Visualization of Curvature Monotonicity Regions of 3D Bézier Curves in 2D and 3D

Norimasa Yoshida[1], Takafumi Saito[2]

[1]Nihon University, yoshida.norimasa@nihon-u.ac.jp
[2]Tokyo University of Agriculture and Technology, txsaito@cc.tuat.ac.jp

Corresponding author: Norimasa Yoshida, yoshida.norimasa@nihon-uac.jp

Abstract. This paper presents a novel method for visualizing the curvature monotonicity region of 3D Bézier curves in both 2D and 3D, a concept not previously explored. For 2D visualization, we extend existing techniques for 2D Bézier curves to 3D polynomial and rational Bézier curves. The curvature monotonicity region is visualized on a constant-depth plane through the control point of interest, enabling users to identify regions where curvature varies monotonically. For 3D visualization, we propose a two-step algorithm to directly visualize the curvature monotonicity region. We demonstrate several 2D and 3D examples, including cases based on a sufficient condition, and present a curve design tool incorporating 2D curvature monotonicity visualization.

Keywords: 2D and 3D curvature monotonicity region, 3D Bézier curves, GPU
DOI: https://doi.org/10.14733/cadaps.2026.41-55

1 INTRODUCTION

Designing aesthetically pleasing surfaces, such as automobile exteriors, relies on fair curves. In [2], Farin defined fair curves as those with curvature plots that exhibit minimal regions of monotonically varying curvature. Numerous studies, including [17, 3, 21], have focused on generating 2D curves with monotonically varying curvature. However, for 3D curves, relatively few works have addressed the problem, and no study has explored visualizing the region around a control point where the curvature varies monotonically. By visualizing this region, users can more easily generate 3D curves with monotonically varying curvature.

We propose methods to visualize the curvature monotonicity region of 3D polynomial and rational Bézier curves in both 2D and 3D. The curvature monotonicity region refers to the area around a control point where the curvature varies monotonically. We introduce two methods for visualizing this region: (1) a 2D visualization that extends GPU-accelerated techniques for 2D Bézier curves [24, 9], and (2) a 3D visualization. Since the 3D curvature monotonicity region is not an algebraic surface, existing visualization methods for algebraic surfaces cannot be directly applied to our problem. The main contributions are summarized as follows.

1. We propose a method to visualize the curvature monotonicity region of a 3D Bézier curve in 2D on a constant-depth plane from the viewpoint. The visualization is real-time and GPU-accelerated. If a user moves the control point within the region, the user can generate a curve with monotonically varying curvature.
2. We propose a method to visualize the curvature monotonicity region of a 3D Bézier curve in 3D. The 3D curvature monotonicity region is not an algebraic surface. We introduce a two-step algorithm to visualize the region using a GPU.
3. We introduce a 3D curve design tool that utilizes 2D curvature monotonicity visualization on a constant-depth plane and present designed examples. Our tool allows users to identify the region around a control point where the curvature varies monotonically.

The paper is organized as follows. Section 2 reviews related work on generating curves with monotonically varying curvature, visualizing algebraic surfaces, and curvature monotonicity evaluation functions. In Section 3, we propose methods for visualizing the curvature monotonicity regions of 3D Bézier curves in both 2D and 3D. Several examples of these visualizations are presented, including one based on a sufficient condition. Section 4 introduces a curve design tool that utilizes our 2D curvature monotonicity visualization and showcases designed examples. Finally, conclusions are presented in Section 5.

A demonstration video is available at

https://youtu.be/SM0xeClSdQU.

2 RELATED WORK

2.1 Freeform Curves with Monotonically Varying Curvature

Sapidis et al. [11] clarified the curvature monotonicity region of quadratic Bézier curves, while Frey et al. [5] extended this to rational quadratic Bézier curves. Diets et al. used precomputed tables to generate cubic curves with monotonically varying curvature [1]. Wang et al. proposed a shape control method for Bézier and B-spline curves based on sufficient monotone curvature variation conditions [17]. Farin proposed class A Bézier curves [3], where the control points are generated by repeatedly applying a matrix M to the first edge of the control polygon. Yoshida et al. proposed a method for interactively generating class A Bézier curves [20], similar to quadratic Bézier curves . Their method is primarily for typical curves. Romani et al. [8] proved that if M has two real eigenvalues, the curve becomes class A. Yoshida et al. [21] proposed a method for generating rational cubic Bézier curves with monotonically varying curvature by approximating log-aesthetic curves.

Unlike most of the above approaches that theoretically investigate the curves with monotonically varying curvature, Yoshida et al. introduced a real-time GPU-based visualization method for the curvature monotonicity region of cubic or higher-degree Bézier curves [24]. In their approach, users can identify the region of a control point where the curvature varies monotonically. Saito et al. further extended this approach to rational Bézier curves by introducing curvature monotonicity evaluation functions based on the Bernstein basis [9]. Yoshida et al. visualized the curvature monotonicity regions based on a sufficient condition provided in [23]. The region can be represented as the intersection of implicit curves. They observed that the size of the sufficient region is slightly smaller than that of the exact curvature monotonicity region in many cases, though not always. They also noted that, in some situations, the implicit curves may form the exact boundary of the curvature monotonicity region.

For 3D curves with monotonically varying curvature, there is less related work compared to 2D curves. Yoshida et al. [19] proposed a method for interactively generating 3D class A Bézier curves by specifying two endpoints and their tangents. Their method is primarily designed for typical 3D class A Bézier curves. In [18], Yoshida et al. showed that 3D typical class A Bézier curves get closer to 3D logarithmic spirals as the degree

increases. Tong et al. [15] proposed a sufficient condition for 3D typical curves. Wang et al. [16] presented a class of 3D Bézier curves of arbitrary degree with monotone curvature by generalizing typical curves [6] introduced by Minuer. No methods exist that visualize the curvature monotonicity region of a 3D Bézier curve in either 2D or 3D. Visualizing the curvature monotonicity region of a 3D Bézier curve enables users to move a control point while maintaining the curvature monotonicity.

2.2 Visualization of Algebraic Surfaces

Our visualization of the 3D curvature monotonicity region is closely related to the visualization of algebraic surfaces, although the region itself is not an algebraic surface. For visualizing algebraic surfaces, GPUs are typically used to enable fast rendering. In [12], Seland et al. proposed a real-time method for visualizing algebraic surfaces using the blossoming principle of trivariate Bernstein-Bézier functions over a tetrahedron. In ray casting for algebraic surfaces, finding the intersection of a ray and an algebraic surface of degree d is equivalent to finding the roots of a univariate polynomial of degree d. Reimers et al. represented the univariate polynomial in Bernstein form to efficiently compute its roots [7]. Singh et al. proposed an adaptive marching points algorithm and demonstrated ray tracing of algebraic surfaces up to order 50 at interactive frame rates [13]. The 3D curvature monotonicity region that we aim to visualize is not an algebraic surface and cannot be represented by a simple equation, such as those for a torus, superquadrics, or metaballs. Additionally, the surface normal required for lighting cannot be computed straightforwardly, as is done for algebraic surfaces. Therefore, a different method is needed to visualize the curvature monotonicity region in 3D.

2.3 Curvature Monotonicity Evaluation Function

This section reviews the curvature monotonicity evaluation functions of 3D Bézier curves introduced by Saito et al. [9]. We use these functions to visualize the curvature monotonicity regions of 3D Bézier curves.

