
16:   else
17:      $M_{div} = M_{div} + 1$ ; /* adaptively bisect the point sequence of the first branch */
18:   end if
19: end while

```

size based on the curvature of the intersection curve, reducing the step size in regions with high curvature to control the error of the intersection curve.

The difficulty in the traditional tracing method in computing surface intersections is the fast and complete locating of starting points on each branch. This is also the advantage of our current method, since the topology determination step quickly tells the information of the branch number, and the octree strategy helps efficiently get the starting points on each branch.

The outline of the whole algorithm for computing the intersection of two ellipsoids is shown in Algorithm 5. The outline is, we first determine the topology of the intersection curve by Algorithm 1 and get the number N of real branches. If $N = 0$, the algorithm terminates. If $N \neq 0$, we find a starting point using Algorithm 2. If $N = 2$, we further find a second starting point on the other branch by Algorithm 4. In our algorithm, we first normalize the leading coefficients of the input ellipsoid equations. During the tracing process, if the values of both implicit equations at the point obtained through Newton iteration are less than the tolerance of 10^{-8} , the point is considered to be an intersection point. If a real branch is a real loop, we trace the whole intersection curve. Otherwise, if the real branch contains only a single isolated point, we use Newton iteration to obtain a more accurate intersection point, which is then regarded as the final isolated point.

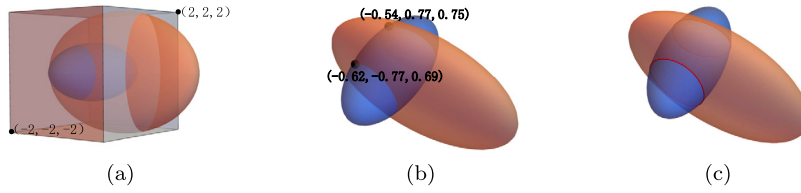


Fig. 8. The illustration of Example 4.2. (a) The box (in gray) corresponding to the root node; (b) Two starting points; (c) The intersection curve.

Example 4.2. Consider the ellipsoids Q_A and Q_B in Example 4.1. Based on the topology determination in Example 4.1, we observe that the intersection of Q_A and Q_B corresponds to case 1 in Table 3, which involves two loops.

The center of Q_A is $(0, 0, 0)$, and the length of its semi-major axis is 2. Construct a cube C_A centered in the center of Q_A , with side lengths equal to the length of the major axis of Q_A . Similarly, the center of Q_B is at the point $(1, 0, 0)$, and its semi-major axis length is 3. Construct a cube C_B centered at the center of Q_B , with side lengths equal to the major axis length of Q_B . The overlap region of C_A and C_B defines the root node, with its vertices at $(-2, -2, -2)$, $(-2, -2, 2)$, $(-2, 2, -2)$, $(-2, 2, 2)$, $(2, -2, -2)$, $(2, -2, 2)$, $(2, 2, -2)$, $(2, 2, 2)$, as shown in Fig. 8(a).

Based on the root node, we construct an octree and first identify a starting point on an arbitrary intersection branch by calling the Algorithm 2. This yields a starting point at $(-0.54, 0.77, 0.75)$, from which the first branch is traced. Then Algorithm 4 is called to locate a starting point on the second intersection branch. We search for this starting point outside the AABB B_1 , which encloses the first branch, and find the starting point $(-0.62, -0.77, 0.69)$. This allows us to trace the second branch, after which the algorithm terminates. Two starting points are shown in Fig. 8(b), and intersection curve is shown in Fig. 8(c).

Algorithm 5 Computing the intersection curve of two ellipsoids.

Input: Implicit equations of two ellipsoids Q_A and Q_B

Output: The intersection curve of Q_A and Q_B

```

1: Determine the topology of the intersection curve between  $Q_A$  and  $Q_B$  by Algorithm 1.
2: if no intersection then
3:   return "No intersection".
4: else if one intersection branch then
5:   Call Algorithm 2;
6:   if there is an isolated point then
7:     return The point obtained by Newton iteration from  $p_1$ .
8:   else if there is a loop then
9:     return The loop traced from  $p_1$ .
10:  end if
11: else if two intersection branches then
12:   Call Algorithm 2 and Algorithm 4;
13:   if there are two isolated points then
14:     return Two points obtained by Newton iteration from  $p_1$  and  $p_2$ .
