Tracking Point-Curve Critical Distances

Xianming Chen*, Elaine Cohen, Richard F. Riesenfeld

School of Computing, University of Utah

Abstract

This paper presents a novel approach to continuously and robustly tracking critical distances from a moving plane point $p \in \mathbb{R}^2$ to a parametrized piecewise smooth curve $\gamma(s)$ ($s \in \mathbb{R}$), with no restriction on curve continuity. Geometrically, critical distances are simply perpendicular distances from point p to curve γ at some finitely many curve points with corresponding finitely many parameter s's. We represent a single critical distance as a point $p_s = (p, s)$ in the so-called augmented parametric space $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$. The locus of all such points p_s, when p moves over the whole plane $\mathbb{R}^2$, forms an implicit surface $\mathcal{I}$ in the augmented parametric space. For a given perturbation vector $\delta_p \in \mathbb{R}^2$ of p, the critical distances, in most situations, are only evolved algorithmically by marching on the sectional curve $\mathcal{I}_\delta$, that is, the intersection of $\mathcal{I}$ with a vertical plane passing through both points p and $p + \delta_p$. In some situations, however, when the perturbation passes through a transition point, that is, a point on the evolute of γ, there will be a transition event when a pair of critical distances is either created or annihilated. The transition event is detected using the boundary volume tree of the evolute, and it is determined to be of either creation type or annihilation type by simply looking the sign of $\kappa\kappa'$ (κ is the curvature of γ), which only changes its sign at isolated points on the curve and is pre-computed; finally, the created pair of critical distances is computed using second order differential computation on $\mathcal{I}_\delta$. Extra transition events due to various curve discontinuities are also investigated. Our implementation assumes B-spline representation for the curve and has interactive speed even on a lower end laptop computer.

Keywords*: Critical distance, minimal distance, symbolic computation on NURBS, bounding volume tree, transition event, singularity.*

*contact author: xchen@cs.utah.edu

1 Introduction

Given a plane point p and a closed plane curve $\gamma(s) : \mathbb{R} \to \mathbb{R}^2$, a Euclidean distance function $g_1 : \mathbb{R} \to \mathbb{R}$ is defined as,

$$g_1 = |\gamma - p|.$$

Another common practice is to define a signed distance function,

$$g_2 = \begin{cases} -|\gamma - p| & \text{if } p \text{ is inside the curve} \\ |\gamma - p| & \text{if } p \text{ is outside the curve} \end{cases}$$

In either case, a **critical distance** occurs when the first order derivative of the considered function vanishes. Geometrically, a critical distance has a configuration where the line segment from p to the curve point, called foot point F, is perpendicular to the tangent to the curve γ at F. A critical distance is generally either locally minimal or maximal, but, under some singular situations, could be neither.

When $p = \gamma(s_0)$ for some $s_0 \in \mathbb{R}$, then neither of the above functions, in general, is well-behaved at $s = s_0$. Excluding this zero distance situation, either version of the distance functions just defined is equivalent to the following **squared distance function**,

$$f = (\gamma - p)^2, \tag{1.1}$$

in the sense that the two functions have exactly the same vanishing derivatives.

Hereafter in this paper, we formulate everything through the squared distance function, nonetheless to achieve results which are equally valid in the context of the Euclidean or the signed distance function.

Following the convention of [4], we call a distance from the plane point $\boldsymbol{p}$ to the curve **foot point** $\boldsymbol{\gamma}(s)$ an $\boldsymbol{\mathcal{A}_n}$ **distance**, if $f^{(n+1)}(s)$ is the first non-vanishing derivatives at s. An $\mathcal{A}_n$ distance is called a critical distance if $n > 0$, and it is further classified as **regular** if $n = 1$, or **degenerate with multiplicity** $\boldsymbol{n}$ if $n > 1$. Also, we will denote a critical distance from p to the curve point $\gamma(s)$ as (p, s),

and consequently regard it as a point in $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$ (cf. Fig. 1(b)).

Critical distances, especially when generalized to those between a space point and a surface, are of primary interest to many geometric computation including minimal distance computation (e.g., [10, 11]), collision detection (e.g. [16, 17]), and with applications in motion planning and haptic rendering. It also has close relation to medial axis transformation [2] and Voronoi diagrams (e.g., see [5] for the closed smooth curve case). In this paper, we consider two issues namely, computing all the topological changes to the critical distances, and evolving critical distances if there is no topological change. Notice that, to track the critical distances, it has to start with some given initial plane point position and *all* the corresponding critical distances. This is typically done by solving Eq. (3.1) (see Section 3) using some constraint solver as discussed in [24, 7]. We will not go into more details on this, and simply assume that the initialization (with no missing critical distances) is given.

In this paper, we are especially interested in the *topological change* of critical distances, called a **transition event**. Mathematically, this is related to singularity and catastrophe theories [1, 21, 12, 20]. In the computer community, most recently, [4] defines the extended curve evolute to serve as the transition set of critical distances on *piecewise smooth* curves, especially deriving *algebraically* all the unfolding formulas for degenerate critical distances. This paper deals with all the practical implementation issues including the robust and efficient detection of transition events; the totality of critical distances, for all moving points, is formulated as an implicit surface $\mathcal{I}$ in the augmented parametric space $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$, and subsequently, the evolution and transition of critical distances are performed using first and second order shape computation on $\mathcal{I}$, respectively.

2 Motivation

We will present a set of algorithms to exactly, continuously, and robustly track point-curve critical distances. While this is important on its own, we are mostly motivated by its future extension to the surface case, that is, tracking the point-surface or surface-surface critical distances, and our ultimate goal is the continuous distance tracking between two trimmed NURBS both under either rigid motion or more general deformation. The current state-of-the-art in either hap tics rendering or motion planning is mainly of discrete and approximate nature [8, 18, 6]. [9, 19] did work directly on continuous NURBS models, however, just like the situation of polygonal models, global minimal search still has to be conducted concurrently (or periodically for the consideration of computation cost, and with the risk of possible wrong haptic rendition).

