On Intersections of B-Spline Curves

Yu, Ying-Ying; Li, Xin; Ji, Ye

DOI
10.3390/math12091344

Publication date
2024

Document Version
Final published version

Published in
Mathematics

mathematics

MDPI

Article

On Intersections of B-Spline Curves

Ying-Ying Yu [1], Xin Li [2] and Ye Ji [3,*]

[1] School of Mathematics, Liaoning Normal University, Dalian 116029, China; yuyydl@lnnu.edu.cn
[2] School of Mathematical Sciences, Dalian University of Technology, Dalian 116024, China; lixin031@mail.dlut.edu.cn
[3] Delft Institute of Applied Mathematics, Delft University of Technology, 2628 CD Delft, The Netherlands
* Correspondence: y.ji-1@tudelft.nl

Abstract: Bézier and B-spline curves are foundational tools for curve representation in computer graphics and computer-aided geometric design, with their intersection computation presenting a fundamental challenge in geometric modeling. This study introduces an innovative algorithm that quickly and effectively resolves intersections between Bézier and B-spline curves. The number of intersections between the two input curves within a specified region is initially determined by applying the resultant of a polynomial system and Sturm's theorem. Subsequently, the potential region of the intersection is established through the utilization of the pseudo-curvature-based subdivision scheme and the bounding box detection technique. The projected Gauss-Newton method is ultimately employed to efficiently converge to the intersection. The robustness and efficiency of the proposed algorithm are demonstrated through numerical experiments, demonstrating a speedup of 3 to 150 times over traditional methods.

Keywords: geometric modeling; Bézier curves; B-spline curves; intersection

MSC: 65D17

Citation: Yu, Y.-Y.; Li, X.; Ji, Y. On Intersections of B-Spline Curves. *Mathematics* **2024**, *12*, 1344. https://doi.org/10.3390/math12091344

Academic Editor: Jay Jahangiri

Received: 28 March 2024
Revised: 19 April 2024
Accepted: 25 April 2024
Published: 28 April 2024

1. Introduction

Geometric modeling is a computer-based technology for the representation, manipulation, analysis, and design of solids [1]. Curves/surfaces modeling technology, which originated from shaping the outer appearance of products such as automobiles, ship fuselages, aircraft wings, and other products, is a significant research area in computer-aided design (CAD) and computer-aided geometric design (CAGD). Its applications span a broad spectrum of industrial engineering fields, encompassing CNC tooling, alignment of roads, robot path planning, design of luggage shells, and clearance detection for sheet metal parts, among others [2–5].

Hoffmann pointed out that computing the intersections of parametric curves and surfaces represents a foundational challenge within the field of CAGD and geometric modeling [6]. The resolution of intersections between curves and surfaces is critical in applications such as Boolean operations on solids [7], surface rendering via ray tracing [8,9], and collision detection [10]. In the process of geometric modeling, intersections play a significant role in determining the shape and structure of the model, so it is important to accurately identify and represent these features to ensure the model is correctly interpreted and manufactured [11,12]. As CAD/CAGD/CAM systems evolve and scientific and technological advancements continue, the computational demands and data volumes required to tackle intersection challenges are rapidly escalating. Therefore, the development of efficient and robust methods to address these challenges has become crucial.

Bézier curves defined by Bernstein basis functions and control points have a simple and intuitive mathematical expression and are easy to calculate [1]. B-spline curves, as an extension of Bézier curves, are a curve representation method renowned for their excellent

local control capacity and smoothness. The flexibility to adjust the shape of B-spline curves through manipulation of knot vectors and control points has made them popular in computer graphics for curve and surface modeling, animation, interpolation and shape editing, and font design. With their different characteristics and advantages, Bézier curves and B-spline curves become important tools for representing and manipulating curves in computer graphics, and many scholars have conducted research on the intersection problems of Bézier [13,14] and B-spline curves/surfaces [15–17] .

In fact, the problem of curve/curve intersection is somewhat equivalent to finding the roots of polynomials, leading some researchers to tackle these issues concurrently. The Bézier clipping algorithm was first proposed by Sederberg to robustly and quickly calculate not only the intersection points but also the tangent points of two Bézier curves [18]. The convergence rate of the Bézier clipping algorithm was proved to be second-order in [19]. Since then, various improved algorithms based on the Bézier clipping algorithm have been developed to address both polynomial root finding and curve intersection challenges. Based on degree reduction, the quadratic and cubic polynomials were generated to enclose the graph of the polynomial within the interval of interest, respectively, by Bartoň and Jüttler [20] and Liu [21] et al. for computing the all roots of a univariate polynomial equation. Additionally, the cubic clipping was proved to have at least a second-order convergence rate and used to compute the intersections of two Bézier curves [22]. A geometric interval algorithm that can tightly bind a curve/surface or contain a point on a curve/surface was proposed by North [23]. Further advancements were made by Yuan [24], who developed the cubic hybrid clipping method with a fourth-order convergence rate for root finding of univariate polynomial equations. This method was subsequently adapted by Wu and Li to address curve intersection problems [25].

This study focuses on resolving the intricate challenge of identifying all intersection points between Bézier curves and B-spline curves. We introduce a robust and efficient approach for determining the intersections for both Bézier curves and B-spline curves. First, a robust algorithm for determining the number of intersections between two Bézier curves is proposed. The core of this problem lies in the task of determining the roots of a system of bivariate polynomial equations. To this end, the resultants of bivariate polynomials and Sturm's sequence are employed. Second, a fast computation algorithm based on pseudo-curvature and subdivision is proposed in this paper. While Newton's iteration is a well-known technique for solving non-linear equations, it is known to encounter difficulties, settling on local solutions in scenarios where multiple solutions exist. To address this, we subdivide the input curves by considering the pseudo-curvature of the curves and the intersection of their bounding boxes. Once segments are deemed to be sufficiently straight, we ensure that any two intersect at most once. Eventually, the Gauss-Newton method is employed to quickly converge on the intersection points. Numerical experiments validate the superior performance of our approach compared to traditional algorithms, achieving a speedup of 3 to 150 times.

The remainder of this paper is structured as follows: Section 2 presents essential definitions of Bézier and B-spline curves, along with Sturm's theorem. In Section 3, we outline a method for computing the number of intersections between polynomial parametric curves, focusing primarily on Bézier curves within the unit square. Section 4 introduces our proposed subdivision-based method tailored for efficiently computing all intersections between B-spline curves. Section 5 presents numerical experiments assessing the performance of the proposed method. Finally, Section 6 offers concluding remarks summarizing our contributions and suggesting potential avenues for future research.

2. Preliminary

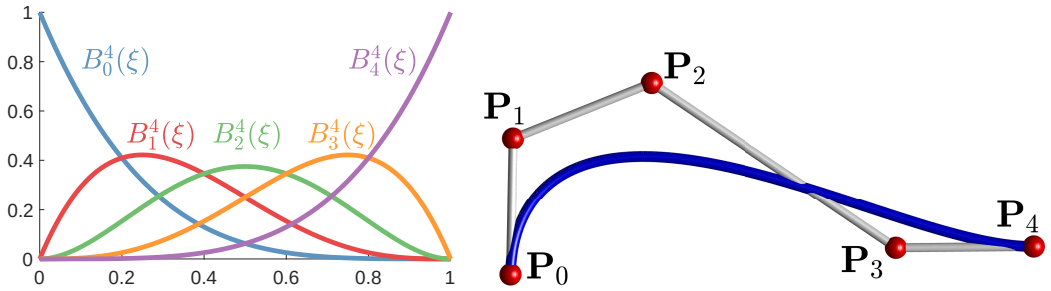
In this section, we commence with a brief overview of the concepts underlying Bézier and B-spline curves. Subsequently, we introduce Sturm's theorem, an essential tool for isolating the real roots of higher-order polynomial equations.

2.1. Bézier and B-Spline Curves

A degree-n Bézier curve is expressed as

$$\mathcal{C}(\xi) = \sum_{i=0}^{n} \boldsymbol{P}_i B_i^n(\xi), \quad \xi \in [0,1], \tag{1}$$

where $\boldsymbol{P}_i \in \mathbb{R}^d$ ($d = 2, 3$ typically) denote the control points, and $B_i^n(\xi) = \binom{n}{i}\xi^i(1-\xi)^{n-i}$ ($i = 0, 1, \cdots, n$) are Bernstein basis functions [1]. These curves are characterized by their high degree of smoothness, ease of computation, and the property that they are contained within the convex hull of their control points. Figure 1 illustrates quartic Bernstein basis functions and its associated Bézier curve.



(**a**) Quartic Bernstein basis functions (**b**) Quartic Bézier curve

Figure 1. Quartic Bernstein basis functions and Bézier Curve.

Building on the concept of Bézier curves, B-spline curves offer increased flexibility and local modification capability through the introduction of a knot vector. The knot vector, a non-decreasing sequence denoted as:

$$\Xi = \{\xi_0, \xi_1, \ldots, \xi_{n+p+1}\}, \tag{2}$$

where $\xi_i \leq \xi_{i+1}$ for $i = 1, 2, \ldots, n+p$ serves as the basis for defining B-spline curves [1].