15:   else if there are two loops then
16:     return Two loops traced from  $p_1$  and  $p_2$ .
17:   else if there are one isolated point and one loop then
18:     Trace ten points  $TP_{i=1}^{10}$  from the first starting point;
19:     if  $\max_i \{distance(TP_{i=1}^{10}, p_1)\} < 1 \times 10^{-3}$  then
20:       return The point obtained by Newton iteration from  $p_1$  and the loop traced from  $p_2$ .
21:     else
22:       return The loop traced from  $p_1$  and the point obtained by Newton iteration from  $p_2$ .
23:     end if
24:   end if
25: end if

```

5. Examples and discussion

We implement our algorithm in numerous examples to illustrate its performance in Section 5.1. The performance comparison with Open CASCADE Technology (OCCT) SAS (2023) and the commercial geometry kernel ACIS Team (2023) shows the advantage of our algorithm in both topology correctness and computation efficiency. Discussions including generality, topology correctness, efficiency and numerical stability of our algorithm are presented in Section 5.2.

5.1. Examples

Examples are organized from three perspectives. Firstly, we provide corresponding examples for all topology cases of the intersection curve between two ellipsoids. Secondly, we present different examples that unlock different levels of the algorithm. Finally, we evaluate the proposed algorithm on 100,000 randomly generated pairs of ellipsoids. Moreover, we compare the topology correctness

and efficiency of our algorithm with OCCT and ACIS. The coefficients of the ellipsoid equations are composed of floating-point numbers. The proposed algorithm is implemented in C++, on a PC with RAM 32 GB and 13th Gen Intel(R) Core(TM) i9-13900H @2.60 GHz. Note that both OCCT and ACIS compute the intersection of two ellipsoids using an intersection function on NURBS.

5.1.1. Examples with all possible intersection topology

Table 5 provides examples that cover all possible intersection topology types of two ellipsoids. For some challenging topology, such as that the two surfaces contact at a single point or intersect along a double curve, OCCT fails to give the correct intersection topology, whereas our algorithm is robust in computing these intersections. Our algorithm also shows much higher computational efficiency compared to OCCT and ACIS.

5.1.2. Examples reaching different levels of our algorithm

In Table 6, we present three examples that reach different levels of our algorithm. The first example has only one real intersection branch; hence it needs only to compute one starting point and then trace the whole intersection curve. The other two examples have both two real intersection branches, hence it needs to compute two starting points, with each on a different real intersection branch. The difference is that the second example gets the second starting point by searching outside B_1 , i.e., the bounding box of the first branch; while the third example fails to get this second starting point outside B_1 , and needs to do a further search inside B_1 , which costs more time. The comparison of our algorithm with OCCT and ACIS shows that our algorithm is more efficient.

5.1.3. Extensive randomized experiments on our algorithm

We have generated 100,000 random ellipsoid pairs for intersection computation testing. The center coordinates of the randomly generated ellipsoids are numbers within the range of $[-0.5, 0.5]$. The axial directions of the ellipsoids are random numbers within the range of $[0, 1]$. The lengths of the major, intermediate and minor semi-axes of ellipsoids are random numbers within the range of $[\frac{1}{11}, 1]$.

After determining the topology type of each example through topology determination, we proceed to search for the required starting points. Our algorithm has successfully identified all necessary starting points in every test example with 100% consistent with the requirement. Furthermore, the average time of our algorithm for computing the intersection curve across all 100,000 examples is 5.5×10^{-4} second. We evaluate the efficiency of OCCT and ACIS on the same set of 100,000 random examples. The average time for OCCT is 2.5×10^{-2} second, while the average time for ACIS is 1.1×10^{-2} second. The results indicate that our algorithm is more efficient.

5.2. Discussion

5.2.1. Generality

The proposed algorithm uses the topology determination for the intersecting curve of quadrics in the projective space, allowing the determination of the topology types of the intersection curve between ellipsoids in affine space due to the bounded nature of ellipsoids. We propose a fast topology-preserving algorithm for computing the intersection curve of two ellipsoids.