A 3D rational Bézier curve $\mathbf{P}(t)$ of degree $n(\geq 3)$ with $n+1$ control point vectors $\boldsymbol{p}_i = [x_i \ y_i \ z_i]^{\mathrm{T}}$ $(0 \leq i \leq n)$ and the weight $w_i(>0)$ is

$$\mathbf{P}(t) = \frac{\sum_{i=0}^{n} B_i^n(t) w_i \boldsymbol{p}_i}{\sum_{i=0}^{n} B_i^n(t) w_i} = \frac{\boldsymbol{Q}(t)}{W(t)}, \tag{1}$$

where $B_i^n(t)$ is a Bernstein polynomial of degree n. We assume $n \geq 3$ since if $n = 2$, the curve becomes planar. We also assume that the curve is regular and the curvature is non-zero. If all the weights w_i are 1, the curve is a polynomial curve. For a 3D curve $\mathbf{P}(t)$, the derivative of the curvature with respect to the arc length is [18]:

$$\frac{d\kappa}{ds} = \frac{\left((\dot{\mathbf{P}} \wedge \dddot{\mathbf{P}}) \cdot (\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}})\right)(\dot{\mathbf{P}} \cdot \dot{\mathbf{P}}) - 3\left((\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}}) \cdot (\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}})\right)(\dot{\mathbf{P}} \cdot \ddot{\mathbf{P}})}{(\dot{\mathbf{P}} \cdot \dot{\mathbf{P}})^3 |\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}}|}, \tag{2}$$

where $\cdot$ and $\wedge$ are dot and cross products, and $\dot{\mathbf{P}}$, $\ddot{\mathbf{P}}$ and $\dddot{\mathbf{P}}$ are the first, second, and third derivative of $\mathbf{P}(t)$ with respect to t. We write the numerator of Eq. (2) as

$$L_n(t) = \left((\dot{\mathbf{P}} \wedge \dddot{\mathbf{P}}) \cdot (\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}})\right)(\dot{\mathbf{P}} \cdot \dot{\mathbf{P}}) - 3\left((\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}}) \cdot (\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}})\right)(\dot{\mathbf{P}} \cdot \ddot{\mathbf{P}}). \tag{3}$$

With the assumption that the curve is regular and the curvature is non-zero, the denominator of Eq. (2) is always positive. Therefore, the curvature monotonicity can be evaluated by $L_n(t)$. For rational curves, $L_n(t)$ is a rational function and its denominator is always positive. Therefore, the curvature monotonicity is evaluated by the numerator of $L_n(t)$, which we write as $l_n(t)$. See Eq. (8). We refer to $L_n(t)$ or $l_n(t)$ as the curvature monotonicity evaluation function.

Saito et al. derived equations of $L_n(t)$ and $l_n(t)$ in Bernstein basis [9]. We use these Bernstein basis equations to visualize the curvature monotonicity regions of 3D curves. To represent $L_n(t)$ and $l_n(t)$ in

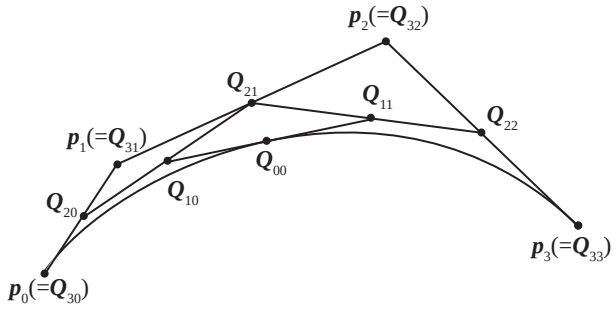


Figure 1: $\boldsymbol{Q}_{m,k}$ of a cubic Bézier curve at $t = 0.5$

Bernstein basis, we use internal division points and weights of de Casteljau's algorithm [4]. For $0 \leq k \leq m \leq n$), k-th internal division point $\boldsymbol{Q}_{m,k}(t)$ and weight $W_{m,k}(t)$ at $(n-m)$-th step are:

$$\boldsymbol{Q}_{m,k}(t) = \sum_{i=0}^{n-m} B_i^{n-m} w_{k+i} \boldsymbol{p}_{k+i}, \tag{4}$$

$$W_{m,k}(t) = \sum_{i=0}^{n-m} B_i^{n-m} w_{k+i}. \tag{5}$$

If both m and k are single digit integers, the comma between them is omitted, as in Q_{01}. Fig. 1 shows internal division points $\boldsymbol{Q}_{m,k}$ of a cubic polynomial Bézier curve at $t = 0.5$. The advantage of the $\boldsymbol{Q}_{m,k}$ expression is that if an expression is related to derivatives up to the m-th order, the characteristics of the expression can be derived by manipulating $\boldsymbol{Q}_{1,k}$, $\boldsymbol{Q}_{2,k}$, ..., and $\boldsymbol{Q}_{m,k}$ for curves of degree n. For example, the curvature of a polynomial Bézier curve is $\frac{|\boldsymbol{V}_3|}{|\boldsymbol{V}_1|^3}$ referring to Eq. (6), and the curvature of a rational Bézier curve is $\frac{nW(t)^3|\boldsymbol{V}_7|}{|\boldsymbol{V}_5|^3}$ referring to Eq. (8). For the proof, use $\kappa = \frac{|\dot{\mathbf{P}} \wedge \ddot{\mathbf{P}}|}{|\dot{\mathbf{P}}|^3}$ and refer to [9]. Saito et al. utilized these characteristics to demonstrate the degrees of curvature monotonicity evaluation functions for Béezier curves of degree n.

For a 3D polynomial curve of degree $n(\geq 3)$,

$$L_n(t) = (\boldsymbol{V}_4 \cdot \boldsymbol{V}_3)(\boldsymbol{V}_1 \cdot \boldsymbol{V}_1) - 3(\boldsymbol{V}_3 \cdot \boldsymbol{V}_3)(\boldsymbol{V}_1 \cdot \boldsymbol{V}_2), \tag{6}$$

where

$$\begin{aligned}
\boldsymbol{V}_1 &= n\ (\boldsymbol{Q}_{11} - \boldsymbol{Q}_{10}),\\
\boldsymbol{V}_2 &= n(n-1)\ (\boldsymbol{Q}_{22} - 2\boldsymbol{Q}_{21} + \boldsymbol{Q}_{20}),\\
\boldsymbol{V}_3 &= n^2(n-1)\ (\boldsymbol{Q}_{20} \wedge \boldsymbol{Q}_{21} + \boldsymbol{Q}_{22} \wedge \boldsymbol{Q}_{20} + \boldsymbol{Q}_{21} \wedge \boldsymbol{Q}_{22}),\\
\boldsymbol{V}_4 &= n^2(n-1)(n-2)\\
&\quad ((1-t)(\boldsymbol{Q}_{31} - \boldsymbol{Q}_{30}) \wedge (2\boldsymbol{Q}_{31} - 3\boldsymbol{Q}_{32} + \boldsymbol{Q}_{33})\\
&\quad + \quad t \quad (\boldsymbol{Q}_{30} - 3\boldsymbol{Q}_{31} + 2\boldsymbol{Q}_{32}) \wedge (\boldsymbol{Q}_{33} - \boldsymbol{Q}_{32})),
\end{aligned}$$

and the curvature monotonicity can be evaluated with the degree $6n - 11$ function $L_n(t)$.