The transition event of critical distances is important for two reasons namely *robustness* and *efficiency*. If every transition event is detected *robustly*, and the corresponding topological change of critical distances is computed subsequently, we are guaranteed of not missing any critical distances, and assured of the correct global minimum or maximum if that is required. On the other hand, the *robust* detection of transition events also allows us to get rid of the typically *expensive global searching* for the global extremum. Notice that the transition events have zero measure compared to that relating the totality of point movement, and consequently, most of the time, our continuous tracking algorithm works on efficient local marching and refining of critical distances, with occasional computation of transition events which either create new critical distances or destroy current critical distances. More specifically, our major point is,

> *The expensive global extremal distance searching is not necessary for motion planning or haptic rendering, provided that*
>
> 1. *there is a sophisticated mathematical formulation of transition set [4], that is, the set of all transition points where topological change to the critical distances is going to happen;*
> 2. *the topological change, or transition event, can be* robustly *detected and computed.*

For the point-curve case, the transition set is identified either with the evolute of the curve if the considered curve is piecewise smooth with at least C^2 continuity at its break points [12], or with the extended evolute if the piecewise smooth curve has its break points of at least C^0 continuity [4]. For the point-surface case, the transition set is the two focal surfaces [13, 20]. Definitely, more complex situations arise for the point-model case where the model consists of more than one trimmed NURBS surfaces with C^0 continuity between surfaces, and for the ultimate surface-surface and model-model cases. On the other hand, we believe that the basic idea of replacing the global extremal distance search with a robust transition detection and computation, has a straightforward extension to higher dimension, and we present the current work as a first step toward a much more ambitious goal.

The rest of the paper is organized as follows. Section 3 presents the problem formulation in the augmented parametric space $\mathbb{R}^3$. For a given perturbation δ_p of p, Section 4 performs the evolution of the critical distance by marching locally on the sectional curve $\mathcal{I}_\delta$ that is the intersection of $\mathcal{I}$ with a vertical plane passing both plane points p and $p + \delta_p$. An osculating circle based refinement algorithm is

presented in Section 5. Section 6 computes the newly created pair of critical distances by contouring the local osculating parabola to $\mathcal{I}_\delta$. The robust and efficient *detection* of transition events is investigated in Section 7 using bounding volume tree of the curve evolute, exploiting the fact that the evolute has B-spline representation. Section 8 presents a simple way to classify the transition event by looking the sign of $\kappa\kappa'$ (κ is the curvature of the curve), which changes only at isolated curve points and is pre-computed. To apply our approach to realistic curve models, Section 9 develops algorithms for extra transition events due to curve discontinuity. Section 10 discusses some specific implementation issues in order to achieve *robust* tracking. After examples in Section 11, the paper finally concludes in Section 12.

3 Implicit Surface Formulation in the Augmented Parametric Space

The condition for critical distances between point $p \in \mathbb{R}^2$ and plane curve $\gamma(s) \in \mathbb{R}^2$ is,

$$(\gamma(s) - p) \cdot \gamma(s)' = 0. \tag{3.1}$$

Regarding the LHS of Eq. (3.1) as a function of $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, the locus of all critical distances, as point (p, s)'s in $\mathbb{R}^3$, is the zero set of f. $\mathbb{R}^3 = \mathbb{R}^2 \times \mathbb{R}$ is called the **augmented parametric space**.

The Jacobian of f,

$$\mathcal{J} = (\nabla f) = \begin{pmatrix} -\gamma'[0] \\ -\gamma'[1] \\ (\gamma - p) \cdot \gamma'' + |\gamma'|^2 \end{pmatrix}, \tag{3.2}$$

always has full rank (i.e., 1) under the assumption that the curve is regular; thus, especially with a regular value of 0, the zero set of f is a 2-manifold in $\mathbb{R}^3$, denoted as $\mathcal{I}$ hereafter.

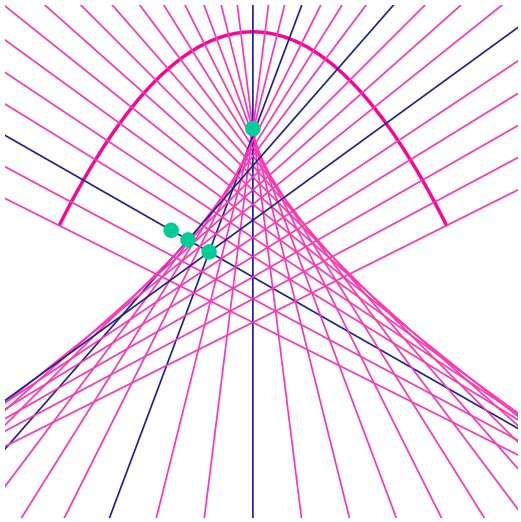
Remark 1 Geometrically, $\mathcal{I}$ is just the **curve normal surface** that is constructed by lifting each curve normal line, say that at $\gamma(s_0)$, along s-axis by a distance of s_0. Fig. 1(b) illustrates the normal surface of a parabola curve.

Notice that the normal to $\mathcal{I}$ actually has the same expression as the RHS of Eq. (3.2),

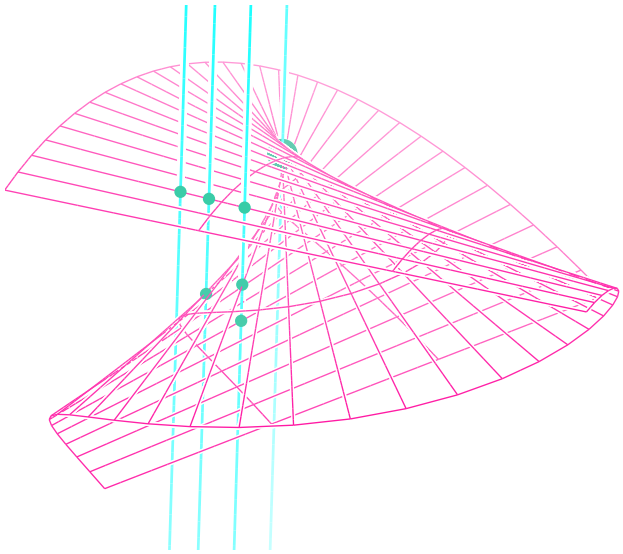
$$N_{\mathcal{I}} = \begin{pmatrix} -\gamma'[0] \\ -\gamma'[1] \\ (\gamma - p) \cdot \gamma'' + |\gamma'|^2 \end{pmatrix}, \tag{3.3}$$