The B-spline basis functions defined over Ξ can be derived recursively by the following de Boor-Cox formula

$$\begin{cases} N_{i,0}(\xi) = \begin{cases} 1, & \xi \in [\xi_i, \xi_{i+1}), \\ 0, & \text{otherwise}, \end{cases} \\ N_{i,p}(\xi) = \dfrac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \dfrac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi), \quad p \geq 1. \end{cases} \tag{3}$$

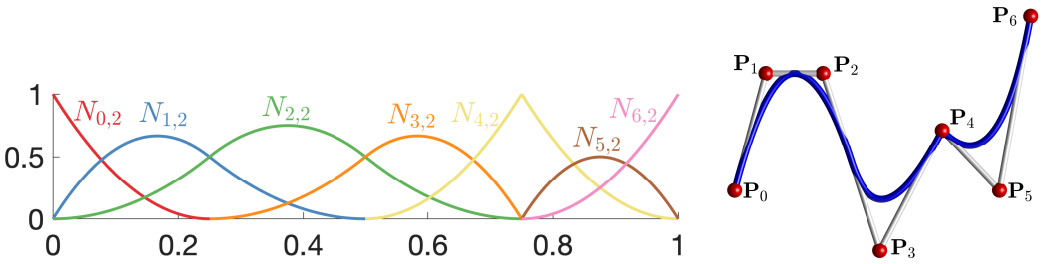
Consequently, a B-spline curve of degree p is formulated as the linear combination of its basis functions, $N_{i,p}(\xi)$, for $i = 1, 2, \ldots, n$, and the corresponding control points, $\boldsymbol{P}_i$, where $\boldsymbol{P}_i \in \mathbb{R}^d$ with $d = 2, 3$. Thus, a B-spline curve can be expressed as:

$$\mathcal{C}(\xi) = \sum_{i=1}^{n} \boldsymbol{P}_i N_{i,p}(\xi), \quad \xi \in [\xi_p, \xi_{n+1}]. \tag{4}$$

Figure 2 shows quadratic B-spline basis functions defined over the knot vector $\Xi = \{0, 0, 0, 0.25, 0.5, 0.75, 0.75, 1, 1, 1\}$ and its associated B-spline curve.

B-spline basis functions are endowed with essential desirable properties such as non-negativity and partition of unity, which contribute significantly to the stability and manipulability of the curves they define. Therefore, B-spline curves, closely related to these basis functions, exhibit the so-called convex-hull property, ensuring they remain within the convex hull defined by their control points. These features contribute to their widespread application in CAGD and CG for their stability and ease of manipulation. Furthermore, a degree-p Bézier curve can be considered a specific form of a B-spline curve, characterized

by a particular knot vector $\Xi = \{\underbrace{0,0,\ldots,0}_{p+1},\underbrace{1,1,\ldots,1}_{p+1}\}$. This connection means that Bézier curves inherit many of the advantageous properties of B-spline curves, enhancing their utility and desirability in various applications.



(**a**) Quadratic B-spline basis functions (**b**) Quadratic B-spline curve

Figure 2. Quadratic B-spline basis functions and curve defined over knot vector $\Xi = \{0,0,0,0.25,0.5,0.75,0.75,1,1,1\}$.

2.2. Resultant and Sturm's Theorem

To lay a solid theoretical foundation for the ensuing discussion, this subsection introduces the basic concept of the resultant of two polynomials, the Sturm sequence associated with a polynomial, and Sturm's theorem.

Definition 1 ([26]). *Let $f_1(x) = \sum_{i=0}^{m} a_i x^i$ and $f_2(x) = \sum_{i=0}^{n} b_i x^i$ represent two polynomials of degree m and n, respectively. The resultant of f_1 and f_2, $Res(f_1, f_2)$, is defined by a determinant that encapsulates the coefficients of these polynomials, i.e.,*

$$Res(f_1,f_2) = \begin{vmatrix} a_m & a_{m-1} & \cdots & a_1 & a_0 & 0 & \cdots & 0 & row\ 1 \\ 0 & a_m & a_{m-1} & \cdots & a_1 & a_0 & \cdots & 0 & row\ 2 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & a_m & a_{m-1} & \cdots & \cdots & a_1 & a_0 & row\ n \\ b_n & b_{n-1} & \cdots & b_1 & b_0 & 0 & \cdots & 0 & row\ n+1 \\ 0 & b_n & b_{n-1} & \cdots & b_1 & b_0 & \cdots & 0 & row\ n+2 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & b_n & b_{n-1} & \cdots & \cdots & b_1 & b_0 & row\ n+m \end{vmatrix}. \tag{5}$$

On a natural extension of Definition 1, when considering the resultant of two bivariate polynomials concerning one of the variables, the other variable is treated as a constant. A simple example is given to illustrate below.

Example 1. *Given two bivariate polynomials $f_1(x,y) = 5x^2 - 6xy + 5y^2 - 16$ and $f_2(x,y) = 2x^2 - (1+y)x + y^2 - y - 4$. The resultant defined by f_1, f_2 with respect to variable x is*

$$Res(f_1,f_2;x) = \begin{vmatrix} 5 & -6y & 5y^2-16 & 0 \\ 0 & 5 & -6y & 5y^2-16 \\ 2 & -(1+y) & y^2-y-4 & 0 \\ 0 & 2 & -(1+y) & y^2-y-4 \end{vmatrix},$$

which is essentially a polynomial of the variable y.

Consider a polynomial $f(x)$ with real coefficients and free of any square factors. The Sturm sequence for $f(x)$ is constructed as follows

$$\begin{aligned} \mathbf{Sturm}^{(0)} &= f(x) \\ \mathbf{Sturm}^{(1)} &= f'(x) \\ \mathbf{Sturm}^{(\mathrm{i})} &= -\mathbf{rem}\left(\mathbf{Sturm}^{(\mathrm{i}-2)}, \mathbf{Sturm}^{(\mathrm{i}-1)}\right), \quad \text{for i} = 2, 3, \ldots, \mathrm{r}, \end{aligned} \tag{6}$$

where **rem** denotes the polynomial remainder operator. In this sequence, each successive term is the negated remainder from dividing the two preceding terms. The construction of this sequence concludes when a constant term or zero is reached at $\mathbf{Sturm}^{(r)}$. The Sturm sequence, denoted as $\mathbf{Sturm} = \{\mathbf{Sturm}^{(0)}, \mathbf{Sturm}^{(1)}, \ldots, \mathbf{Sturm}^{(r)}\}$, serves as an effective tool for identifying the count of zeros that the polynomial $f(x)$ possesses within a given interval.

Consider the sequence $\mathbf{\Gamma} = \{\gamma_0, \gamma_1, \ldots, \gamma_t\}$. Denote by $pre(\mathbf{\Gamma})$ the number of sign-preserving occurrences within $\mathbf{\Gamma}$. This is quantified by examining each pair of consecutive values, γ_i and γ_{i+1}. If these values share the same sign, the pair (γ_i, γ_{i+1}) is counted as contributing 1 to $pre(\mathbf{\Gamma})$. Notably, a term $\gamma_i = 0$ is treated as though it possesses the opposite sign of its immediate predecessor γ_{i-1} [27,28].

Example 2. *Suppose* $\mathbf{\Gamma}_1 = [1, 2, 1, 2, -1, -2]$ *and* $\mathbf{\Gamma}_2 = [1, 0, 1, 2, -1, -2]$. *Then, we have the numbers of sign-preserving occurrences* $pre(\mathbf{\Gamma}_1) = 4$ *and* $pre(\mathbf{\Gamma}_2) = 2$.

Integrating the definition of the resultant for two bivariate polynomials with the concept of sign-preserving within a Sturm sequence of a specified polynomial, Sturm's theorem emerges as a powerful method to ascertain the count of real roots for a bivariate polynomial system within a defined interval.

Theorem 1 ([29]). *Consider a compact set* $\mathbb{D} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \subset \mathbb{R}^2$. *Let* $f_1(x, y)$ *and* $f_2(x, y)$ *be two bivariate polynomials with the degrees of* x *as* n_1 *and* n_2, *respectively. Define a polynomial in terms of* α *by*

$$RES(\alpha, \alpha_1, \alpha_2) = \frac{Res(Res(f_1, g; x), Res(f_2, g; x); y)}{\alpha^n}, \tag{7}$$

where $g(x, y) = \alpha + (x - \alpha_1)(y - \alpha_2)$ *and* $n = n_1 \times n_2$.