Our algorithm can also be extended to the intersection problem of the ellipsoids and other quadrics. Table 7 present several examples of intersection cases for ellipsoid-cylinder and ellipsoid-cone, respectively. The table includes cases of no intersection, one intersection branch, and two intersection branches. We compare the performance of our algorithm with OCCT and ACIS. All three algorithms provide the correct intersection results. However, our algorithm is three orders of magnitude faster than OCCT and at least one order of magnitude faster than ACIS in the case of no intersection. In other cases, it outperforms both OCCT and ACIS by at least one order of magnitude.

5.2.2. Topology correctness

The main challenge in surface intersection lies in determining the topology of the intersection curve, particularly in special cases such as surface tangency, the presence of isolated points, small loops, and singular points on the intersection curve. To address these issues, we first determine the topology of the intersection curve using a set of discriminants, guaranteeing the topology correctness of the intersection curve.

Among all the examples in Section 5.1, our algorithm always presents correct topology results, while OCCT produces errors in some challenging cases. As shown in Table 5, when the intersection curve of two ellipsoids includes isolated points or when two intersecting branches are tangent, OCCT cannot ensure the correctness of results. For example, in the cases of Table 5 where two ellipsoids are internally tangent at two isolated points, OCCT incorrectly computes the intersection as empty. In contrast, our algorithm determines the topology of the intersection curve in advance through topological analysis, ensuring the correctness of the intersection topology as consisting of isolated points.

5.2.3. Efficiency

In existing intersection computation algorithms, the tracing method is very popularly used. The main time-consuming part of the tracing method is the complete computation of the starting points on each intersection branch. Our method quickly determines the intersection topology and obtains the number N of real intersection branches. This tells us how many starting points we need to compute, which saves computation time.

Table 5

The intersection computation time (in seconds) of two ellipsoids. ‘—’ represents the corresponding algorithm fails to give correct results.


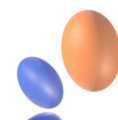



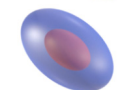

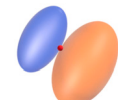

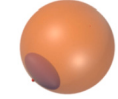

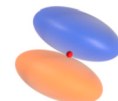

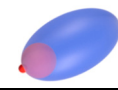

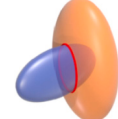









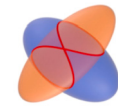

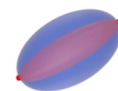

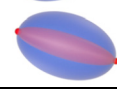
Intersection types	QSIC	Examples	Time (s)		
			OCCT	ACIS	Ours
No intersection			6.2×10^{-3}	5.9×10^{-5}	3.6×10^{-6}
			6.0×10^{-3}	6.2×10^{-5}	2.2×10^{-6}
			1.4×10^{-2}	4.0×10^{-4}	8×10^{-7}
One isolated point			3.1×10^{-2}	7.1×10^{-4}	4.8×10^{-6}
			—	6.7×10^{-3}	1.6×10^{-4}
			—	6.5×10^{-4}	5.3×10^{-6}
			—	5.1×10^{-3}	5.3×10^{-4}
One loop			2.2×10^{-2}	1.8×10^{-3}	3.4×10^{-4}
			1.4×10^{-1}	7.8×10^{-2}	7.7×10^{-4}
			1.9×10^{-2}	4.3×10^{-2}	3.8×10^{-4}
			—	6.8×10^{-2}	5.5×10^{-4}
			—	7.8×10^{-2}	6.2×10^{-4}
One 8-shaped branch			3.4×10^{-2}	1.7×10^{-1}	8.9×10^{-4}
Two isolated points			—	2.5×10^{-3}	2.3×10^{-4}
			—	2.2×10^{-3}	2.5×10^{-4}

Table 5 (continued)


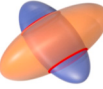



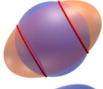

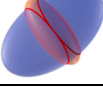



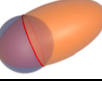

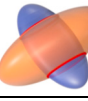

Intersection types	QSIC	Examples	Time (s)		
			OCCT	ACIS	Ours
Two loops			3.1×10^{-2}	5.0×10^{-3}	9.1×10^{-4}
			4.7×10^{-2}	1.7×10^{-2}	4.0×10^{-3}
			4.3×10^{-2}	7.5×10^{-3}	7.0×10^{-4}
			—	9.3×10^{-3}	1.1×10^{-3}
One isolated point and one loop			—	9.2×10^{-3}	6.8×10^{-4}
			—	6.4×10^{-3}	3.9×10^{-4}

Table 6

Time cost in examples reaching different levels of the algorithm.