or,

$$N_{\mathcal{I}} = -\gamma' + \mathfrak{D} e_s. \tag{3.4}$$



(a) Normal Bundle to a Parabola



(b) Lifted Normal Bundle

Figure 1. Curve Normal Surface $\mathcal{I}$ and Critical Distances

Also shown in (a) are 4 plane points (colored in green) with their corresponding critical distances (the blue normal lines). From left to right, they have one regular critical distance (CD), one regular plus another degenerate (with multiplicity 2) CD's, 3 regular CD's, and a degenerate (with multiplicity 3) CD to the curve, respectively. Lifting into the 3-space, the corresponding vertical lines pierce $\mathcal{I}$ once, pierce once and touch once (on the fold), pierce three times, and pierce once (at the cusp of the fold), respectively.

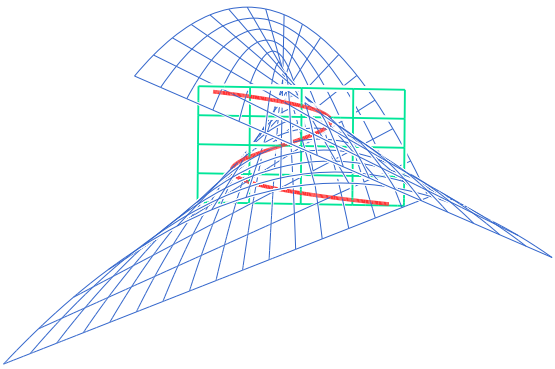


Figure 2. $\mathcal{I}_\delta$: Sectional Curve on Implicit Surface $\mathcal{I}$

where e_s is the unit vector along s-axis in $\mathbb{R}^3$, $\gamma' \in \mathbb{R}^2$ is regarded as $\gamma' \in \mathbb{R}^3$ in a natural way (i.e., the last component is 0), and $\mathfrak{D} = (\gamma - p) \cdot \gamma'' + |\gamma'|^2$ [1]

Finally, we recall a few identities which are used later in this paper.

$$\begin{aligned} e_s \times a &= a_\mathfrak{r}, \\ a \times b &= (a_\mathfrak{r} \cdot b) e_s, \\ (a_\mathfrak{r})_\mathfrak{r} &= -a, \\ a_\mathfrak{r} \cdot b_\mathfrak{r} &= a \cdot b \end{aligned} \tag{3.5}$$

where a and b are vectors in $\mathbb{R}^2$, and the subscript $\mathfrak{r}$ denotes 90 degree rotation around positive s-axis.

4 Evolution

Given any point perturbation δ_p of $p \in \mathbb{R}^2$, we want to evolve all the critical distances at p to those at $p' = p + \delta_p$. Geometrically, critical distance (p, s)'s, for a fixed p, are the intersection points of $\mathcal{I}$ with the vertical line passing through p, while the critical distances $(p + \delta_p, s + \delta_s)$'s are the intersection points of $\mathcal{I}$ with the vertical line passing through $p' = p + \delta_p$ (cf. Fig. 1(b)). Hence, the evolution is as simple as finding the correlation between δ_p and δ_s's. Considering *locally* only a single critical distance (p, s), the correlation is obviously related to the slope of the line connecting these two points (i.e., (p, s) and $(p+\delta_p, s+\delta_s)$), which is, in a limiting sense, simply the tangent vector.

[1] Notice that $\mathfrak{D} = |\gamma'|^2 \mathcal{D}$, cf. [4]

First we construct a curve on $\mathcal{I}$ that is the intersection of $\mathcal{I}$ with a vertical plane $\mathcal{P}$ passing both plane points p and $p + \delta_p$; let us call it **sectional curve** and denote it as $\mathcal{I}_\delta$ (Fig. 2).

By Eq. (3.4), and noticing that $\mathcal{P}$ has a normal $N_\mathcal{P} = (\delta_p)_\mathfrak{r}$, the tangent to $\mathcal{I}_\delta$ is,

$$\begin{aligned} T &= N_\mathcal{P} \times N_\mathcal{I} \\ &= (\delta_p)_\mathfrak{r} \times (-\gamma' + \mathfrak{D} e_s) \\ &= -(\delta_p)_\mathfrak{r} \times \gamma' + \mathfrak{D} (\delta_p)_\mathfrak{r} \times e_s \\ &= (\delta_p \cdot \gamma')\, e_s + \mathfrak{D}\, \delta_p, \end{aligned}$$

where we have used identities from Eq. (3.5). Introducing the notation

$$\delta = \delta_p \cdot \gamma',$$

the tangent vector of $\mathcal{I}_\delta$ is,

$$T = \delta\, e_s + \mathfrak{D}\, \delta_p, \tag{4.1}$$

Assuming $\mathfrak{D}$ non-vanishing, we can evolve the critical distance (p, s) by the following evolution algorithm.

Algorithm 1 Evolution

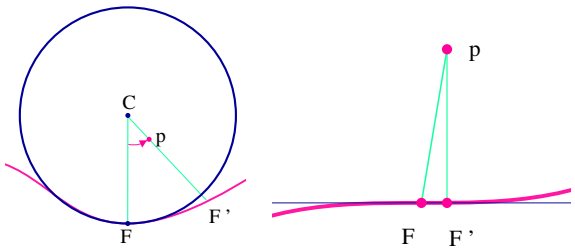
1. *Evaluate γ' and $\mathfrak{D}$ at s of the original critical distance (p, s),*
2. *Set the new critical distance to* $(p + \delta_p, s + \frac{\delta_p \cdot \gamma'}{\mathfrak{D}})$.

Remark 2 Notice that, in the algorithm, $p' = p + \delta_p$ is assumed to be very close to p; otherwise, pp' is cut into multiple marching steps, and Algorithm 1 and refinement algorithm (to be discussed in Section 5) are applied iteratively.

5 Refinement

Algorithm 1 evolves critical distances, essentially using tangent plane approximation to the implicit surface $\mathcal{I}$. Over time, there will be accumulated approximation error, especially when large evolution steps are used for efficiency. Therefore, for the sake of both accuracy and efficiency (relatively large evolution step can be used safely), we develop a curvature-based refinement algorithm in this section.