For a 3D rational Bézier curve of degree $n(\geq 3)$,

$$L_n(t) = \frac{l_n(t)}{(W(t))^{11}}, \tag{7}$$

where

$$
\begin{aligned}
l_n(t) &= n(((\boldsymbol{V}_5 \wedge \boldsymbol{V}_6) \cdot \boldsymbol{V}_7)(\boldsymbol{V}_5 \cdot \boldsymbol{V}_5) + 3(\boldsymbol{V}_7 \cdot \boldsymbol{V}_7)(\boldsymbol{V}_8 \cdot \boldsymbol{V}_5)), \\
\boldsymbol{V}_5 &= n\ (W_{10}\boldsymbol{Q}_{11} - W_{11}\boldsymbol{Q}_{10}), \\
\boldsymbol{V}_6 &= n(n-1)(n-2) \\
&\qquad (W_{30}\boldsymbol{Q}_{33} - 3W_{31}\boldsymbol{Q}_{32} + 3W_{32}\boldsymbol{Q}_{31} - W_{33}\boldsymbol{Q}_{30}), \\
\boldsymbol{V}_7 &= n(n-1) \\
&\qquad (W_{22}(\boldsymbol{Q}_{20} \wedge \boldsymbol{Q}_{21}) + W_{21}(\boldsymbol{Q}_{22} \wedge \boldsymbol{Q}_{20}) + W_{20}(\boldsymbol{Q}_{21} \wedge \boldsymbol{Q}_{22})), \\
\boldsymbol{V}_8 &= n^2(n-1)\ ((1-t)(2W_{11}(W_{20}\boldsymbol{Q}_{21} - W_{21}\boldsymbol{Q}_{20}) \\
&\qquad\qquad - W_{10}(W_{20}\boldsymbol{Q}_{22} - W_{22}\boldsymbol{Q}_{20})) \\
&\qquad + \quad t \quad (W_{11}(W_{20}\boldsymbol{Q}_{22} - W_{22}\boldsymbol{Q}_{20}) \\
&\qquad\qquad - 2W_{10}(W_{21}\boldsymbol{Q}_{22} - W_{22}\boldsymbol{Q}_{21})),
\end{aligned}
\tag{8}
$$

and the curvature monotonicity can be evaluated with the degree $11n - 18$ function $l_n(t)$.

In 3D polynomial or rational curves, $L_n(t)$ or $l_n(t)$ can be represented as a polynomial of degree n_c in Bernstein form:

$$
\lambda(t) = \sum_{j=0}^{n_c} B_j^{n_c}(t)\xi_j. \tag{9}
$$

For 3D polynomial curves, $n_c = 6n-11$ and $\lambda(t)$ correspond to Eq. (6). For 3D rational curves, $n_c = 11n-18$ and $\lambda(t)$ correspond to Eq. (8). Since both $L_n(t)$ and $l_n(t)$ are represented in the form of $\lambda(t)$, we use $\lambda(t)$ instead of $L_n(t)$ or $l_n(t)$.

The curvature monotonicity can be evaluated by checking if $\lambda(t)$ changes its sign or not within $t \in [0, 1]$. We refer to the condition described below as the exact curvature monotonicity condition, or simply *the exact condition*.

$$
\lambda(t) \leq 0 \quad \text{or} \quad \lambda(t) \geq 0 \quad \text{for } t \in [0, 1] \tag{10}
$$

If $\lambda(t) \leq 0$, the curvature is monotonically decreasing. Conversely, if $\lambda(t) \geq 0$, the curvature is monotonically increasing. We refer to the following condition as the *sufficient condition*.

$$
\xi_j \leq 0\ (0 \leq i \leq n_c) \quad \text{or} \quad \xi_j \geq 0\ (0 \leq i \leq n_c) \tag{11}
$$

From the convex hull property, it is evident if the sufficient condition is satisfied, the exact condition is also satisfied. Note that the exact condition may be satisfied even if ξ_j have different signs. We visualize the curvature monotonicity regions of 3D curves using the sufficient conditions or the exact conditions.

When computing ξ_j in Eq. (9) by utilizing Eq. (6) or Eq. (8), Bernstein multiplications are required. These multiplications include the dot product and the vector product of two vector-valued polynomials, as well as the multiplication of a vector-valued polynomial by a scalar-valued polynomial, with all polynomials represented in Bernstein form. When we perform Bernstein multiplications in the fragment shader, the scaled Bernstein [10] is especially useful since the binomial coefficients are only necessary at the beginning and the end of computing ξ_j. An example Mathematica code for computing the curvature monotonicity evaluation function for 2D polynomial Bézier curves is available in [22]. The code can be easily modified for 3D polynomial and rational Bézier curves.

3 VISUALIZATION OF CURVATURE MONOTONICITY REGIONS

We visualize the curvature monotonicity regions of 3D polynomial or rational Bézier curves with cubic degree or higher. Since quadratic Bézier curves are planar, they reduce to the work of [11, 5].

For 3D curves, the curvature monotonicity region exists in three-dimensional space. The curvature monotonicity region for a control point $\boldsymbol{p}_k$ is the set of 3D points where the curvature varies monotonically. We propose two methods for visualizing this region: (1) In the 2D visualization, the curvature monotonicity region is visualized by showing its intersection with a constant-depth plane passing through the control point of interest. (2) Directly rendering the curvature monotonicity region in 3D.

We implemented our algorithms using C++ and OpenGL. Our program supports polynomial and rational Bézier curves of arbitrary degree, provided the fragment shader fits within the GPU memory. For both 2D and 3D visualizations, the degree-specific parts of the fragment shaders are dynamically modified in the application program, and the shaders are then recompiled.

3.1 2D Visualization

Let $\boldsymbol{p}_k$ be the control point for which we wish to visualize the curvature monotonicity region. Let T_p represent the matrix that transforms a point, represented in homogeneous coordinates, from the world coordinates to the screen coordinates. This matrix T_p is the product of the viewing, projection, and viewport transformations. By multiplying $\boldsymbol{p}_k$, expressed in homogeneous coordinates, by T_p, we can obtain the depth d of $\boldsymbol{p}_k$. We visualize the curvature monotonicity region of $\boldsymbol{p}_k$ as its intersection with a plane at depth d.

Similarly to [9, 24], we visualize the curvature monotonicity region by checking the curvature monotonicity for each pixel in a screen window using a GPU. Algorithm 1 outlines the proposed method for visualizing the curvature monotonicity region, implemented in a fragment shader using OpenGL.

Algorithm 1. Curvature monotonicity region of $\boldsymbol{p}_k$

Input:

The degree n of a Bézier curve and control points $\boldsymbol{p}_i$ (and weights w_i) $(0 \leq i \leq n)$.

The index k, indicating that the curvature monotonicity region for the control point $\boldsymbol{p}_k$ is to be visualized.

The matrix T_p that transforms a point, represented in homogeneous coordinates, from world coordinates to screen coordinates.

Output:

The color of each pixel.