Let $\{\mathbf{Sturm}(\alpha, \alpha_1, \alpha_2)\}$ *stand for the Sturm sequence of the polynomial* $RES(\alpha, \alpha_1, \alpha_2)$ *of* α. *Here,* $Num\_pre(a, b)$ *represents the count of sign-preserving occurrences in the Sturm sequence* $\{\mathbf{Sturm}(\alpha, \alpha_1, \alpha_2)\}$ *when* $\alpha = 0$, $\alpha_1 = a$, *and* $\alpha_2 = b$. *Consequently, the total number of real roots for the system of equations* $f_1 = f_2 = 0$ *within the compact set* $\mathbb{D}$ *can be determined by*

$$\begin{aligned} Num(f_1, f_2, \mathbb{D}) = \frac{1}{2}[&Num\_pre(x_{min}, y_{min}) - Num\_pre(x_{min}, y_{max}) \\ &- Num\_pre(x_{max}, y_{min}) + Num\_pre(x_{max}, y_{max})]. \end{aligned} \tag{8}$$

Remark 1. *In cases where the first term in a sequence is* 0, *it is stated that* $pre([0, 1]) = pre([0, -1]) = 1/2$, *as detailed in [29]. This convention implies that roots located on the boundaries of a specified compact set* $\mathbb{D}$ *are represented by* $1/2$ *in Equation (8). Similarly, roots situated at the vertices of the compact set* $\mathbb{D}$ *are attributed a value of* $1/4$.

Theorem 1 provides a theoretical foundation for determining the exact number of roots for a pair of bivariate polynomial equations $f_1(x, y) = f_2(x, y) = 0$ within a specified compact set $\mathbb{D}$. In the subsequent discussion, we will apply Theorem 1 to ascertain the number of intersections between two polynomial parametric curves. This step can be

regarded as a preparatory phase, essential for enhancing the precision and reliability of the subsequent calculation of intersections.

3. Determining the Number of Intersections

In this section, we provide a method for counting the number of intersections between two polynomial parametric curves, with an emphasis on Bézier curves within the interval $[0,1] \times [0,1]$ for clarity and brevity.

Given two Bézier curves

$$\boldsymbol{\mathcal{C}}_A(\xi) = \sum_{i=0}^{n} \boldsymbol{P}_i B_i^n(\xi)$$

and

$$\boldsymbol{\mathcal{C}}_B(\eta) = \sum_{j=0}^{m} \boldsymbol{Q}_j B_j^m(\eta),$$

where $\boldsymbol{P}_i = (x_i, y_i, z_i)^{\mathrm{T}}$, $\boldsymbol{Q}_j = (\widetilde{x}_j, \widetilde{y}_j, \widetilde{z}_j)^{\mathrm{T}}$ are their control points. Computing the intersection of $\boldsymbol{\mathcal{C}}_A(\xi)$ and $\boldsymbol{\mathcal{C}}_B(\eta)$ involves identifying the parameter pair (ξ, η) that fulfills the condition $\boldsymbol{\mathcal{C}}_A(\xi) - \boldsymbol{\mathcal{C}}_B(\eta) = \mathbf{0}$.

From the partition of the unity property of Bernstein basis functions, we have

$$\begin{aligned} \boldsymbol{\mathcal{C}}_A(\xi) - \boldsymbol{\mathcal{C}}_B(\eta) &= \sum_{i=0}^{n} \boldsymbol{P}_i B_i^n(\xi) - \sum_{j=0}^{m} \boldsymbol{Q}_j B_j^m(\eta) \\ &= \sum_{j=0}^{m} B_j^m(\eta) \cdot \sum_{i=0}^{n} \boldsymbol{P}_i B_i^n(\xi) - \sum_{i=0}^{n} B_i^n(\xi) \cdot \sum_{j=0}^{m} \boldsymbol{Q}_j B_j^m(\eta) \\ &= \sum_{i=0}^{n} \sum_{j=0}^{m} (\boldsymbol{P}_i - \boldsymbol{Q}_j) B_i^n(\xi) B_j^m(\eta). \end{aligned} \tag{9}$$

To simplify the notation and facilitate computation, let us denote

$$\mathbf{B}^n(\xi) = (B_0^n(\xi), B_1^n(\xi), \ldots, B_n^n(\xi))^{\mathrm{T}}, \tag{10}$$

$$\mathbf{B}^m(\eta) = (B_0^m(\eta), B_1^m(\eta), \ldots, B_m^m(\eta))^{\mathrm{T}}, \tag{11}$$

$$\Delta_i = (\boldsymbol{P}_i - \boldsymbol{Q}_0, \boldsymbol{P}_i - \boldsymbol{Q}_1, \ldots, \boldsymbol{P}_i - \boldsymbol{Q}_m)^{\mathrm{T}}, \tag{12}$$

and

$$\mathbf{R} = (\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_n)^{\mathrm{T}}, \tag{13}$$

where $\mathbf{R}_i = \sum_{j=0}^{m} (\boldsymbol{P}_i - \boldsymbol{Q}_j) B_j^m(\eta) = \Delta_i^{\mathrm{T}} \cdot \mathbf{B}^m(\eta)$.

Therefore, Equation (9) can be expressed in a matrix form as

$$\boldsymbol{\mathcal{C}}_A(\xi) - \boldsymbol{\mathcal{C}}_B(\eta) = \mathbf{R}^{\mathrm{T}} \cdot \mathbf{B}^n(\xi). \tag{14}$$

Subsequently, according to the relationship between the Bernstein basis functions and power basis functions, we have

$$\mathbf{B}^n(\xi) = \boldsymbol{M} \cdot \boldsymbol{\xi},$$

where $\boldsymbol{M}$ is the basis transformation matrix and $\boldsymbol{\xi} = (1, \xi, \xi^2, \ldots, \xi^n)^{\mathrm{T}}$ is the collection of power basis functions. Therefore, we have

$$\boldsymbol{\mathcal{C}}_A(\xi) - \boldsymbol{\mathcal{C}}_B(\eta) = \begin{pmatrix} x(\xi, \eta) \\ y(\xi, \eta) \\ z(\xi, \eta) \end{pmatrix} = \mathbf{R}^{\mathrm{T}} \cdot \boldsymbol{M} \cdot \boldsymbol{\xi}. \tag{15}$$

Recall that computing the intersections between $\boldsymbol{\mathcal{C}}_A(\xi)$ and $\boldsymbol{\mathcal{C}}_B(\eta)$ is equivalent to solving the roots of $\boldsymbol{\mathcal{C}}_A(\xi) - \boldsymbol{\mathcal{C}}_B(\eta) = \mathbf{0}$. That is,

$$\begin{cases} x(\xi,\eta) = 0 \\ y(\xi,\eta) = 0 \\ z(\xi,\eta) = 0 \end{cases}. \tag{16}$$

Given that Theorem 1 is applicable solely for isolating the real roots of bivariate polynomials, Equation (16) can be equivalently converted into the following form

$$\begin{cases} f_1(\xi,\eta) = x^2(\xi,\eta) + y^2(\xi,\eta) = 0 \\ f_2(\xi,\eta) = y^2(\xi,\eta) + z^2(\xi,\eta) = 0 \end{cases}. \tag{17}$$

Denote by Δ_i^1 and Δ_i^2 the first two lines and the last two lines of Δ_i, respectively. Similarly, we have

$$\mathbf{R}^{\varphi} = \left(\mathbf{R}_0^{\varphi}, \mathbf{R}_1^{\varphi}, \ldots, \mathbf{R}_n^{\varphi}\right)^{\mathrm{T}}, \tag{18}$$

where $\mathbf{R}_i^{\varphi} = \left(\Delta_i^{\varphi}\right)^{\mathrm{T}} \cdot \mathbf{B}^m(\eta)$ with $\varphi = 1, 2$. Accordingly, the Equation (17) to compute the intersection of $\boldsymbol{\mathcal{C}}_A(\xi)$ and $\boldsymbol{\mathcal{C}}_B(\eta)$ can thus be reformulated as follows

$$\begin{cases} f_1(\xi,\eta) = \left(\mathbf{R}^1 \boldsymbol{M}\boldsymbol{\xi}\right)^{\mathrm{T}} * \left(\mathbf{R}^1 \boldsymbol{M}\boldsymbol{\xi}\right) = \boldsymbol{\xi}^{\mathrm{T}}\left(\boldsymbol{M}^{\mathrm{T}}(\mathbf{R}^1)^{\mathrm{T}}\mathbf{R}^1\boldsymbol{M}\right)\boldsymbol{\xi}, \\ f_2(\xi,\eta) = \left(\mathbf{R}^2 \boldsymbol{M}\boldsymbol{\xi}\right)^{\mathrm{T}} * \left(\mathbf{R}^2 \boldsymbol{M}\boldsymbol{\xi}\right) = \boldsymbol{\xi}^{\mathrm{T}}\left(\boldsymbol{M}^{\mathrm{T}}(\mathbf{R}^2)^{\mathrm{T}}\mathbf{R}^2\boldsymbol{M}\right)\boldsymbol{\xi}. \end{cases} \tag{19}$$

Remark 2. *Referring to Equation (7) from Theorem 1, the computation of the polynomial $RES(\alpha, \alpha_1, \alpha_2)$ necessitates initially deriving the resultant of the polynomials f_i ($i = 1, 2$) and g with respect to the variable ξ, treating η as a constant during this process. Consequently, we reformulate Equation (17) into a quadratic equation in terms of ξ.*

Remark 3. *In the case of plane polynomial parametric curves, the approach simplifies significantly. We can directly set $f_1(\xi,\eta) = x(\xi,\eta)$ and $f_2(\xi,\eta) = y(\xi,\eta)$, thereby streamlining the process.*

For clearer understanding, we present an algorithm that determines the number of intersections between two Bézier curves based on Theorem 1, followed by an illustrative example in Example 3. Moreover, the proposed algorithm may be naturally applicable to B-spline curves by combining with the Bézier extraction technique (Algorithm 1).