The basic idea is to approximate the local curve with its osculating circle (Fig. 3(a)). Suppose, for a plane point p, one of its approximate critical distances, under consideration, has a foot point F. However, p is not really on the normal line to F; instead, it deviates from the normal line by an angle $d\alpha = FCp$, where C is the curvature center corresponding to F. Recall that, for a plane curve, the



(a) Osculating Circle (b) Tangent Projection

Figure 3. Refine by Circle/Tangent Approx.

signed curvature is the change rate of $d\alpha$ with respect to arc length, i.e.,

$$\kappa = \frac{1}{|\gamma'|}\frac{d\alpha}{ds}; \tag{5.1}$$

hence,

$$ds = \frac{d\alpha}{\kappa|\gamma'|}. \tag{5.2}$$

Notice that, for the situation illustrated in Fig. 3(a), the angle FCp is close to 0 (enlarged a lot for visual effect). However, FCp could also be close to either π or 2π depending on p's actual location relative to C and the normal line, and consequently, π or 2π should be subtracted from FCp to get the actual $d\alpha$.

If F is, or is close to, a curvature zero point, Eq. (5.2) is numerically unstable to compute. A simple solution is to replace the osculating circle based refine algorithm temporarily (the condition will not stay next iteration) by a tangent based one (Fig. 3(b)),

$$ds = \frac{FF'}{|\gamma'|} = \frac{Fp\cdot\gamma'}{|\gamma'|^2}. \tag{5.3}$$

6 Transition

If $\mathfrak{D} = 0$, $\mathcal{I}_\delta$ is locally vertical at the considered $(p,s) \in \mathbb{R}^3$ (cf. Eq. (4.1)), and there is no way evolving (p,s) to the next approximate critical distance by following $\mathcal{I}_\delta$ tangentially. Mathematically, the locus of such points (p,s) form the fold singularity of projection of $\mathcal{I}$ on s-axis. For C^2 curve case, the projection of the fold is actually the curve evolute [12].

When the plane point moves across an evolute point, or **a transition point**, there will be a transition event, i.e. some structural change of the critical distances. At a transition point, the corresponding critical distance, is degenerate with multiplicity 2. Away from the folding point, the critical distance disappears completely in one direction, and unfolded into *a pair* of critical distances in the other direction. They are called an annihilation event and a creation event, respectively. Notice that the created pair of critical distances have to be of opposite types (one minimum, and the other maximum). [4] gives detailed algebraic derivations related to transition events.

In this Section, we initialize the created pair of critical distances by second order differential computation on $\mathcal{I}_\delta$, i.e, more specifically, by contouring the local osculating parabola to $\mathcal{I}_\delta$.

Eq. (4.1) gives the tangent vector field on the curve $\mathcal{I}_\delta$, and it allows us to compute the covariant derivative with respect to itself. At a singular point where $\mathfrak{D} = 0$, the curvature of $\mathcal{I}_\delta$ is (we write, generally, κN as $\boldsymbol{\kappa}$)

$$\begin{aligned}\boldsymbol{\kappa}_{\mathcal{I}_\delta} &= \frac{(T\times\nabla_T T)\times T}{T^4}\\ &= \frac{(\delta e_s \times \delta\frac{\partial T}{\partial s})\times \delta e_s}{(\delta)^4}\\ &= \frac{(e_s \times \frac{\partial(\delta e_s + \mathfrak{D}\ \delta_p)}{\partial s})\times e_s}{\delta}\\ &= \frac{\partial\mathfrak{D}}{\partial s}\ \frac{(e_s\times\delta_p)\times e_s}{\delta}\\ &= \frac{\mathfrak{D}'}{\delta}\,\delta_p\end{aligned} \tag{6.1}$$

where, as proved in Appendix M,

$$\mathfrak{D}' = -|\gamma'|^2\frac{\kappa'}{\kappa}. \tag{6.2}$$

Assume that p is originally *at a transition point* (i.e., on the evolute), with a degenerate critical distance (p,s) of multiplicity 2, and that p is now perturbed by δ_p. If $\frac{\mathfrak{D}'}{\delta} > 0$, then $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ has the same direction as δ_p by Eq. (6.1), or the perturbation direction is toward the curved side of $\mathcal{I}_\delta$; therefore, there will be a pair of new critical distances upon this perturbation (cf. Fig 2). Approximating the local curve of $\mathcal{I}_\delta$ by its osculating parabola, we have,

$$\begin{aligned}\delta_p &= \frac{1}{2}\boldsymbol{\kappa}_{\mathcal{I}_\delta}\delta_s^2\\ &= \frac{1}{2}\frac{\mathfrak{D}'}{\delta}\delta_s^2\,\delta_p,\end{aligned}$$

or

$$\frac{1}{2}\frac{\mathfrak{D}'}{\delta}\delta_s^2 = 1.$$

Therefore, the pair of critical distances are $(p+\delta_p, s+\delta_s)$ and $(p+\delta_p, s-\delta_s)$, where δ_s is,

$$\delta_s = \sqrt{\frac{2\delta}{\mathfrak{D}'}}. \tag{6.3}$$

On the other hand, if the perturbation is away from the curved side of $\mathcal{I}_\delta$, the original critical distance will disappear. The pair of critical distances to disappear or annihilate is trivial to find (from all the current critical distances), given the fact that it is the unique neighboring critical distances (p, s_0) and (p, s_1) with $s \in (s_0, s_1)$.