Steps:

1. Using the screen coordinates and depth d, perform the inverse transformation T_p^{-1} to obtain the 3D coordinates $\boldsymbol{p}'_k$.
2. Replace $\boldsymbol{p}_k$ with $\boldsymbol{p}'_k$ and check the curvature monotonicity using the algorithm in [24].
3. If the curvature is monotonically varying, color the pixel with the user-specified color.

Fig. 2 shows the curvature monotonicity regions of a cubic Bézier curve with $\boldsymbol{p}_0 = [0\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_1 = [0.2\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_2 = [0.6\ 0.2\ 0.1]^{\mathrm{T}}$, $\boldsymbol{p}_3 = [0.8\ 1\ 0.2]$. In the figure, the curvature monotonicity regions are visualized simultaneously for all control points. To achieve this, Algorithm 1 is repeated for each control point within the fragment shader. Note that the depth of each region differs depending on the control point.

In Fig. 2(a), the curve is polynomial, since all weights are 1. Note that the depth of each region, corresponding to each control point, is different. In the upper left of the figure, the curvature plot is shown. Fig. 2(b) and (c) show the same curve from different viewpoints. Since we visualize the curvature monotonicity region as the intersection with a constant-depth plane, the shape of the region varies depending on the viewpoint. In Fig. 2(d), with the same viewpoint A, w_1 is set to 0.75, and the curvature remains monotonically

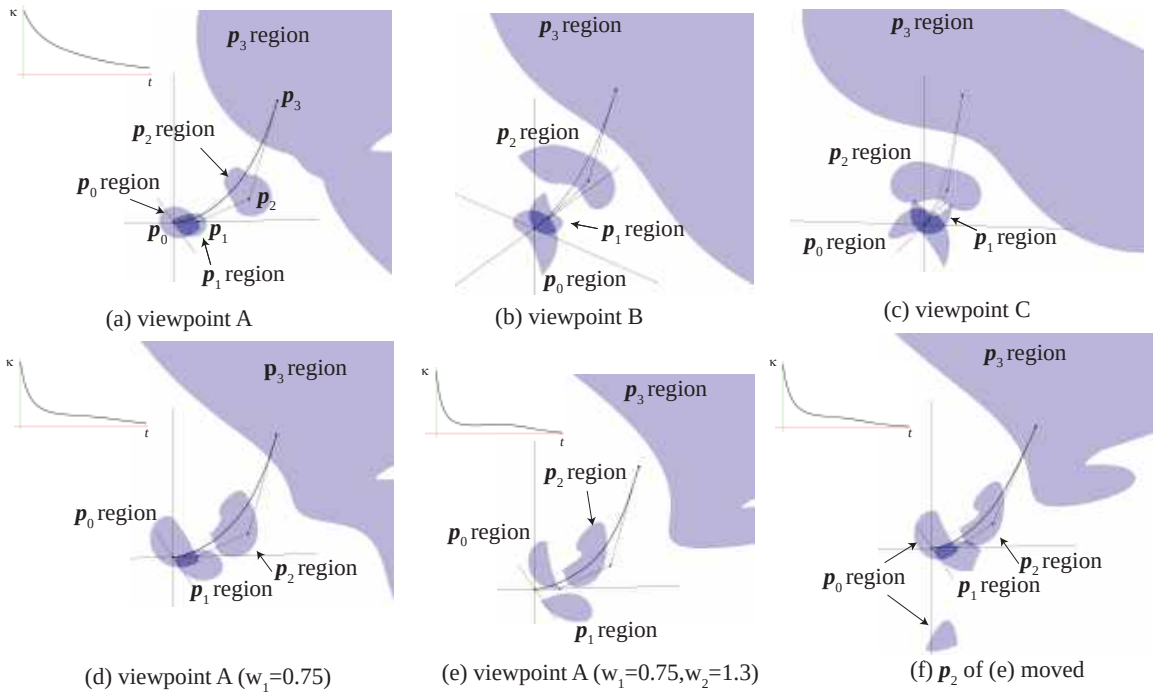


Figure 2: Curvature monotonicity region of cubic Bézier curves

decreasing. In Fig. 2(e), w_2 is set to 1.3, where it can be observed that the curvature is no longer monotonically decreasing. In Fig. 2(f), $\boldsymbol{p}_2$ is moved within the sufficient region so that the curvature becomes monotonically decreasing again.

3.2 3D Visualization

In this section, we introduce a method for visualizing the curvature monotonicity region in 3D. Our method is closely related to Singh et al.'s ray marching method [13] for visualizing implicit surfaces. However, we encounter the challenges of visualizing surfaces that cannot be represented by implicit surfaces. In [13], adaptive ray sampling is performed by normalizing the highest coefficient of the top-order term. This approach is not applicable in our case, as our surface cannot be represented as an implicit surface, making it impossible to determine the coefficient of the top order term. In addition, in [13], the gradient is used to determine the color of the pixel. However, computing the gradient on our surface is not straightforward. Therefore, we need to adopt a different method for computing the normal of the surface.

Our algorithm for visualizing the 3D curvature monotonicity region consists of the following two steps. The first step (Algorithm 2) uses the ray marching method to find the 3D coordinates where the curvature monotonicity changes and records these coordinates in an OpenGL frame buffer object. The alpha channel is used to indicate whether the surface of the region is intersected from the outside or inside. If the alpha channel is set to 1, it indicates that the curvature changes from non-monotonically varying to monotonically varying, meaning the ray intersects the surface from the outside. If the alpha channel is set to 0, it indicates the ray intersects from the inside. If the alpha channel is -1, it means the ray does not intersect the surface.

Algorithm 2. Step 1: Ray marching

Input:

The degree n of a Bézier curve, along with its control points $\boldsymbol{p}_i$ (and weights w_i for a rational curve), where $0 \leq i \leq n$.

The index k, indicating that the curvature monotonicity region for the control point $\boldsymbol{p}_k$ is to be visualized.

The matrix T_p that transforms a point, represented in homogeneous coordinates, from world coordinates to screen coordinates.

The ray marching step $rmStep$.

The coordinates of the viewpoint $\boldsymbol{p}_v$.

The information of the bounding box (or sphere).

Output:

OpenGL frame buffer object: Each pixel is represented as (r, b, g, a), where r, b and g correspond to the coordinates of x, y and z of the point where the ray intersects the surface, indicating that the curvature monotonicity changes at that point. If $a = -1$, the ray does not intersect the surface. If $a = 1$, the ray intersects the surface from the outside, indicating a transition from non-monotonically varying curvature to monotonically varying curvature. If $a = 0$, the ray intersects the surface from the inside.