Algorithm 1: Number of intersections between two Bézier curves

Input: Control points $\{\boldsymbol{P}_i\}_{i=0}^n$ and $\{\boldsymbol{Q}_j\}_{j=0}^m$.

Output: The number of intersections of two Bézier curves $Num(f_1, f_2, \mathbb{D})$.

```
1  𝔻 ← [0,1] × [0,1];
2  Compute f1(ξ,η) and f2(ξ,η);                          // Equation (19)
3  for 𝔻_α ← (α1,α2)(α1,α2 = 0,1) do
4      g(ξ,η) ← α + (ξ − α1)(η − α2);
5      Compute Res(f1,g;ξ) and Res(f2,g;ξ);               // Example 1
6      Compute RES(α,α1,α2);                              // Equation (7)
7      {Sturm(α,α1,α2)} ← the Sturm sequence of RES(α,α1,α2);
8      Num_pre(𝔻_α) ← pre([Sturm(0,α1,α2)]);
9  end
10 Compute Num(f1,f2,𝔻);                                 // Equation (8)
```

Example 3. *Let $\boldsymbol{P}_0 = (0,0)$, $\boldsymbol{P}_1 = (1,1)$, $\boldsymbol{P}_2 = (2,0)$ and $\boldsymbol{Q}_0 = (0.5,0)$, $\boldsymbol{Q}_1 = (1,1)$, $\boldsymbol{Q}_2 = (0,2)$ be the control points of two quadratic Bézier curves $\boldsymbol{\mathcal{C}}_A(\xi)$ and $\boldsymbol{\mathcal{C}}_B(\eta)$, as shown in*

Figure 3, respectively. Thus computing the intersections between $\boldsymbol{\mathcal{C}}_A(\xi)$ *and* $\boldsymbol{\mathcal{C}}_B(\eta)$ *is equivalent to the roots of the following bivariate polynomials*

$$\begin{cases} f_1(\xi,\eta) = 2\xi + \frac{3}{2}\eta^2 - \eta - \frac{1}{2}, \\ f_2(\xi,\eta) = -2\xi^2 + 2\xi - 2\eta \end{cases}$$

Now, we propose the detailed procedure for determining the roots of $f_1(\xi,\eta) = f_2(\xi,\eta) = 0$ *in* $[0,1]\times[0,1]$ *by Theorem 1.*

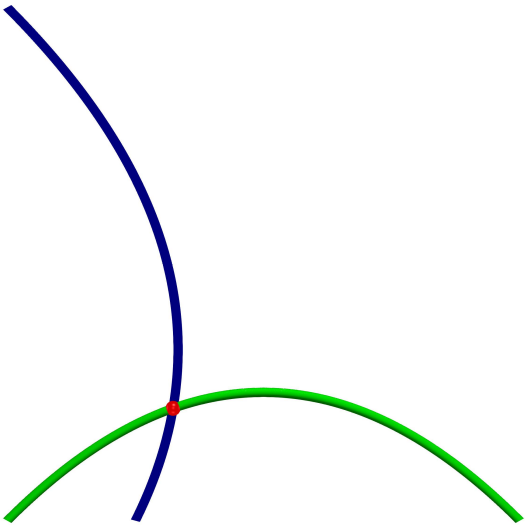


Figure 3. Two quadratic Bézier curves $\boldsymbol{\mathcal{C}}_A(\xi)$ and $\boldsymbol{\mathcal{C}}_B(\eta)$ in Example 3.

First, compute $Num\_pre(0,0)$, $Num\_pre(0,1)$, $Num\_pre(1,0)$, *and* $Num\_pre(1,1)$ *in Theorem 1, respectively. A comprehensive description of the process for calculating* $Num\_pre(0,0)$ *is shown below, which can also be applied to appraising* $Num\_pre(0,1)$, $Num\_pre(1,0)$, *and* $Num\_pre(1,1)$.

1. *Let* $\alpha_1 = 0, \alpha_2 = 0$, *then* $g(\xi,\eta) = \alpha + \xi\eta = \eta\xi + \alpha$. *Similar to Example 1, we can state that the resultant* $Res(f_1, g; \xi)$ *and* $Res(f_2, g; \xi)$ *in Equation (7) as*

$$Res(f_1, g; \xi) = \begin{vmatrix} 2 & \frac{3}{2}\eta^2 - \eta - \frac{1}{2} \\ \eta & \alpha \end{vmatrix} = -\frac{3}{2}\eta^3 + \eta^2 + \frac{1}{2}\eta + 2\alpha, \tag{20}$$

and

$$Res(f_2, g; \xi) = \begin{vmatrix} -2 & 2 & -2\eta \\ \eta & \alpha & 0 \\ 0 & \eta & \alpha \end{vmatrix} = -2\eta^3 - 2\alpha\eta - 2\alpha^2. \tag{21}$$

The outer resultant in Equation (7) is computed by

$$\begin{aligned} &Res(Res(f_1, g; \xi), Res(f_2, g; \xi); \eta) \\ &= \begin{vmatrix} 3/2 & 1 & 1/2 & 2\alpha & 0 & 0 \\ 0 & 3/2 & 1 & 1/2 & 2\alpha & 0 \\ 0 & 0 & 3/2 & 1 & 1/2 & 2\alpha \\ -2 & 0 & -2\alpha & -2\alpha^2 & 0 & 0 \\ 0 & -2 & 0 & -2\alpha & -2\alpha^2 & 0 \\ 0 & 0 & -2 & 0 & -2\alpha & -2\alpha^2 \end{vmatrix} \\ &= 27\alpha^6 + 126\alpha^5 + 173\alpha^4 + 54\alpha^3 + 3\alpha^2, \end{aligned} \tag{22}$$

and

$$\begin{aligned} RES(\alpha,0,0) &= \frac{Res(Res(f_1,g;\xi),Res(f_2,g;\xi);\eta)}{\alpha^{1\times 2}} \\ &= 27\alpha^4+126\alpha^3+173\alpha^2+54\alpha+3. \end{aligned} \tag{23}$$

2. *The Sturm sequence* $\{\mathbf{Sturm}(\alpha,0,0)\}$ *of* $RES(\alpha,0,0)$ *can be obtained by the method of successive division. That is*

$$\begin{aligned} \mathbf{Sturm}^{(0)} &= -0.017341-0.312139\alpha-\alpha^2-0.728324\alpha^3-0.156069\alpha^4, \\ \mathbf{Sturm}^{(1)} &= -0.142857-0.915344\alpha-\alpha^2-0.285714\alpha^3, \\ \mathbf{Sturm}^{(2)} &= -0.211034-\alpha-0.393103\alpha^2, \\ \mathbf{Sturm}^{(3)} &= -0.056683+\alpha, \\ \mathbf{Sturm}^{(4)} &= 1. \end{aligned} \tag{24}$$

3. *By substituting* $\alpha=0$ *into Equation* (24)*, we obtain the sequence* $\mathbf{\Gamma}=[-0.017341,-0.142857,-0.211034,-0.056683,1]$*. It is observed that* $Num\_pre(0,0)=3$*, corresponding to the sign-preserving of* $\mathbf{\Gamma}$*.*

Analogously, we find that $Num\_pre(0,1)=2$*,* $Num\_pre(1,0)=2$*, and* $Num\_pre(1,1)=3$*, respectively. Consequently, according to Equation* (8)*, it is concluded that the Bézier curves* $\boldsymbol{P}(\xi)$ *and* $\boldsymbol{Q}(\eta)$ *intersect exactly once.*

4. A Fast Subdivision-Based Intersection Algorithm

In the realm of geometric modeling and computer graphics, accurately determining the intersections between two parametric curves is a fundamental challenge with numerous applications. This task is particularly demanding due to the need for computational efficiency and high precision. In this section, we propose a novel subdivision-based approach specifically designed to compute all intersections between B-spline curves efficiently.

4.1. Pseudo-Curvature of B-Spline Curves

Central to our intersection algorithm lies an effective subdivision scheme, capitalizing on the curvature characteristics of B-spline curves for more efficient computation. This scheme introduces an effective curvature-based subdivision strategy, boosting computational efficiency and accuracy when identifying intersection points.