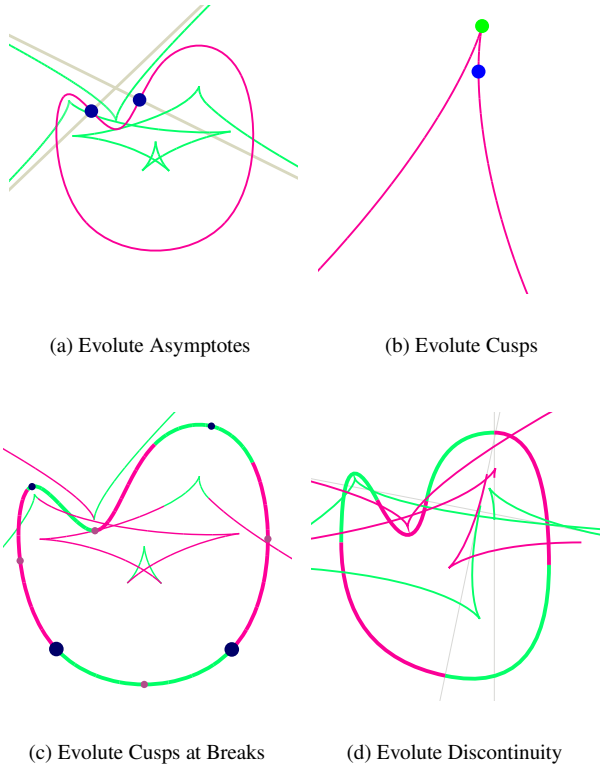
Remark 3 Notice that, for practical implementation of both creation and annihilation events, it rarely happens that the original plane point p is exactly on a transition point; Section 7 develops algorithms to check if the given perturbation will pass a transition point and if so, also compute the intersected transition point. Also, in the algorithm, δ_p is essentially the height where the osculating parabola is contoured, and thus for good approximation, it has to be very close to 0, or $p' = p+\delta_p$ very close to p; otherwise, p is perturbed to a closer point q first, and then the same procedure applies to qp' recursively. For our implementation, we set a pre-defined *fixed* contouring height, yet allow it to have adaptive resolution near evolute cusps (cf. Section 10).

Notice also that, at *isolated* evolute cusps, the corresponding critical distances have degeneracy of multiplicity 3. Refer to [4] for the details of the corresponding transition events. However, this could never exactly happen numerically, and there is really no need for special implementation of this type of transition. On the other hand, at a close neighbor of such a degenerate transition point, evolution and refine algorithms do have difficulty, and special care, as commented later in Section 10, has to be taken.

7 Detection of Transition Events

The detection of transition event is, in principle, as simple as a special curve-curve intersection problem [15, 14, 23, 22]. We are basically using the interval subdivision method [14]; our extra work, though, is to construct a bounding volume tree (BVT) from the axis aligned bounding boxes resulting from the interval subdivision so that, for most situations, intersection algorithm can be stopped at some very early stage. In this section, we will discuss some special issues related to interval subdivision on the *evolute* which has non-regular points, and present an algorithm for intersecting a line segment to a box *diagonal* (notice that one of the diagonal of the BVT leaf bounding box is supposed to be an approximation to the local evolute).

Just for the sake of discussion, we reserve, in this section, the word "intersection" for *line-diagonal* intersection, while using the word "hit" to refer the intersection of the line to the box *edges*.



(a) Evolute Asymptotes (b) Evolute Cusps

(c) Evolute Cusps at Breaks (d) Evolute Discontinuity

Figure 4. Special Issues for Interval Subdivision on Evolute

7.1 Construct Evolute BVT by Interval Subdivision

Interval subdivision requires a pre-processing of breaking the initial curve at any point where the tangent having one of its component vanished. However, the curve evolute is not a regular curve, and the pre-processing need to address two special issues.

Evolute Asymptotes At a curve inflection point, the corresponding evolute point is at infinity, and there is an asymptote to the evolute (see Fig 4(a)). Any infinite point poses difficulty for the interval method because the local evolute segment even does not have convex hull property. A simple way out of this is to add any infinite point as an extra splitting point for the pre-processing, and after all the subdivision, to discard the two bounding boxes adjacent to this infinite point.

Evolute Cusps An evolute segment will have a cusp corresponding to a vertex (where $\kappa = 0$) of the primary curve. At a cusp point, the evolute tangent direction turns by 180

degrees, and it is obvious that the considered interval is not appropriate for constructing a bounding box directly from its 2 Bézier end points; in other words, the evolute also need to be splitted at any *cusps* corresponding to curve *vertexes*. However, recalling that the evolute has 0 derivative at a cusp point [20, 4], i.e., both tangent components vanish, a cusp point will be regarded as a coordinate extremal point, and no special splitting is required. Fig. 4(b) shows an evolute cusp returned as a coordinate extremum, in addition to a genuine one.

An evolute cusp could also occur at a C^2 break point of a piecewise smooth curve, if the left and right limit curve curvatures have the opposite signs even though neither of them vanishing. The major difference, though, is that the evolute derivatives do not vanish at such cusp points and they can not be regarded as special coordinate extrema. Therefore, to make interval subdivision work, the curve should also be splitted at those C^2 break points that correspond to evolute cusps. Fig. 4(c) shows two evolute cusps corresponding to two curve C^2 break point (the larger dots), in addition to several other cusps corresponding to curve vertexes. Notice that, corresponding to the other type of C^2 break point, the evolute is actually G^1 in spite of its C^0 parametrization (see details in [4]), and curve splitting is not necessary. Because the evolute is discontinuous corresponding to a curve C^1 or C^0point (Fig. 4(d), the splitting, of course, should also be done there.

Putting all these together, the curve is splitted at all its break points, just for simple implementation. Notice that this is actually to convert the original B-spline curve into consecutive Bézier segments, which is a typical way to develop B-spline subdivision anyway.

Fig. 5 illustrates a BVT of the evolute of an ellipse. The actual BVT's used in our algorithms, though, have typically more subdivision levels.

Remark 4 For many applications, it may be desirable that the subdivision (after pre-processing the curve into monotone segments) be adaptive so that the finally subdivided curve segments all have similar arc lengths (instead of similar domain interval sizes). However, for transition detection and computation of critical distances, the simple uniform subdivision is not only easy to implement, but also has some advantage. Recall that, approaching an evolute cusp (corresponding to a curve vertex), the evolute curve velocity decreases to 0, and consequently, that the bounding boxes get smaller and smaller. Notice that this is exactly what we want, because a finer resolution is required near an evolute cusp to make the various algorithms, developed through out this paper, to work properly (cf. Section 10).