Steps:

1. Using the screen coordinates and depth d, perform the inverse transformation T_p^{-1} to obtain the 3D coordinates $\boldsymbol{p}'_k$. Construct a ray from the viewpoint $\boldsymbol{p}_v$ to $\boldsymbol{p}'_k$.
2. Intersect the ray against the bounding box (or sphere) and compute the nearest and farthest intersections r_s, r_e. If the ray does not intersect, set the alpha channel of an OpenGL frame buffer object to -1 and terminate.
3. Set $r = r_s$ and $r_b = (r_e - r_s)/rmStep$.
4. Replace the coordinates of $\boldsymbol{p}_k$ with $(1-r)\boldsymbol{p}_v + r\boldsymbol{p}_p$, compute ξ_j using (6) for polynomial curves or Eq. (8) for rational curves, and check if the curvature is monotonically varying.
5. If the curvature is monotonically varying, set $c_s = 1$. Otherwise, set $c_s = 0$.
6. Set $r = r + r_b$.
7. while $r \leq r_e$ do
   - 7.1 Replace the coordinates of $\boldsymbol{p}_k$ with $(1-r)\boldsymbol{p}_v + r\boldsymbol{p}_p$, compute ξ_j using (6) or Eq. (8), and check if the curvature is monotonically varying. If the curvature is monotonically varying set $c_c = 1$. Otherwise, set $c_c = 0$.
   - 7.2 If $c_s \neq c_c$, perform the following steps: Set $r_m = r - 0.5r_b$ and record $(1-r_m)\boldsymbol{p}_v + r_m\boldsymbol{p}_p$ in an OpenGL frame buffer object. The value of c_c is also recorded in the alpha channel to indicate whether the ray intersects from the outside or inside. Exit the while loop.
   - 7.3 Set $r = r + r_b$.
8. If $r > r_e$, the ray does not intersect the surface. Set the alpha channel to -1.

In our actual implementation of Algorithm 2, we apply the following refinement process. Suppose an intersection is found in Step 7.2, meaning the intersection point lies between $r - r_b$ and r. To refine the intersection point, we update the parameters as follows: set $r_e = r$, $r_s = r_e - r_b$, $r = r_s$, and $rmStep = 10$. Then we repeat Steps 4 to 7.

In the second step (Algorithm 3), triangles are constructed using the coordinates of nearby pixels in the frame buffer object, and the averaged normal is computed. The pixel color is then determined based on this normal. Both Algorithm 2 and Algorithm 3 are executed in the fragment shader of the GPU.

Algorithm 3. Step 2: Shading using surface normals

Input:
OpenGL framebuffer object: the output of Algorithm 2.
Shading parameters required for determining the pixel color using the normal.

Output:
The color of each pixel.

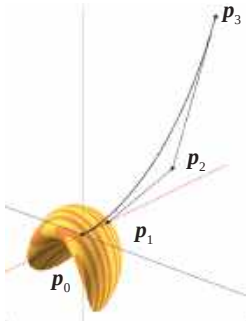
Steps:

1. If the alpha channel is -1, terminate processing.
2. Retrieve the 3D coordinates of the target pixel from the frame buffer object.
3. Identify neighboring pixels whose alpha channels are not -1 and retrieve the 3D coordinates.
4. Construct up to 8 triangles. If no triangle is constructed, terminate processing. Otherwise, compute the averaged normal.
5. Determine the pixel color based on the computed normal. If the ray intersects from the inside (i.e., if the alpha channel is 0), apply a darker shade.

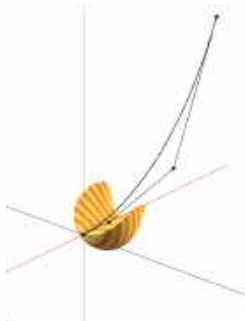
Figs. 3(a)-(d) show the 3D curvature monotonicity regions for each control point of a polynomial cubic Bézier curve with $\boldsymbol{p}_0 = [0\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_1 = [0.2\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_2 = [0.6\ 0.2\ 0.1]^{\mathrm{T}}$, and $\boldsymbol{p}_3 = [0.8\ 1\ 0.2]$. A zebra-like shading pattern based on x-coodinates is applied to enhance the perceptibility of the region's shape. Additionaly, the depth test is disabled when rendering the axes, control polygon, and curve to ensure their visibility at all times. Fig. 3(e)-(f) illustrates the 2D curvature monotonicity regions of $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{p}_3$. The 2D curvature monotonicity region is the intersection between the constant-depth plane of the control point of interest and the 3D region. Fig. 3(i) represents the curvature plot. In Figs. 3 (j) and (k), w_1 is set to 0.7, making the curve rational. The corresponding curvature plot is shown in Fig. 3(l). In Figs. 3 (m) and (n), $\boldsymbol{p}_3$ is repositioned as indicated. Since we moved $\boldsymbol{p}_3$ within its 2D curvature monotonicity region, the curvature is still monotonically varying, as verified in Fig. 3(o).

Fig. 4 shows the 3D and 2D curvature monotonicity regions of a polynomial quintic Bézier curve with $\boldsymbol{p}_0 = [0\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_1 = [0.1\ 0\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_2 = [0.25\ 0.1\ 0]^{\mathrm{T}}$, $\boldsymbol{p}_3 = [0.35\ 0.4\ 0]$, $\boldsymbol{p}_4 = [0.17\ 0.9\ 0]$, and $\boldsymbol{p}_5 = [0.35\ 1.2\ 0]$. While the curve itself lies in a 2D plane, its curvature monotonicity regions are 3D since the control points can be positioned in three-dimensional space. In Fig. 4(k), the visualization of the 3D region for $\boldsymbol{p}_5$ is shown from an internal viewpoint, hence dark shading is applied. Fig. 4(m) provides an alternative perspective of the same 3D region. Fig.4(o) displays the curvature plot of the quintic Bézier curve.

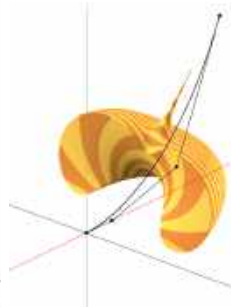
Table 1 presents an example of the computation time (in seconds) required to visualize the 3D curvature monotonicity region of each control point for cubic, rational cubic, quartic, rational quartic, and quintic Bézier curves. Due to GPU resource limitations, the 3D region of the rational quintic Bézier curve could not be generated. The computation time was measured using a Ryzen 9 5950X CPU with 64 GB of memory and an Nvidia GeForce RTX 4080 GPU with 16 GB of memory. Each computation was performed three times, and the average time was recorded. For the quintic Bézier curve, the control points are the same as in Fig. 4. For the cubic and quartic Bézier curve, only the first three or four control points were used. The rational cubic Bézier curve was obtained by setting w_1 to 1.1, while for the rational quartic Bézier curve, w_1 was set to 1.05. The ray marching step was set to 300, the bounding box size to ± 1 (or ± 2 for $\boldsymbol{p}_5$), and the window size to 1507×1462 pixels. The computation time varies depending on factors such as the size of the 3D region, the degree of the curve, the ray marching step, the bounding box size, and whether the curve is polynomial or rational.