Generally, the point-wise curvature at any given parameter ξ along a parametric curve is determined by

$$\kappa_{\mathrm{p}}(\boldsymbol{\mathcal{C}}(\xi))=\frac{\left|\boldsymbol{\mathcal{C}}'(\xi)\times\boldsymbol{\mathcal{C}}''(\xi)\right|}{\left|\boldsymbol{\mathcal{C}}'(\xi)\right|^3}. \tag{25}$$

To ascertain the total curvature energy across the curve, one integrates the point-wise curvature as follows:

$$\kappa_{\text{tradition}}=\int \kappa_{\mathrm{p}}(\boldsymbol{\mathcal{C}}(\xi))\,\mathrm{d}\xi. \tag{26}$$

In practice, the above integration is typically executed using numerical quadrature techniques, such as Simpson's rule or Gauss-Legendre quadrature.

Although the above traditional methods for calculating the curvature of parametric curves are accurate, they involve intricate calculations that can be computationally expensive. To address this, we introduce the concept of pseudo-curvature, with a particular focus on B-spline curves. This approach seeks to offer a less computationally intensive alternative while maintaining sufficient accuracy for intersection determination tasks.

For a B-spline curve represented as $\boldsymbol{\mathcal{C}}(\xi)=\sum_{i=0}^{n}\mathbf{P}_iN_i(\xi)$, we define its pseudo-curvature as:

$$\kappa(\boldsymbol{\mathcal{C}})=\sum_{i=0}^{n-1}\frac{\left\|\overline{\mathbf{P}_i\mathbf{P}_{i+1}}\right\|_2}{\left\|\overline{\mathbf{P}_0\mathbf{P}_n}\right\|_2}, \tag{27}$$

where $\|\cdot\|$ stands for the Euclidean length of the vector. This formula effectively approximates the bending of the curve, as shown in Figure 4, by considering the relative lengths of segments formed by consecutive control points $\overline{\mathbf{P}_i\mathbf{P}_{i+1}}$ $(i = 0, 1, \ldots, n-1)$ in comparison to the segment formed by the first and last control points $\overline{\mathbf{P}_0\mathbf{P}_n}$.

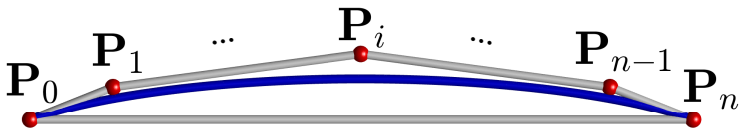


Figure 4. Illustration of pseudo-curvature concept applied to a B-spline curve.

The principle underlying the pseudo-curvature defined in Equation (27) capitalizes on the convex hull property inherent to B-spline curves, which ensures that the B-spline curve lies entirely within the convex polygon formed by its control points. Leveraging the convex hull property, we ascertain that the bending of the B-spline curve approaches its minimum when the value of the pseudo-curvature in Equation (27) approaches 1.

The pseudo-curvature offers a significant reduction in computational complexity compared to traditional curvature metrics. By leveraging the geometric properties inherent in the control points of B-spline curves, we can obtain a useful approximation of the original curve with far less computational effort.

4.2. Subdivision-Based Potential Intersection Ranges Computation

Central to our methodology is a sophisticated subdivision scheme tailored to the unique characteristics of B-spline curves, which significantly enhances the efficiency and accuracy of intersection point identification. With the aforementioned pseudo-curvature concept established in Section 4.1, we have the essential tools for splitting curves effectively. In this section, we integrate subdivision techniques with bounding box intersection detection to compute potential intersection ranges between B-spline curves efficiently. The algorithmic framework for this process is presented in Algorithm 2.

The algorithm depicted in Algorithm 2 begins by computing the bounding boxes for the input curves. It then proceeds to check for bounding box intersections. If no intersection is detected, the algorithm concludes that the curves themselves cannot intersect, reducing unnecessary computation.

In cases where bounding boxes intersect, the algorithm evaluates the pseudo-curvature of the curves in Equation (27). If the curvature falls below a predefined threshold $\kappa_{\max}$, indicative of minimal bending, direct intersection checks between curve segments are performed. Conversely, if the curvature exceeds this threshold, suggesting potential complex intersections, the algorithm recursively applies itself to subdivided curve segments, iteratively refining the search space.

This iterative process continues until all potential intersection ranges have been exhaustively examined, ensuring a comprehensive identification of intersection points without omission.

In Algorithm 2, the selection of $\kappa_{\max}$ is critical. Setting it too low may result in excessive subdivision, thereby prolonging computation time. Conversely, setting it too high may risk overlooking complex intersections. Striking the right balance for this parameter is essential to achieving optimal performance. In our numerical experiments, we set the default value of $\kappa_{\max}$ to $1 + 5 \times 10^{-6}$.

While bounding boxes provide an initial filter for potential intersections, the precision of intersection detection relies on the subsequent subdivision and direct intersection checks. Maintaining geometric precision throughout the process is crucial. To enhance computational accuracy and efficiency, we employ the projected Gauss-Newton method in the following section.

Algorithm 2: Subdivision-based potential intersection ranges computation

Input: Curve $\mathcal{C}_A$, curve $\mathcal{C}_B$, max curvature $\kappa_{\max}$.
Output: Vector of potential intersection ranges.

$h_A \leftarrow$ BoundingBox($\mathcal{C}_A$);
$h_B \leftarrow$ BoundingBox($\mathcal{C}_B$);
if h_A *does NOT intersect* h_B **then**
  return $\varnothing$;
end
Compute pseudo-curvatures $\kappa(\mathcal{C}_A)$ and $\kappa(\mathcal{C}_B)$; // Equation (27)
if $\kappa(\mathcal{C}_A) \leq \kappa_{max}$ ***and*** $\kappa(\mathcal{C}_B) \leq \kappa_{max}$ **then**
  return Potential intersection range h_A and h_B;
end
if $\kappa(\mathcal{C}_A) > \kappa_{max}$ ***and*** $\kappa(\mathcal{C}_B) > \kappa_{max}$ **then**
  Split both curves $\mathcal{C}_A$ and $\mathcal{C}_B$ at their central parameters;
  Recursively call this function;
else if $\kappa(\mathcal{C}_A) \leq \kappa_{max}$ ***and*** $\kappa(\mathcal{C}_B) > \kappa_{max}$ **then**
  Split only $\mathcal{C}_B$ at its central parameter and recursively call this function;
else if $\kappa(\mathcal{C}_B) \leq \kappa_{max}$ ***and*** $\kappa(\mathcal{C}_A) > \kappa_{max}$ **then**
  Split only $\mathcal{C}_A$ at its central parameter and recursively call this function;
else
  return $\varnothing$;
end

4.3. Projected Gauss-Newton Method

Upon obtaining the output from Algorithm 2, we acquire pairs of potential intersection ranges where the curve segments exhibit very limited bending. According to Bézout's theorem, as these segments closely resemble straight lines (degree-1 curves), only one intersection point is anticipated within each pair. Hence, leveraging Gauss-Newton iteration proves to be an efficient approach for intersection computation. However, it is essential to note that the computed results may extend beyond the parameter ranges of the current segments. To mitigate this issue, we implement the projected Gauss-Newton method, thereby ensuring that the computed intersections remain within the designated segments.

Consider a nonlinear system $\boldsymbol{\mathcal{F}}(\boldsymbol{\xi}) = \mathbf{0}$. The essence of the projected Gauss-Newton method is captured by a two-fold iterative process. Initially, we calculate an intermediate point $\widetilde{\boldsymbol{\xi}}^{k+1}$ using the classic Gauss-Newton iteration formula

$$\widetilde{\boldsymbol{\xi}}^{k+1} = \boldsymbol{\xi}^k - \boldsymbol{J}_k^{\dagger} \boldsymbol{F}(\boldsymbol{\xi}^k), \tag{28}$$

where $\boldsymbol{J}_k^{\dagger} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$ is the Moore-Penrose inverse of the Jacobian matrix of the nonlinear system $\boldsymbol{\mathcal{F}}$ at iteration k. The Moore-Penrose inverse $\boldsymbol{J}_k^{\dagger}$ can be computed using the singular value decomposition (SVD).

Subsequently, we refine our estimate by projecting $\widetilde{\boldsymbol{\xi}}^{k+1}$ onto the feasible set $\mathbb{D}_k = [\xi_{\min}^k, \xi_{\max}^k] \times [\eta_{\min}^k, \eta_{\max}^k]$, optimizing for proximity to the initial estimate

$$\boldsymbol{\xi}^{k+1} = \mathbb{P}_{\boldsymbol{\xi} \in \mathbb{D}_k}\left(\widetilde{\boldsymbol{\xi}}^{k+1}\right), \tag{29}$$

where $\mathbb{P}_{\boldsymbol{\xi} \in \mathbb{D}_k}$ denotes the projection operator. In our implementation, we adopt the following projection operator

$$\mathbb{P}_{\boldsymbol{\xi} \in \mathbb{D}_k}(\widetilde{\boldsymbol{\xi}}) = \left(\min\left(\max(\widetilde{\xi}, \xi_{\min}^k), \xi_{\max}^k\right), \min\left(\max\left(\widetilde{\eta}, \eta_{\min}^k\right), \eta_{\max}^k\right)\right). \tag{30}$$

This projection ensures our solution remains within the parameter ranges of the curves, an essential consideration for maintaining geometric validity.