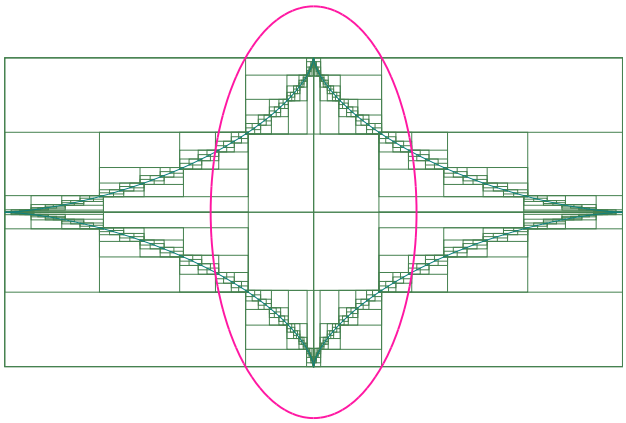


Figure 5. BVT of Ellipse Evolute

7.2 Line Hits Axis Aligned Box

The first stage of transition detection checks if a parametrized line $\mathcal{L}(t)$, with $\mathcal{L}(0) = p$ and $\mathcal{L}(1) = p'$, hits an axis aligned box. While this is essentially the typical ray tracing algorithm [25], more hit information is required for the next stage algorithm; specifically, if there is a hit, the algorithm should compute the following three pieces of information for both near and far hit points.

1. the hit edge;
2. the ratio of hit point with respect to pp', i.e., parameter t of hit point.
3. the ratio of hit point with respect to the hit edge.

If both hit points are outside of pp', the line *segment* pp' can not intersect the box diagonal in any case and therefore there will not be any transition event, and consequently, no need to go to next stage. Notice that, for the second ratio to make sense, a positive direction of any box edge has to be specified. In our implementation, an edge that is parallel to the i-th coordinate axis has the same positive direction as the positive axis if $pp'[i] \geq 0$, and opposite otherwise.

7.3 Line Segment Intersects Box Diagonal

A leaf node axis aligned box of the BVT, is constructed simply from the two end points of the control polygon resulting from interval subdivision. Through out this section, by "diagonal", we mean only the diagonal that connects these two end points. Notice that the diagonal segment is supposed to approximate the local curve, and has parameters for both of its ends that are *on* the evolute, and has positive direction that is the same to the curve.

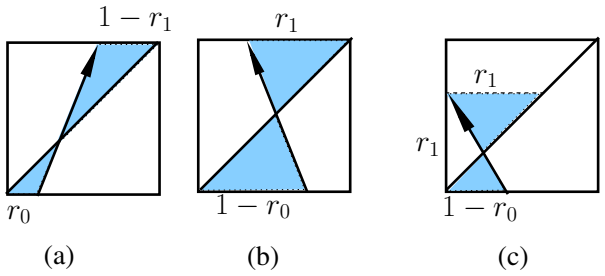


Figure 6. Generic Line Box-Diagonal Intersection

(a) and (b) show through-intersections, while (c) shows corner-intersection. On the other hand, (a) shows the situation that the diagonal vector has its two components the same relation w.r.t. those of pp' (i.e., either both the same sign or both the opposite sign), while (b) and (c) shows the other situation.

The second stage of the transition detection checks if the line segment intersects the diagonal of the hit box, and, if so, computes the ratio of the intersected point with respect to the two end points of the diagonal. The intersection point is an approximation to the real intersection point of pp' with the curve evolute, and its parameter is interpolated from those of the two diagonal ends.

Three generic situations of line-diagonal intersection are illustrated in Fig. 6. It is either a through-intersection, when the near and far hit points are on the opposite edges, or a corner-intersection, when they are on the neighboring edges. On the other hand, considering the relative orientation of the diagonal segment qq' with respect to the line segment pp', it is either that $qq'[i]$ has the same sign as that of $pp'[i]$ for *both* $i = 0$ and $i = 1$, or the sign relations are opposite for $i = 0$ and $i = 1$ (the boolean array $diag[2]$ in Algorithm 2 keeps this information).

The ratio of intersection point w.r.t. the diagonal is computed, based on this classification, and by constructing the axillary similar triangles (shaded in Fig. 6). See details in Algorithm 2.

8 Classification of Transition Types

If Algorithm 2 returns true, then the line segment pp' (i.e., the perturbation) does intersect the evolute and there is going to be a transition event. The transition event is of creation type, if the perturbation is toward curved side of $\mathcal{I}_\delta$, and of annihilation type otherwise (cf. Fig. 2). By Eq. (6.1), the perturbation direction is toward the curved side of the sectional curve $\mathcal{I}_\delta$, or $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ has the same direction as δ_p, if and only if $\mathfrak{D}'(\gamma' \cdot \delta_p) > 0$; hence, evaluation of $\mathfrak{D}'$ and $\gamma' \cdot \delta_p$ at the transition point will determine the exact transition type. However, by Eq. (6.2), $\mathfrak{D}'$ only changes sign

Algorithm 2. Line Segment Intersects Box Diagonal

Function: Intersect-Diag
Input:
$\mathbf{diag[2]}$ box diagonal direction w.r.t. $\mathbf{pp'}$
$(\mathbf{e_0}, \mathbf{t_0}, \mathbf{r_0})$ near hit point
$(\mathbf{e_1}, \mathbf{t_1}, \mathbf{r_1})$ far hit point
Output: on *true* return,
$\mathbf{t}$ ratio of intersected point w.r.t. $\mathbf{pp'}$
$\mathbf{r}$ ratio of intersected point w.r.t. diagonal
Return:
true if intersected (i.e. $t \in [0, 1]$), *false* otherwise
Begin
If $\mathbf{diag[0]} \neq \mathbf{diag[1]}$
$\mathbf{t} \Leftarrow (\mathbf{1} - \mathbf{r_0})/(\mathbf{1} - \mathbf{r_0} + \mathbf{r_1})$
$\mathbf{i} \Leftarrow is\_horizontal(\mathbf{e_0})\ ?\ \mathbf{0} : \mathbf{1}$
$\mathbf{diag[i]} \Leftarrow\ !\ \mathbf{diag[i]}$
Else
If corner cut situation **Return** *false*
$\mathbf{t} \Leftarrow \mathbf{r_0}/(\mathbf{r_0} + \mathbf{1} - \mathbf{r_1})$
$\mathbf{r} \Leftarrow \mathbf{t} \Leftarrow (\mathbf{1}-\mathbf{r})\mathbf{t_0}+\mathbf{r}\mathbf{t_1}$
If $\mathbf{t} \notin [\mathbf{0}, \mathbf{1}]$ **Return** *false*
If case (c) in Fig. 6 $\mathbf{r} \Leftarrow \mathbf{r} * \mathbf{r_1}$
If(! $\mathbf{diag[0]}$) $\mathbf{r} \Leftarrow \mathbf{1} - \mathbf{r}$
Return *true*
End

at isolated curve points where $\kappa\kappa'$ changes sign; therefore, the evaluation of $\mathfrak{D}'$, which involves the third order derivative, is not necessary at run time, provided that all the sign flipping points of $\kappa\kappa'$ are already pre-computed.