Algorithm invocation	Examples	Time (s)		
		OCCT	ACIS	Ours
Calling Algorithm 2		2.2×10^{-2}	1.8×10^{-3}	3.4×10^{-4}
Calling Algorithm 2 and 4 and searching the second starting point in exterior of B_1		3.1×10^{-2}	5.0×10^{-3}	9.1×10^{-4}
Calling Algorithm 2 and 4 and searching the second starting point in interior of B_1		4.7×10^{-2}	1.7×10^{-2}	4.0×10^{-3}

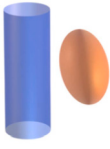

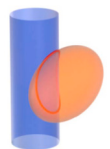
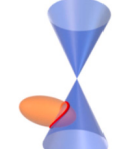
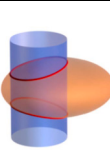

In Table 5 and Table 6, the implementation time of our algorithm consistently outperforms that of OCCT and ACIS in all these examples, *i.e.*, our algorithm is at least two orders of magnitude faster than OCCT and at least one order of magnitude faster than ACIS for most examples. Even in the most time-consuming case for our algorithm, as shown in the third example of Table 6, which requires all the algorithm steps outlined in Section 4, our algorithm still demonstrates superior efficiency compared to OCCT and ACIS.

5.2.4. Numerical stability

Since the algorithm ultimately runs in a floating point environment within CAD systems, numerical stability is a crucial requirement for intersection computation. Traditional algebraic methods for topological determination are typically only suitable for exact computations. In contrast, the topological determination method that we employ, which involves calculating a series of algebraic discriminants to identify the topological type of the intersection curve, remains applicable in a floating-point environment. We discuss the numerical stability in relation to two tolerances, *i.e.*, one algebraic tolerance associated with the sign determination of algebraic discriminants for topological classification, and one geometric tolerance related to the termination condition for the octree search in Remark 4.4.

Table 7

Examples for the computation of intersection between a cylinder and an ellipsoid, as well as the intersection between a cone and an ellipsoid.

Examples	Time (s)			Examples	Time (s)		
	OCCT	ACIS	Ours		OCCT	ACIS	Ours
	2.0×10^{-3}	4.2×10^{-5}	3.6×10^{-6}		3.0×10^{-3}	1.0×10^{-4}	2.3×10^{-6}
	4.4×10^{-3}	4.6×10^{-3}	3.6×10^{-4}		3.6×10^{-3}	5.1×10^{-3}	3.5×10^{-4}
	5.7×10^{-3}	7.4×10^{-3}	8.6×10^{-4}		6.5×10^{-3}	1.3×10^{-2}	9.7×10^{-4}

For the first tolerance, the key is to correctly determine the sign (zero, positive or negative) of the algebraic discriminant when the surface equation coefficients are floating-point numbers. This becomes particularly challenging when the discriminant value is close to zero. The explicit relationship between this algebraic tolerance and the geometric tolerance in three-dimensional space remains an open problem. In our algorithm, the strategy we adopt is to first normalize the leading coefficients of the input ellipsoid equations and set an algebraic tolerance $\epsilon_d = 10^{-8}$, treating the discriminant as zero when its absolute value is smaller than ϵ_d .

For the second tolerance, the geometric tolerance in Remark 4.4 is used to determine the intersection point of two surfaces within the final subdivided leaf nodes, and the size ϵ in Remark 4.4 can be selected based on the algorithm requirements. Since our algorithm has prior knowledge of the topology type and the number of intersection branches, it continues searching until all required starting points are found. As a result, once the intersection topology is determined, an improper setting of this geometric tolerance may impact the efficiency of the algorithm but does not affect its ability to identify all intersection branches. In this paper, the size of the box located in the root node is less than 5, and the maximum depth of 15 can meet the requirements with the selected tolerance $\epsilon = 2 \times 10^{-4}$.