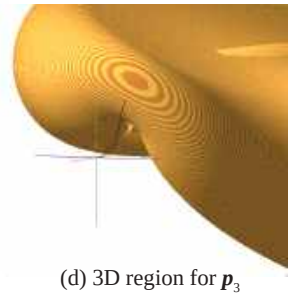
(a) 3D region for p_0



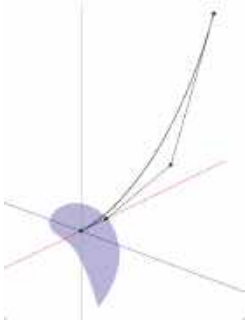
(b) 3D region for p_1



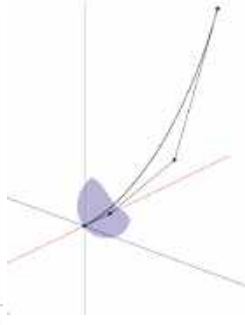
(c) 3D region for p_2



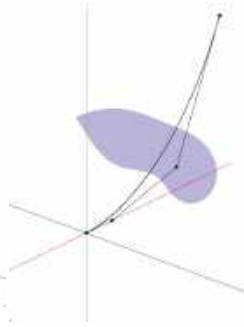
(d) 3D region for p_3



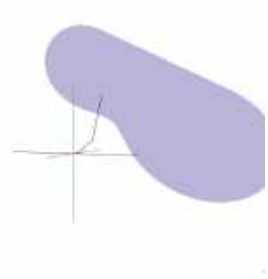
(e) 2D region for p_0



(f) 3D region for p_1

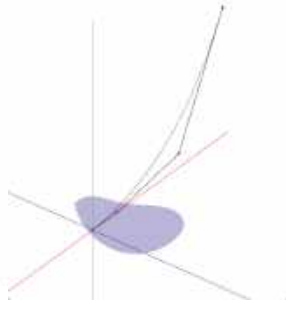


(g) 3D region for p_2

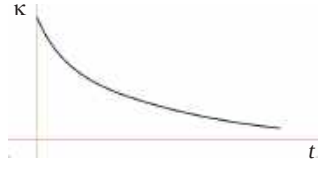


(h) 3D region for p_3

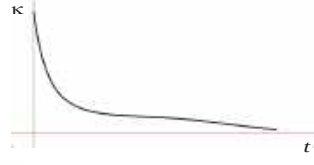
(j) 3D region for p_1 (w_1=0.7)



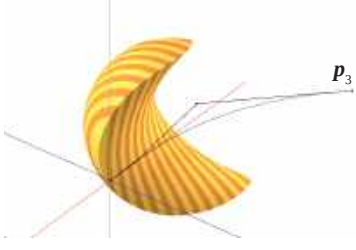
(k) 2D region for p_1 (w_1=0.7)



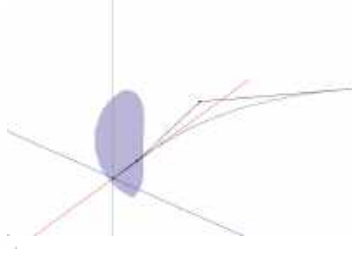
(i) curvature plot of the curve in (a)



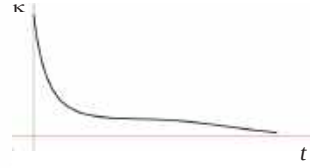
(l) curvature plot of the curve in (j)



(m) 3D region for p_1 (p_3 moved)

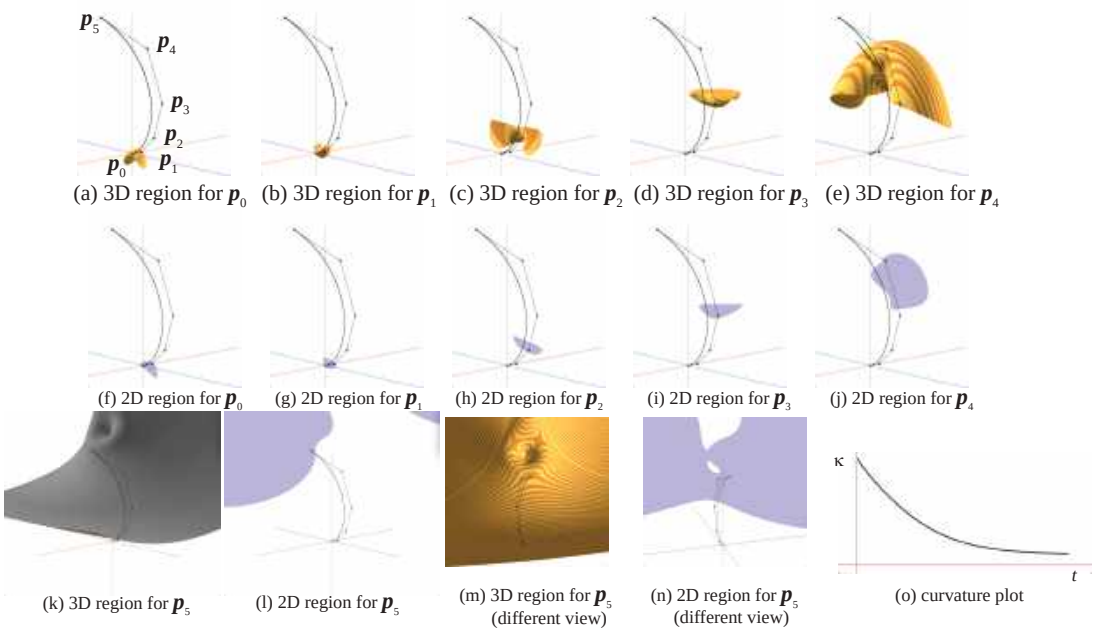


(n) 2D region for p_1 (p_3 moved)



(o) curvature plot of the curve in (m)

Figure 3: 3D and 2D curvature monotonicity regions of a cubic Bézier curve and its curvature plot



(a) 3D region for p_0 (b) 3D region for p_1 (c) 3D region for p_2 (d) 3D region for p_3 (e) 3D region for p_4

(f) 2D region for p_0 (g) 2D region for p_1 (h) 2D region for p_2 (i) 2D region for p_3 (j) 2D region for p_4

(k) 3D region for p_5 (l) 2D region for p_5 (m) 3D region for p_5 (different view) (n) 2D region for p_5 (different view) (o) curvature plot

Figure 4: 3D and 2D curvature monotonicity regions of a quintic Bézier curve and its curvature plot

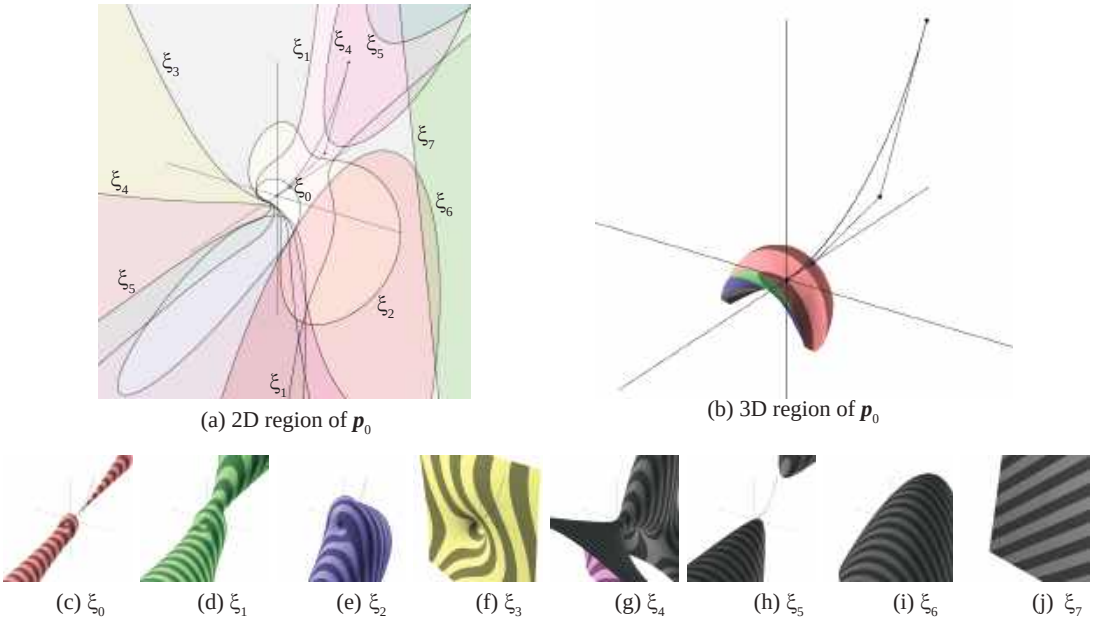
Table 1: Computation time in seconds for the 3D visualization of each control point.