In the case of 3D scenarios, our approach to the curve intersection problem involves addressing the following nonlinear system

$$\boldsymbol{\mathcal{F}}(\xi,\eta)=\boldsymbol{\mathcal{C}}_A(\xi)-\boldsymbol{\mathcal{C}}_B(\eta)=\begin{pmatrix} x_A(\xi)-x_B(\eta) \\ y_A(\xi)-y_B(\eta) \\ z_A(\xi)-z_B(\eta) \end{pmatrix}=\mathbf{0}. \tag{31}$$

The corresponding Jacobian matrix, a 3×2 matrix, can be computed as follows:

$$\boldsymbol{\mathcal{J}}(\xi,\eta)=\begin{bmatrix} \frac{\mathrm{d}\boldsymbol{\mathcal{C}}_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}\boldsymbol{\mathcal{C}}_B(\eta)}{\mathrm{d}\eta} \end{bmatrix}=\begin{bmatrix} \frac{\mathrm{d}x_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}x_B(\eta)}{\mathrm{d}\eta} \\ \frac{\mathrm{d}y_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}y_B(\eta)}{\mathrm{d}\eta} \\ \frac{\mathrm{d}z_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}z_B(\eta)}{\mathrm{d}\eta} \end{bmatrix}. \tag{32}$$

For the 2D case, the nonlinear system becomes

$$\boldsymbol{\mathcal{F}}(\xi,\eta)=\boldsymbol{\mathcal{C}}_A(\xi)-\boldsymbol{\mathcal{C}}_B(\eta)=\begin{pmatrix} x_A(\xi)-x_B(\eta) \\ y_A(\xi)-y_B(\eta) \end{pmatrix}=\mathbf{0}. \tag{33}$$

The corresponding Jacobian matrix, a 2×2 matrix, is computed by

$$\boldsymbol{\mathcal{J}}(\xi,\eta)=\begin{bmatrix} \frac{\mathrm{d}\boldsymbol{\mathcal{C}}_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}\boldsymbol{\mathcal{C}}_B(\eta)}{\mathrm{d}\eta} \end{bmatrix}=\begin{bmatrix} \frac{\mathrm{d}x_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}x_B(\eta)}{\mathrm{d}\eta} \\ \frac{\mathrm{d}y_A(\xi)}{\mathrm{d}\xi} & -\frac{\mathrm{d}y_B(\eta)}{\mathrm{d}\eta} \end{bmatrix}. \tag{34}$$

Different from the 3D case, the Moore-Penrose inverse in Equation (28) degenerates into the inverse of the Jacobian matrix (34).

The comprehensive algorithm is presented in Algorithm 3.

Algorithm 3: Projected Gauss-Newton method

Input: Initial guess $\boldsymbol{\xi}_0=\{\xi_0,\eta_0\}$,
Curve $\boldsymbol{\mathcal{C}}_A$ and curve $\boldsymbol{\mathcal{C}}_B$,
Residual tolerance tol_res,
$\boldsymbol{\xi}$ variation tolerance tol_ξ,
Maximum iterations $maxIter$.
Output: Residual r, Intersection parameter pair $\boldsymbol{\xi}=\{\xi,\eta\}$.

Compute initial nonlinear system $\boldsymbol{\mathcal{F}}$ and its Jacobian $\boldsymbol{\mathcal{J}}$ // Equations (31) and (32)
Compute initial residual $r\leftarrow \mathrm{norm}(\boldsymbol{\mathcal{F}})$;
$\Delta\leftarrow \boldsymbol{\mathcal{J}}^{\dagger}\boldsymbol{\mathcal{F}}$;
for $k\leftarrow 1$ **to** $maxIter$ **do**
  $\boldsymbol{\xi}_0\leftarrow\boldsymbol{\xi}_0-\Delta$; // Gauss-Newton iteration, Equation (28)
  $\boldsymbol{\xi}\leftarrow\mathbb{P}_{\xi\in\mathbb{D}_k}\left(\tilde{\boldsymbol{\xi}}\right)$; // Projection, Equation (29)
  $\boldsymbol{\xi}_0\leftarrow\boldsymbol{\xi}$; // Update parameter pair
  if $\mathrm{norm}(\boldsymbol{\xi}-\boldsymbol{\xi}_0)>100*\mathrm{MACHINE\_EPSILON}$ **then**
    Evaluate $\boldsymbol{\mathcal{F}}$ at $\boldsymbol{\xi}_0$ and update current residual $r\leftarrow\mathrm{norm}(\boldsymbol{\mathcal{F}})$;
    break;
  end
  Evaluate $\boldsymbol{\mathcal{F}}$ and its Jacobian $\boldsymbol{\mathcal{J}}$ at $\boldsymbol{\xi}_0$; // Equations (31) and (32)
  Compute current residual $r\leftarrow\mathrm{norm}(\boldsymbol{\mathcal{F}})$;
  if $r<tol\_res$ **then**
    break;
  end
  $\Delta\leftarrow\boldsymbol{\mathcal{J}}^{\dagger}\boldsymbol{\mathcal{F}}$;
  if $\mathrm{norm}(\Delta)<tol\_\xi$ **then**
    break;
  end
end

5. Numerical Experiments

This section presents numerical experiments conducted to evaluate the effectiveness and efficiency of the proposed algorithm across both Bézier curves and B-spline curves. We implemented the algorithm in C++ using Clang-15 compiler and conducted the experiments on a MacBook Pro (14-inch, 2021) equipped with an Apple M1 Pro CPU and 16 GB of RAM. Matlab® 2023a and ParaView (version 5.10.1) were employed for visualization.

In all experiments, the parameter settings were standardized as follows: maximum curvature, $\kappa_{\max}$, was set to $1 + 5 \times 10^{-6}$; residual tolerance, tol_res, was defined at 1×10^{-5}; variation tolerance for ξ, denoted as tol_{ξ}, was also set at 1×10^{-5}; and the limit for the maximum number of iterations, $maxIter$, was fixed at 20. Computing all the intersections between two Bézier curves is a topic that has received considerable attention in the CAGD community. The Bézier clipping algorithm, as proposed by Sederberg in 1990 [18], stands out for its second-order convergence rate, a finding further supported by Schulz in 2009 [19]. Since then, the algorithm has seen several enhancements aimed at improving its efficiency. Noteworthy among these are the quadratic and cubic clipping methods [20,22], which are based on degree reduction techniques; the geometry interval clipping algorithm [23]; and the cubic hybrid clipping method [25].

For B-spline curves, a prevalent strategy is to first convert them into Bézier segments using the Bézier extraction technique. This transformation allows the application of the previously mentioned clipping methods to each pair of Bézier segments, facilitating the computation of intersections. In this section, we compare our proposed method with these existing intersections computation techniques through detailed analysis. Specifically, different methods are conducted on four examples depicted in Figure 5, which include cubic Bézier curves with two intersections, a quadratic B-spline and a cubic B-spline each with two intersections, and two cubic B-splines with eight intersections, including a scenario where two intersection are notably proximal.

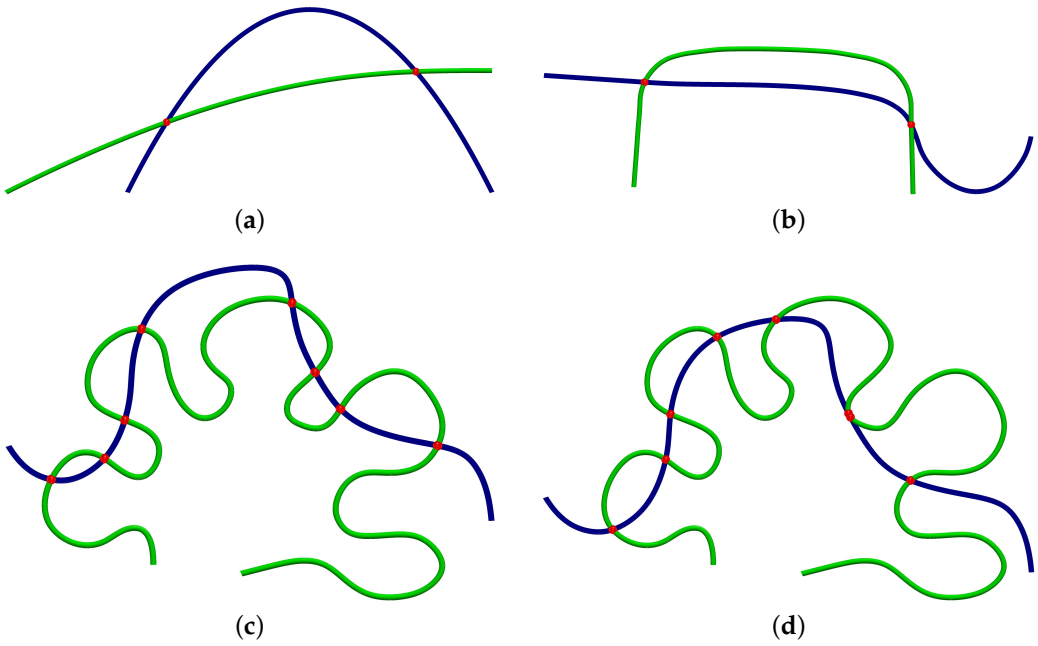


Figure 5. Examples used for comparisons of different methods. (**a**) Example 1: cubic Bézier curves with 2 intersections. (**b**) Example 2: quadratic B-spline and cubic B-spline with 2 intersections. (**c**) Example 3: cubic B-splines with 8 intersections. (**d**) Example 4: cubic B-splines with 8 intersections.