For piecewise smooth curve, κ could change sign across either zero curvature points or curve break points with $C^{(<2)}$ continuity. Similarly, κ' could change sign at either critical curvature points or curve break points with $C^{(<3)}$ continuity. Notice that κ may not change sign across a zero curvature point if it is a critical curvature point at the same time (a simple example is the point $(0, 0)$ on the curve $y = x^4$); similarly, κ' may not change sign across a critical curvature point if $\kappa'' = 0$ as well.

The following simple algorithm does the job without even evaluating κ' at any point, yet it is robust enough to deal with *higher order* singularity.

Algorithm 3 Pre-computing signs of $\kappa\kappa'$

1. *split the curve at all its zero curvature points, critical curvature points, and all break points with continuity* $C^{(<3)}$,

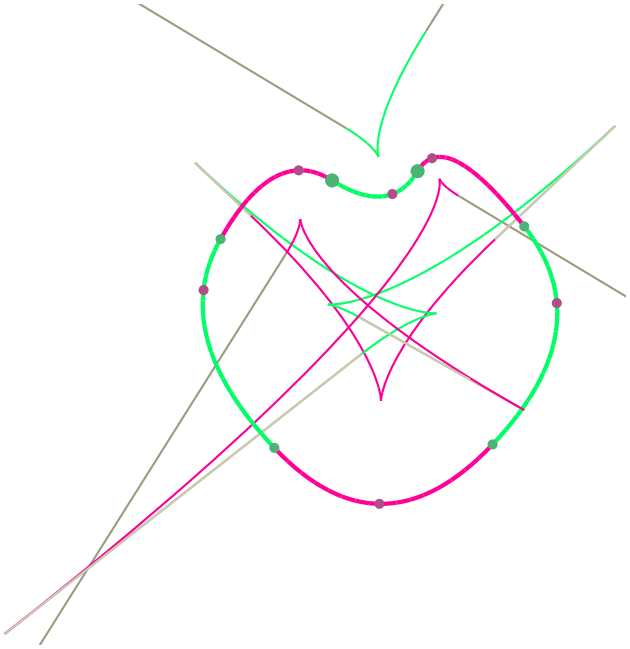


Figure 7. Flipping Points of $\kappa\kappa'$

Flipping points of $\kappa\kappa'$ on a quadratic B-spline curve of 6 segments with 6 C^1 break points. The two green larger filled circles denote curvature sign change at 2 inflexion points, and the rest dots denote curvature derivative sign change, 4 of which are at break points, and the rest of which are at critical curvature points. Also shown are the extended evolute (see [4] for details) with cusps and asymptotes corresponding to these flipping points.

2. *evaluate only* curvature *at all splitted points. The evaluation may be not necessary, if the point is zero curvature point; and it may have to be performed twice for left and right limit evaluations, for a $C^{(<2)}$ break point.*

3. *For each segment, tag the sign of κ the same as that of any of its non zero curvature end point, and the sign of κ' the same as the difference of κ at its two end points.*

The algorithm requires as input the zero and critical curvature points, which are computed using NURBS symbolic computation, with degree reduction strategy detailed out in [3]. Fig. 7 shows all the flipping points of $\kappa\kappa'$ of a quadratic B-spline curve.

9 Extra Transition Events at Curve Break Points

In this section, we make extension to our algorithms so that critical distances can be tracked across curve break points of at least C^0 continuity. Notice that the corresponding $C^{(-1)}$ situation is not any more difficult at all, and is omitted here under the consideration that any curve, as a boundary to some 2D shape, has to be closed.

9.1 C^2 Break Points

First, let us observe that, by Eq. (4.1), Eq. (5.2) and Eq. (6.3), all the algorithms developed so far work fine so long as the curve is C^3. However, the requirement can be relaxed to C^2. A C^2 point of the curve corresponds to a C^1 point on the implicit surface $\mathcal{I}$, and thus it does not affect evolution algorithm at all. On the other hand, it does affect the transition computation based on $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$, because $\boldsymbol{\kappa}_{\mathcal{I}_\delta}$ is C^{-1} by Eq. (6.1). However, due to numerical error, or by ϵ-perturbation intentionally, no transition computation can occur *exactly* and *exactly* at *isolated* curvature centers corresponding to isolated C^2 break points. On the other hand, if the dynamic of critical distances are sought *specifically* at those isolated transition points, a simple left limit and right limit evaluations of Eq. (6.3) suffice to get around this discontinuity problem.

9.2 C^1 Break Points

Usually only one point of any lifted normal line is on the fold of $\mathcal{I}$, and the projection of that point to $\mathbb{R}^2$ is on the evolute, or it is the curvature center (cf. Fig. 1(b)). However, if the considered normal line corresponds to a C^1 curve point, there will be a whole *segment* of it being on the fold of $\mathcal{I}$, and there will be a creation or an annihilation of a pair of new critical distances when the perturbation crosses any point of that segment. The normal line *segment*, serving as extra transition points, is either the line segment connecting the two (left limit and right limit) curvature centers, or the compliment of it with respect to the whole normal line (see [4] for more details). The important thing to note, though, is that the apparent *transition* event is actually because of two *evolution* events, one performed on the left segment and the other performed on the right segment. Therefore, the following simple algorithm will compute the (apparent) transition event or evolution event at a point (p, s) where γ is C^1 at s, given a perturbation of δ_p.