	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$
cubic	0.18	0.24	0.06	0.2		
rational cubic	0.29	0.26	0.69	0.42		
quartic	0.24	0.26	0.24	0.34	0.77	
rational quartic	0.5	1.24	1.55	2.05	3.46	
quintic	0.4	0.39	0.54	0.76	1.47	1.03

3.3 Visualization of Sufficient Regions

Our program can also visualize the sufficient regions both in 2D and 3D. The sufficient region is the region that satisfies Eq. (11). Each ξ_j is the function of x_k, y_k, and z_k, which correspond to the coordinates of the control point $\boldsymbol{p}_k$. In other words, the sufficient region is the area where all ξ_j values are either negative or positive. In a 2D visualization, ξ_j appears as an algebraic curve, whereas in a 3D visualization, it forms an algebraic surface. The 2D visualization of the sufficient region is similar to that of 2D curves as in [23], except that the region is visualized on a constant-depth plane. Fig. 5(a) shows the 2D sufficient region of $\boldsymbol{p}_0$ for the same cubic Bézier curve as in Fig. 3(a). In Fig. 5(a), the sufficient region (white region) along with ξ_j is shown. Within this region, all ξ_j values are negative.

For the 3D visualization of a sufficient region, we use a single-step algorithm similar to Algorithm 2. However, unlike Algorithm 2, we identify a point where all ξ_j (or a specific ξ_j) become either positive or negative, and the approximate normal is computed using divided differences. Fig. 5(b) shows the 3D sufficient region of $\boldsymbol{p}_0$ of the same cubic Bézier curve as in Fig. 3(a). Figs. 5(c)-(j) illustrate each ξ_j individually. The



(a) 2D region of $\boldsymbol{p}_0$ (b) 3D region of $\boldsymbol{p}_0$

(c) ξ_0 (d) ξ_1 (e) ξ_2 (f) ξ_3 (g) ξ_4 (h) ξ_5 (i) ξ_6 (j) ξ_7

Figure 5: 2D and 3D sufficient regions of a cubic Bézier curve and implicit surfaces that bounds the 3D region

3D sufficient region is the intersection of all ξ_j, where all xi_j values are either positive or negative.

As described in [23], if a portion of ξ_0 or ξ_{n_c} forms the boundary of the sufficient region, it also becomes part of the boundary of the exact region. In Fig. 5(a), ξ_0 forms most of the boundary of the sufficient region; it also forms the boundary of the exact region. See Fig. 3(e). Similarly, in Fig. 5(b), ξ_0 forms most of the visible boundary of the sufficient region; it also forms the boundary of the exact region. See Fig. 3(a).

4 APPLICATIONS

As an application of our approach, we developed a 3D curve design tool. The tool is similar to the curve tool in Adobe Illustrator and the one described in [24], but it differs in that it supports the design of 3D curves and can visualize the curvature monotonicity region in 2D. In our tool, users can identify the region on a constant-depth plane where a control point maintains monotonically varying curvature. When a manipulation point, such as $\mathbf{P}_i$ in Fig. 6, is moved, it remains within a constant depth plane. If $\mathbf{P}_i$ is moved, $\mathbf{P}_{i-1}$ and $\mathbf{P}_{i+1}$ also move while maintaining their relative positions. When $\mathbf{P}_i$ stays within its purple region, the curvature monotonicity of the two segments connected to $\mathbf{P}_i$ is guaranteed. Similarly, if $\mathbf{P}_{i-1}$ or $\mathbf{P}_{i+1}$ moves within its respective orange or green region, the curvature monotonicity of the corresponding curve segment is also preserved.

Fig. 7(a) shows an arrow created using our curve design tool. Due to an optical illusion, the arrow tends to appear rightward-facing [14]. Each curve segment is designed so that its curvature varies monotonically, while the connection points are intentionally G^0. Since displaying the curvature comb for all segments would make the visualization complex, only the curvature combs of four segments are shown in Fig. 7(b). Figs. 7(c)-(f) show rendered images of the arrow after it has been converted into a solid object. In these images, the arrow is rotated 180 degrees clockwise. Despite originally pointing to the right, it still appears to do so after the rotation.

Fig. 8(a) shows an umbrella frame created using our tool. The curvature of each curve segment is designed

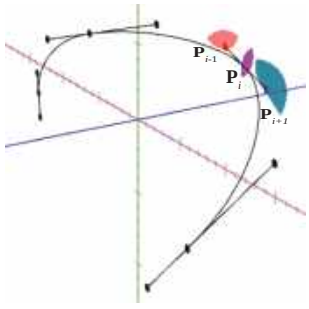


Figure 6: 3D curve design tool

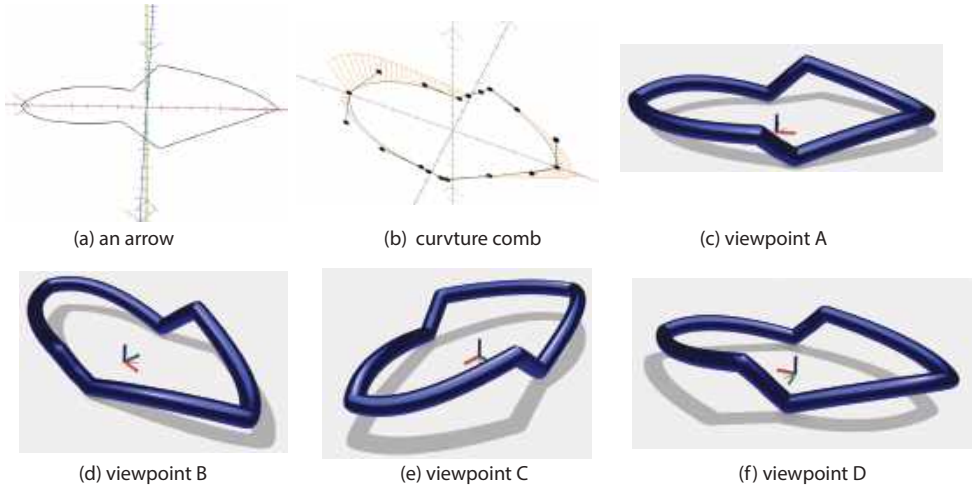


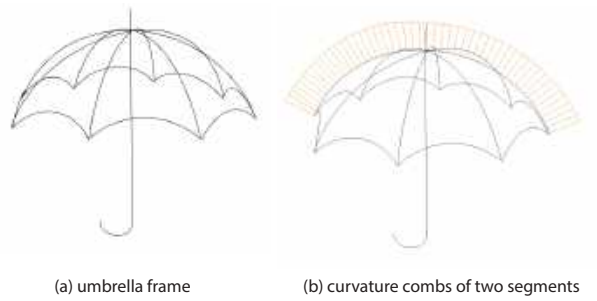
Figure 7: An arrow that tends to face rightward

to vary monotonically. In Fig. 8(b), the curvature combs of two segments are shown. Since the connection point of the two segments is G^0, the curvature combs overlap.