Table 1 presents a comprehensive comparison of the computational efficiency between our proposed method and several established techniques, including Bézier Clipping [18], Quadratic Degree Reduction [20], Cubic Degree Reduction [22], Geometry Interval Clipping [23], and Cubic Hybrid Clipping [25]. The evaluation encompasses a range of scenarios, as depicted in the table and Figure 5, characterized by varying numbers of control points, Bézier segments, and intersections, thereby offering a broad perspective on performance across different curve complexities.

Table 1. Computation results comparison of the proposed method against existing methods. The curves information includes the numbers of control points, Bézier segments, and intersections for clearer interpretation. Data highlighted in **bold** indicate the best performance.

Example	Curves Info.	Methods	Time (μs)	Relative Time
Example 1 (Figure 5a)	$\mathcal{C}_A$: 4 / 1 / 2 $\mathcal{C}_B$: 4 / 1 / 2	Bézier Clipping [18]	210	1.84
		Quadratic Degree Reduction [20]	3293	28.89
		Cubic Degree Reduction [22]	17,074	149.77
		Geometry Interval Clipping [23]	2941	25.79
		Cubic Hybrid Clipping [25]	2861	25.10
		Our Method	**114**	**1.00**
Example 2 (Figure 5b)	$\mathcal{C}_A$: 29 / 26 / 2 $\mathcal{C}_B$: 29 / 27 / 2	Bézier Clipping [18]	6435	38.53
		Quadratic Degree Reduction [20]	7312	43.78
		Cubic Degree Reduction [22]	7028	42.08
		Geometry Interval Clipping [23]	4219	25.26
		Cubic Hybrid Clipping [25]	1873	11.22
		Our Method	**167**	**1.00**
Example 3 (Figure 5c)	$\mathcal{C}_A$: 63 / 60 / 8 $\mathcal{C}_B$: 24 / 21 / 8	Bézier Clipping [18]	2176	3.74
		Quadratic Degree Reduction [20]	4440	7.63
		Cubic Degree Reduction [22]	3384	5.81
		Geometry Interval Clipping [23]	2066	3.55
		Cubic Hybrid Clipping [25]	1667	2.86
		Our Method	**582**	**1.00**
Example 4 (Figure 5d)	$\mathcal{C}_A$: 63 / 60 / 8 $\mathcal{C}_B$: 24 / 21 / 8	Bézier Clipping [18]	1863	4.05
		Quadratic Degree Reduction [20]	3948	8.58
		Cubic Degree Reduction [22]	2970	6.46
		Geometry Interval Clipping [23]	1558	3.39
		Cubic Hybrid Clipping [25]	1184	2.57
		Our Method	**460**	**1.00**

The findings underscore the superior performance of our method, consistently achieving the lowest computation times (highlighted in **bold**) across all examples. Specifically, in Example 1 (Figure 5a), our method demonstrates a significant efficiency advantage, achieving a computation time of only 114 microseconds, which is substantially lower than the next fastest method, Bézier Clipping, by a factor of 1.84. This trend of outperformance persists across the other examples, with our method maintaining the position of optimal efficiency with the lowest relative times of 1.00 in all cases.

Example 2 (Figure 5b) further illustrates the robustness of our approach, handling complex curves (29 control points, 26/27 Bézier segments, and 2 intersections) with a computation time of merely 167 microseconds, whereas the competing methods exhibit significantly higher times, up to 43.78 times slower in the case of quadratic degree reduction.

In Examples 3 (Figure 5c) and 4 (Figure 5d), which involve an even greater number of intersections and control points, our method continues to exhibit unparalleled efficiency, further evidencing its potential as a superior computational tool for the intersection of Bézier and B-spline curves.

Traditional techniques require computing intersections for each pair of Bézier segments, leading to a computational complexity of $\mathcal{O}(\#BezSeg^2)$, with $\#BezSeg$ denoting the number of Bézier segments. For instance, in Example 3, curve $\boldsymbol{\mathcal{C}}_A$ comprises 60 Bézier segments, as illustrated in Figure 6, whereas curve $\boldsymbol{\mathcal{C}}_B$ contains 21 segments. Consequently, the process of determining Bézier intersections must be executed as many as $60 \times 21 = 1260$ times. As the number of Bézier segments grows—reflecting an increase in the number of inner knot values—the computational demand escalates.

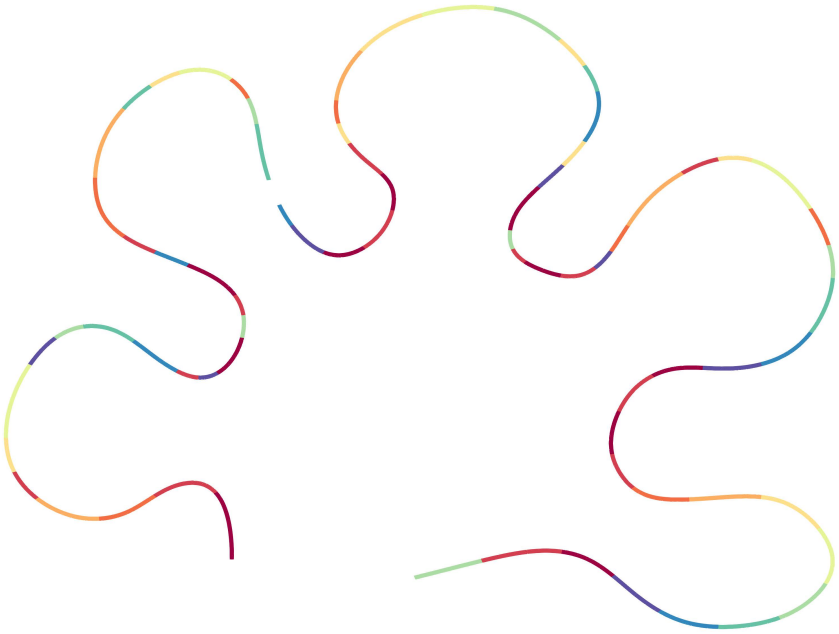


Figure 6. Illustration of Bézier segments for curve $\boldsymbol{\mathcal{C}}_A$ in Example 3, which is composed of 60 segments.

In contrast, our method is engineered to function independently of Bézier curve segmentation. As demonstrated in Figure 7, the computation time of our method remains constant regardless of the increase in Bézier segment count. However, the computation times of the other methods rise linearly with $\#BezSeg^2$, which coincides with the theoretical computational complexity analysis. This design choice eliminates the necessity for intersection calculations between segment pairs, thereby significantly reducing computational overhead. This strategic departure from segment-pair dependency not only streamlines the process but also enhances efficiency, providing a robust solution for handling complex curve intersections with greater computational economy.

The comparative analysis not only highlights the computational merits of our proposed method but also underscores the significance of algorithmic efficiency in the realm of geometric computation, especially for applications requiring the rapid processing of complex B-spline curve intersections.

However, it is crucial to highlight that the selection of parameters in the algorithm presented in this paper is heuristic. A larger tolerance will result in the obtained intersection point deviating from the actual position. Conversely, a smaller tolerance will increase the number of iterations, thereby reducing the speed of the solution. The selection of the most suitable parameters is therefore imperative for users, following the specific circumstances of the problem. Furthermore, the subdivision method based on pseudo-curvature is only applicable to parametric curves formed by control points and basis functions.