6. Conclusion

We propose an efficient and topologically stable algorithm to compute the intersection of two ellipsoids. The basic idea is to first apply topology determination in order to quickly obtain the intersection topology and the number N of real intersection branches. If $N = 0$, the two ellipsoids are separate; if $N = 1$ or $N = 2$, we need only to compute one starting point on each of the N intersection branches. This strategy improves not only the topology stability but also the computational efficiency of the intersection algorithm. On the other hand, an open problem in surface intersection computation is how to determine whether your result is good or not. An error function on measuring the distances from the intersection points to the two surfaces is apparently not enough. Think about the case where your computation result computes one intersection branch in a precise way but has lost the other intersection branch. If you measure your result quality by the error function, you get a satisfied feedback; however, you have lost a whole intersection branch! Therefore, this is also an advantage of our approach: we tell at the very beginning the number of real intersection branches, and you have to find them all before you stop. A possible future problem is to design an algorithm to quickly decide the number of intersection branches between two NURBS. Once this is solved, the process of our algorithm also applies to the intersection computation of two NURBS.

CRedit authorship contribution statement

Xiao Chu: Conceptualization, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Kai Li:** Methodology, Visualization, Writing – original draft, Writing – review & editing, Validation. **Xiaohong Jia:** Conceptualization, Formal analysis, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing. **Jieyin Yang:** Methodology, Software, Validation. **Jiarui Kang:** Methodology, Software, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work is supported by National Key R&D Program of China (2024YFE0204400), Strategic Priority Research Program of the Chinese Academy of Sciences XDB0640000 & XDB0640200, Key Project of the National Natural Science Foundation of China (12494550 & 12494551), and China Postdoctoral Science Foundation (2024M763473).

Data availability

The authors are unable or have chosen not to specify which data has been used.

References

- Abdel-Malek, K., Yeh, H.J., 1996. Determining intersection curves between surfaces of two solids. *Comput. Aided Des.* 28, 539–549.
- Bajaj, C.L., Hoffmann, C.M., Lynch, R.E., Hopcroft, J., 1988. Tracing surface intersections. *Comput. Aided Geom. Des.* 5, 285–307.
- Barnhill, R.E., Kersey, S.N., 1990. A marching method for parametric surface/surface intersection. *Comput. Aided Geom. Des.* 7, 257–280.
- Choi, Y.K., Chang, J.W., Wang, W., Kim, M.S., Elber, G., 2008. Continuous collision detection for ellipsoids. *IEEE Trans. Vis. Comput. Graph.* 15, 311–325.
- Dokken, T., Skytt, V., Ytrehus, A.M., 1989. Recursive subdivision and iteration in intersections and related problems. In: *Mathematical Methods in Computer Aided Geometric Design*. Elsevier, pp. 207–214.
- Dupont, L., Lazard, D., Lazard, S., Petitjean, S., 2008a. Near-optimal parameterization of the intersection of quadrics: I. the generic algorithm. *J. Symb. Comput.* 43, 168–191.
- Dupont, L., Lazard, D., Lazard, S., Petitjean, S., 2008b. Near-optimal parameterization of the intersection of quadrics: ii. a classification of pencils. *J. Symb. Comput.* 43, 192–215.
- Dupont, L., Lazard, D., Lazard, S., Petitjean, S., 2008c. Near-optimal parameterization of the intersection of quadrics: iii. parameterizing singular intersections. *J. Symb. Comput.* 43, 216–232.
- Farouki, R.T., Neff, C., O'Conner, M., 1989. Automatic parsing of degenerate quadric-surface intersections. *ACM Trans. Graph.* 8, 174–203.
- Ghossein, E., Lévesque, M., 2013. Random generation of periodic hard ellipsoids based on molecular dynamics: a computationally-efficient algorithm. *J. Comput. Phys.* 253, 471–490.
- Grandine, T.A., Klein IV, F.W., 1997. A new approach to the surface intersection problem. *Comput. Aided Geom. Des.* 14, 111–134.
- Hass, J., Farouki, R.T., Chang, Y.H., Song, X., Sederberg, T.W., 2007. Guaranteed consistency of surface intersections and trimmed surfaces using a coupled topology resolution and domain decomposition scheme. *Adv. Comput. Math.* 27, 1–26.