5 CONCLUSIONS

In this paper, we proposed a method for visualizing the curvature monotonicity regions of 3D polynomial and rational Bézier curves in both 2D and 3D. Prior to this work, the visualization of curvature monotonicity regions for 3D Bézier curves had not been explored. In our approach, we provide a novel method to visualize these regions, enabling better understanding and design of 3D curves. In the 2D visualization of the curvature monotonicity region, the region is displayed on the constant-depth plane of the control point of interest. The 2D visualization is real-time, as long as the degree of the curve is not high. Using the 2D visualization tool, we developed a curve design tool for 3D curves, allowing users to identify the region of a control point where the curvature varies monotonically. For the 3D visualization of the curvature monotonicity region, we proposed a two-step algorithm. The 3D visualization is almost interactive for 3D cubic Bézier curves; however, the computation time increases as the degree of the curve increases or when the curve becomes rational.

Future work includes a more detailed analysis of the curvature monotonicity region and improvements to the efficiency of 3D visualization. Currently, there is no direct application for visualizing the curvature



(a) umbrella frame (b) curvature combs of two segments

Figure 8: Umbrella frame

monotonicity region in 3D. Exploring potential applications will be another focus of future research.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number 24K07280. The 3D curve design tool, the 3D arrow in Fig. 7, and the umbrella frame in Fig. 8 were created by Hikaru Yasuda. We also thank the reviewers for their valuable comments and suggestions.

Norimasa Yoshida https://orcid.org/0000-0001-8889-0949
Takafumi Saito https://orcid.org/0000-0001-5831-596X

REFERENCES

[1] Dietz, B., D. A.and Piper: Interpolation with cubic spirals. Computer Aided Geometric Design, 21(2), 165–180, 2004. http://doi.org/10.1016/j.cagd.2003.09.002.

[2] Farin, G.: Curves and Surfaces for CAGD. Academic Press, 2001.

[3] Farin, G.: Class A Bézier curves. Computer Aided Geometric Design, 23(7), 573–581, 2006. http://doi.org/10.1016/j.cagd.2006.03.004.

[4] Floater, M.S.: Derivatives of rational Bézier curves. Computer Aided Geometric Design, 9(3), 161–174, 1992. http://doi.org/10.1016/0167-8396(92)90014-G.

[5] Frey, W.H.; Field, D.A.: Designing Bézier conic segments with monotone curvature. Computer Aided Geometric Design, 17(6), 457–483, 2000. http://doi.org/10.1016/S0167-8396(00)00011-X.

[6] Mineur, Y.; Lichah, T.; Castelain, J.M.; Giaume, H.: A shape controlled fitting method for Bézier curves. Computer Aided Geometric Design, 15(9), 879–891, 1998. http://doi.org/10.1016/S0167-8396(98)00025-9.

[7] Reimers, M.; Seland, J.: Ray casting algebraic surfaces using the frustum form. Computer Graphics Forum(Eurographics), 27(2), 361–370. http://doi.org/10.1111/j.1467-8659.2008.01133.x.

[8] Romani, L.; Viscardi, A.: Planar class A Bézier curves: The case of real eigenvalues. Computer Aided Geometric Design, 89, 2021. http://doi.org/10.1016/j.cagd.2021.102021.

[9] Saito, T.; Yoshida, N.: Curvature monotonicity evaluation functions on rational Bézier curves. Computers & Graphics, 114, 219–229, 2023. http://doi.org/10.1016/j.cag.2023.05.019.

[10] Sánchez-Reys, J.: Algebraic manipulation in the Bernstein form made simple via convolutions. Computer-Aided Design, 10(10), 959–967, 2003. http://doi.org/10.1016/S0010-4485(03)00021-6.

[11] Sapidis, N.S.; Frey, W.H.: Controlling the curvature of quadratic Bézier curve. Computer Aided Geometric Design, 9(2), 85–91, 1992. http://doi.org/10.1016/0167-8396(92)90008-D.

[12] Seland, J.S.; Dokken, T.: Real-time algebraic surface visualization in Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF. 163–183, 2007. http://doi.org/10.1007/978-3-540-68783-2_6.

[13] Singh, J.M.; Narayanan, P.J.: Real-time ray tracing of implicit surfaces on the GPU. IEEE Transaction on Visualization and Computer Graphics, 16(2), 2010. http://doi.org/10.1109/TVCG.2009.41.

[14] Sugihara, K.: Family tree of impossible objects created by optical illusion. In Proceedings of the Bridges between Mathematics and the Arts, 329–336, 2020.

[15] Tong, W.; Chen, M.: A sufficient condition for 3D typical curves. Computer Aided Geometric Design, 87, 2021. http://doi.org/10.1016/j.cagd.2021.101991.

[16] Wang, A.; He, C.; Zheng, J.; Zhao, G.: 3D class A Bézier curves with monotone curvature. Computer Aided Design, 159, 2023. http://doi.org/10.1016/j.cad.2023.103501.

[17] Wang, Y.; Zhao, B.; Zhang, L.; Xu, J.; Wang, K.; Wang, S.: Designing fair curves using monotone curvature pieces. Computer Aided Geometric Design, 21(5), 515–527, 2004. http://doi.org/10.1016/j.cagd.2004.04.001.

[18] Yoshida, N.; Fukuda, N.; Saito, T.: Logarithmic curvature and torsion graphs. In in Mathematical Methods for Curves and Surfaces 2008, LNCS 5862, 434–443, 2010. http://doi.org/10.1007/978-3-642-11620-9_28.

[19] Yoshida, N.; Fukuda, R.; Saito, T.: Interactive generation of 3D class A Bézier curve segments. Computer-Aided Design and Application, 7(2), 163–172, 2010. http://doi.org/10.3722/cadaps.2010.163-172.

[20] Yoshida, N.; Hiraiwa, T.; Saito, T.: Interactive control of planar class A Bézier curves using logarithmic curvature graphs. Computer-Aided Design and Applications, 5(1-4), 121–130, 2008. http://doi.org/10.3722/cadaps.2008.121-130.

[21] Yoshida, N.; Saito, T.: Quasi-aesthetic curves in rational cubic Bézier forms. Computer-Aided Design and Applications, 4(1-4), 477–486, 2007. http://doi.org/10.1080/16864360.2007.10738567.

[22] Yoshida, N.; Saito, T.: Curvature monotonicity evaluation function for 2D polynomial Bézier curves. the Notebook Archive, 2024. https://notebookarchive.org/2024-06-1xe104l.

[23] Yoshida, N.; Saito, T.: Curvature monotonicity regions of 2D polynomial and rational Bézier curves as the intersection of implicit regions. Computer-Aided Design and Applications, 2(1), 68–80, 2025. http://doi.org/10.14733/cadaps.2025.68-80.

[24] Yoshida, N.; Sakurai, S.; Yasuda, H.; Inoue, T.; Saito, T.: Visualization of the curvature monotonicity regions of polynomial curves and its application to curve design. Computer-Aided Design and Applications, 21(12), 75–87, 2024. http://doi.org/10.14733/cadaps.2024.75-87.