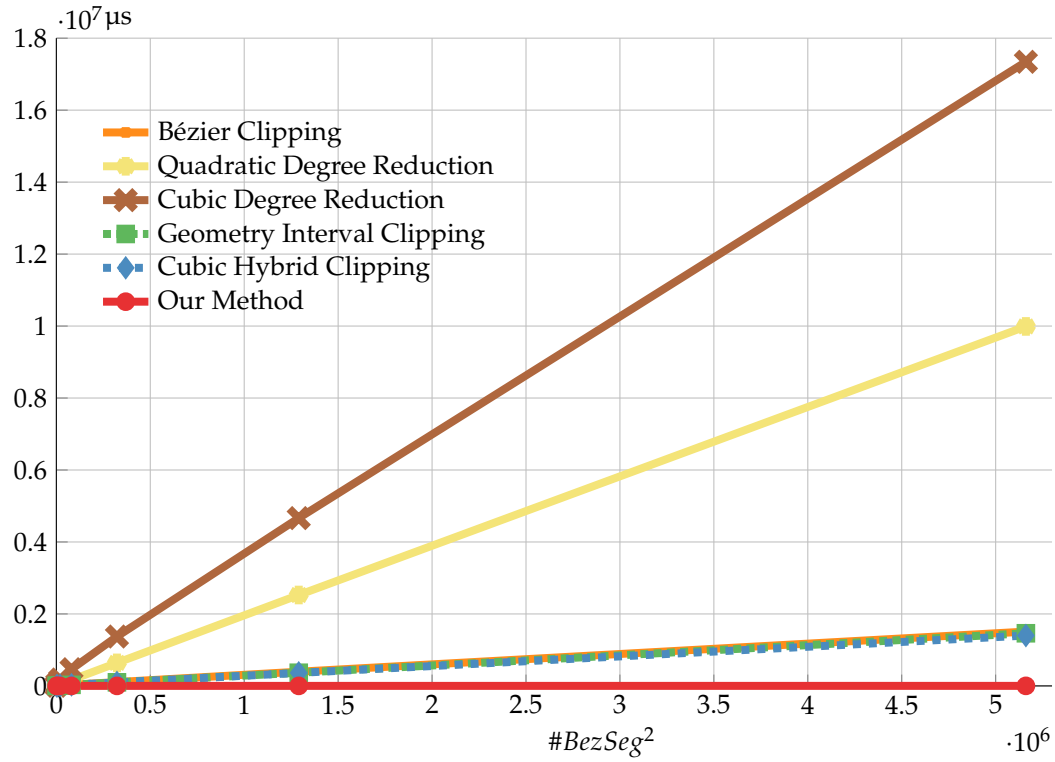


Figure 7. Comparison of computation times across different methods w.r.t. the square of Bézier segment. This assessment encompasses Bézier clipping [18], quadratic degree reduction [20], cubic degree reduction [22], geometric interval clipping [23], cubic hybrid clipping [25], and our method.

6. Conclusions

In this paper, we develop a rapid and robust algorithm designed to address the intersection challenges of Bézier and B-spline curves, which are fundamental components in the realms of computer graphics and computer-aided geometric design. By incorporating Sturm's theorem, we have established a methodical approach to determine the number of intersections within a predetermined region. To handle multiple intersections, a fast subdivision-based algorithm is proposed. The bounding box detection and the definition of pseudo-curvature for curve segmentation further refine the process, enabling the partitioning of curves into segments that closely emulate straight lines. The deployment of the projected Gauss-Newton method facilitates efficient convergence to intersection points, marking a significant advancement over traditional techniques.

For future work, the exploration of efficient and robust algorithms for the surface/surface intersection problem presents a more challenging and interesting frontier.

Author Contributions: Conceptualization, Y.-Y.Y. and Y.J.; methodology, Y.-Y.Y. and Y.J.; software, Y.-Y.Y., X.L. and Y.J.; validation, X.L.; formal analysis, Y.-Y.Y.; investigation, Y.-Y.Y., X.L. and Y.J.; writing—original draft preparation, Y.-Y.Y. and Y.J.; writing—review and editing, Y.-Y.Y., X.L. and Y.J.; visualization, Y.-Y.Y. and Y.J.; project administration, Y.-Y.Y.; funding acquisition, Y.-Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (No. 12301490), the Youth Foundation of Liaoning Provincial Department of Education (No. JYTQN2023262), and the Young Science and technology Talent Foundation of Dalian Science and Technology Bureau (No. 2023RQ088).

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Farin, G. *Curves and Surfaces for CAGD: A Practical Guide*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2001.
2. Choi, Y.K.; Wang, W.P.; Liu, Y.; Kim, M. Continuous collision detection for two moving elliptic disks. *IEEE Trans. Robot.* **2006**, *22*, 213–224. [CrossRef]
3. Patrikalakis, N. Surface-to-surface intersections. *IEEE Comput. Graph. Appl.* **1993**, *13*, 89–95. [CrossRef]
4. Kruppa, K.; Kunkli, R.; Hoffmann, M. A skinning technique for modeling artistic disk B-spline shapes. *Comput. Graph.* **2023**, *115*, 96–106. [CrossRef]
5. Yousif, M.A.; Hamasalh, F.K. The fractional non-polynomial spline method: Precision and modeling improvements. *Math. Comput. Simul.* **2024**, *218*, 512–525. [CrossRef]
6. Hoffmann, C.M. *Geometric and Solid Modeling*; Morgan Kaufmann: Burlington, MA, USA, 1989.
7. Wyvill, B.; Kees, V.O. Polygonization of implicit surfaces with constructive solid geometry. *Int. J. Shape Model.* **1996**, *2*, 257–274. [CrossRef]
8. Nishita, T.; Sederberg, T.W.; Kakimoto, M. Ray tracing trimmed rational surface patches. In Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques, Dallas, TX, USA, 6–10 August 1990; pp. 337–345.
9. Efremov, A.; Havran, V.; Seidel, H.P. Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. In Proceedings of the 21st Spring Conference on Computer Graphics, Budmerice, Slovakia, 12–14 May 2005; pp. 127–135.
10. Lin, M.C.; Gottschalk, S. Collision detection between geometric models: A survey. In Proceedings of the IMA Conference on Mathematics of Surfaces, 1998; Volume 1, pp. 602–608. Available online: https://api.semanticscholar.org/CorpusID:9315916 (accessed on 1 April 2024).
11. Patrikalakis, N.M.; Maekawa, T. *Shape Interrogation for Computer Aided Design and Manufacturing*; Springer: Berlin/Heidelberg, Germany, 2002; Volume 15.
12. Ji, Y.; Yu, Y.Y.; Wang, M.Y.; Zhu, C.G. Constructing high-quality planar NURBS parameterization for isogeometric analysis by adjustment control points and weights. *J. Comput. Appl. Math.* **2021**, *396*, 113615. [CrossRef]
13. Galligo, A.; Pavone, J.P. Self-intersections of a Bézier bicubic surface. In Proceedings of the International Symposium on Symbolic and Algebraic Computation, Beijing, China, 24–27 July 2005.
14. Yu, Y.Y.; Li, X.; Ji, Y. On self-intersections of cubic Bézier curves. *Mathemetics* **2024**, *12*, 882. [CrossRef]
15. Pekerman, D.; Elber, G.; Kim, M.S. Self-intersection detection and elimination in freeform curves and surfaces. *Comput.-Aided Des.* **2008**, *40*, 150–159. [CrossRef]
16. Elber, G.; Grandine, T.; Kim, M.S. Surface self-intersection computation via algebraic decomposition. *Comput.-Aided Des.* **2009**, *41*, 1060–1066. [CrossRef]
17. Yang, J.Y.; Jia, X.H.; Yan, D.M. Topology Guaranteed B-Spline Surface/Surface Intersection. *ACM Trans. Graph.* **2023**, *42*, 1–16. [CrossRef]
18. Sederberg, T.W.; Nishita, T. Curve intersection using Bézier clipping. *Comput.-Aided Des.* **1990**, *22*, 538–549. [CrossRef]
19. Schulz, C. Bézier clipping is quadratically convergent. *Comput. Aided Geom. Des.* **2009**, *26*, 61–74. [CrossRef]
20. Bartoň, M.; Jüttler, B. Computing roots of polynomials by quadratic clipping. *Comput. Aided Geom. Des.* **2007**, *24*, 125–141. [CrossRef]
21. Liu, L.; Zhang, L.; Lin, B.B.; Wang, G.J. Fast approach for computing roots of polynomials using cubic clipping. *Comput. Aided Geom. Des.* **2009**, *26*, 547–559. [CrossRef]
22. Lou, Q.; Liu, L. Curve intersection using hybrid clipping. *Comput. Graph.* **2012**, *36*, 309–320. [CrossRef]
23. North, N.S. Intersection Algorithms Based on Geometric Intervals. Ph.D. Thesis, Brigham Young University, Provo, UT, USA, 2007.
24. Yuan, Q. Study on Hybrid Clipping Method for Solving Polynomial Roots. Ph.D. Thesis, Zhejiang University, Hangzhou, China, 2012.
25. Wu, Y.; Li, X. Curve intersection based on cubic hybrid clipping. *Vis. Comput. Ind. Biomed. Art* **2022**, *5*, 17. [CrossRef] [PubMed]
26. Mishra, B. *Algorithmic Algebra*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
27. Bensimhoun, M. Historical account and ultra-simple proofs of Descartes's rule of signs, De Gua, Fourier, and Budan's rule. *arXiv* **2013**, arXiv:1309.6664.
28. Shao, W.B.; Chen, F.L.; Liu, X.F. Robust Algebraic Curve Intersections with Tolerance Control. *Comput.-Aided Des.* **2022**, *147*, 103236. [CrossRef]
29. Milne, P.S. On the solutions of a set of polynomial equations. In *Symbolic and Numerical Computation for Artificial Intelligence*; Academic Press: Cambridge, MA, USA, 1992; pp. 89–102.