- Jia, X., Choi, Y.K., Mourrain, B., Wang, W., 2011. An algebraic approach to continuous collision detection for ellipsoids. *Comput. Aided Geom. Des.* 28, 164–176.
- Jia, X., Tu, C., Mourrain, B., Wang, W., 2020. Complete classification and efficient determination of arrangements formed by two ellipsoids. *ACM Trans. Graph.* 39, 1–12.
- Kasik, D.J., Buxton, W., Ferguson, D.R., 2005. Ten CAD challenges. *IEEE Comput. Graph. Appl.* 25, 81–92.
- Krishnan, S., Manocha, D., 1997. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Trans. Graph.* 16, 74–106.
- Lee, B.H., Jeon, J.D., Oh, J.H., 2017. Velocity obstacle based local collision avoidance for a holonomic elliptic robot. *Auton. Robots* 41, 1347–1363.
- Levin, J., 1976. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Commun. ACM* 19, 555–563.
- Levin, J.Z., 1979. Mathematical models for determining the intersections of quadric surfaces. *Comput. Graph. Image Process.* 11, 73–87.
- Meagher, D., 1982. Geometric modeling using octree encoding. *Comput. Graph. Image Process.* 19, 129–147.
- Mortenson, M.E., 1997. *Geometric Modeling*. John Wiley & Sons, Inc.
- Park, Y., Son, S.H., Kim, M.S., Elber, G., 2020. Surface-surface-intersection computation using a bounding volume hierarchy with osculating toroidal patches in the leaf nodes. *Comput. Aided Des.* 127, 102866.
- Patrikalakis, N.M., 1993. Surface-to-surface intersections. *IEEE Comput. Graph. Appl.* 13, 89–95.
- Piegl, L.A., 2005. Ten challenges in computer-aided design. *Comput. Aided Des.* 37, 461–470.
- Rossignac, J.R., Requicha, A.A., 1987. Piecewise-circular curves for geometric modeling. *IBM J. Res. Dev.* 31, 296–313.
- Sarraga, R.F., 1983. Algebraic methods for intersections of quadric surfaces in GMSOLID. *Comput. Vis. Graph. Image Process.* 22, 222–238.
- SAS, O.C., 2023. *Open cascade technology*. <https://dev.opencascade.org/>.
- Shao, W., Chen, F., 2023. Topological classification of the intersection curves of two quadrics using a set of discriminants. *Comput. Aided Geom. Des.* 107, 102244.
- Shuai, L., Li, C., Guo, X., Prabhakaran, B., Chai, J., 2016. Motion capture with ellipsoidal skeleton using multiple depth cameras. *IEEE Trans. Vis. Comput. Graph.* 23, 1085–1098.
- Team, S., 2023. *3d acis modeler*. <https://www.spatial.com/products/3d-acis-modeling>.
- Tu, C., Wang, W., Mourrain, B., Wang, J., 2009. Using signature sequences to classify intersection curves of two quadrics. *Comput. Aided Geom. Des.* 26, 317–335.
- Ventura, M., Guedes Soares, C., 2012. Surface intersection in geometric modeling of ship hulls. *J. Mar. Sci. Technol.* 17, 114–124.
- Wang, H., 2014. Defending continuous collision detection against errors. *ACM Trans. Graph.* 33, 1–10.
- Wang, W., 2002. *Modeling and processing with quadric surfaces*.
- Wang, W., Goldman, R., Tu, C., 2003. Enhancing Levin's method for computing quadric-surface intersections. *Comput. Aided Geom. Des.* 20, 401–422.
- Wang, W., Joe, B., Goldman, R., 2002. Computing quadric surface intersections based on an analysis of plane cubic curves. *Graph. Models* 64, 335–367.
- Wang, W., Wang, J., Kim, M.S., 2001. An algebraic condition for the separation of two ellipsoids. *Comput. Aided Geom. Des.* 18, 531–539.
- Yang, L., 1999. Recent advances on determining the number of real roots of parametric polynomials. *J. Symb. Comput.* 28, 225–242.
- Ye, Y., Wang, Y., Cao, J., Chen, Z., 2024. Watertight surface reconstruction method for cad models based on optimal transport. *Comput. Vis. Media* 10, 859–872.