Algorithm 4 Transition/Evolution at C^1 Break

1. *If $(\delta_s)_l = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_l} < 0$, $(p+\delta_p, s+(\delta_s)_l)$ is a perturbed critical distance.*

2. *If $(\delta_s)_r = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_r} > 0$, $(p+\delta_p, s+(\delta_s)_r)$ is a perturbed critical distance.*

A creation/evolution/annihilation event happens if two/one/none perturbed distances are returned from the algorithm. Notice that, just like the transition event computation we did in Section 6, the point p in the algorithm is actually the intersection point of an original perturbation δ_{p0} with the considered normal line (see Section 9.4 below).

Remark 5 Algorithm 4 does not work if p is on either of the two curvature centers, because in that case, there is a *genuine* transition event. However, this situation is not generic, and an ϵ-perturbation will allow the algorithm work normally.

9.3 C^0 Break Points

At a C^0 curve break point, there are two normal lines, and each of the lifted ones has some segment on a fold to the implicit surface $\mathcal{I}$. We could have done transition computation directly for C^0 break points. However, as suggested by [4], a C^0 break point can be converted into two (collapsed) C^1 break points by inserting an arc with 0-radius in between the two folded break points. The arc has tangent continuity at its both ends, and has positive (negative) infinite curvature if the left and right end tangents form a right (left) hand rotation. Notice that this is essentially assigning a whole span of normal lines to a C^0 point, generated by right (left) rotating the left limit normal to the right limit normal; consequently, there will be an extra critical point if the plane point is on any of these normal lines. Introducing extra critical distances corresponding to C^0 points has the following benefits.

1. The ordered list of critical distances always comes in pairs of opposite types, which means, among others, a simple and unified annihilation algorithm regardless of the specific continuity including C^0.
2. Instead of tracking local minimal and maximal critical distances, now we are tracking local *extremal* distances.
3. Minimal code redundancy for C^0 situation is achieved.

Notice that, however, there are some implementation details arising because of folding a C^0 break points into two C^1 ones. We comment only on one important issue, i.e. the condition of $(\delta_s)_l = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_l} < 0$ (and the other similar one) in Algorithm 4. Recall that the inserted 0-radius arc has infinite curvature, or equivalently infinite norm for the second order derivative; therefore, when evaluated at either of the two arc end points,, $\mathfrak{D}$ in Algorithm 4 is (cf. Eq. (3.2)),

$$\begin{aligned}\mathfrak{D} &= (\gamma - p) \cdot \gamma'' + |\gamma'|^2 \\ &= (\gamma - p) \cdot \gamma''\end{aligned}$$

which evaluates to either positive or negative infinity, depending on the actually direction of γ'', which is $\boldsymbol{\gamma}'_{\mathbf{t}}$ ($-\boldsymbol{\gamma}'_{\mathbf{t}}$) if the curvature of the arc is positive (negative) infinity. However, the sign information disappears once $(\delta_s)_l = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_l}$ is evaluated to 0. Therefore, to make Algorithm 4 work both for real C^1 break points and C^1 break points converted from C^0 ones, the condition of $(\delta_s)_l = \frac{\delta_p \cdot \gamma'}{\mathfrak{D}_l} < 0$ should be changed into $(\delta_p \cdot \gamma')\mathfrak{D}_l < 0$. Similar argument applies to the other condition in the algorithm.

9.4 Detection of Extra Transition Events

Compared to the BVT approach for detecting transition events related to the curve evolute, this is really as trivial as intersecting the perturbation line segment with the normal lines. We simply take any intersection as a potential transition event, which may later turn out to be just an evolution event by Algorithm 4.

10 Some Implementation Comments

In this section, we make some comments on implementation details that are essential to a *robust* critical distance tracking.

No gap between two adjacent leaf bounding boxes of the evolute This has to be guaranteed by an appropriate data structure for constructing BVT of the evolute; otherwise, there would be missing transition events when the plane point is perturbed exactly across the tiny gap in between two adjacent leaf bounding boxes.

Multiple transition events performed in order When tracking the original critical distances at p to those at p', multiple transition events my be returned from the detection algorithms as detailed out in Section 7 and Section 9.4. All the transition events have to be performed *in order*, because a later transition event may depend on the result from an earlier one. The ordering can be simply done based on the returned t (ratio of transition point w.r.t. pp') values from Algorithm 2.

Special algorithms around extended evolute cusps Cusps of extended evolute are special transition points that pose difficulty for evolution and refinement algorithms (when applied in their close neighbors). There are a few related issues here.

1. The cusps have to be detected first. For curves of at least C^1 continuity, Algorithm 3 actually also gives

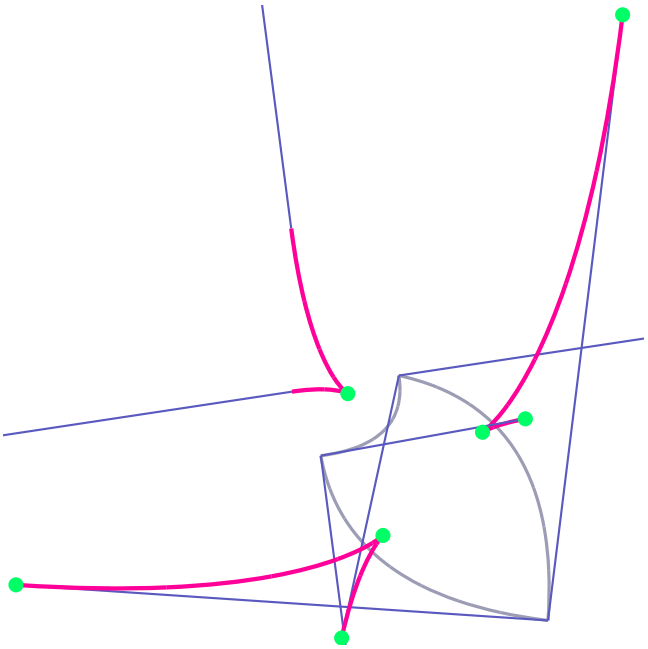


Figure 8. Cusps of the extended evolute

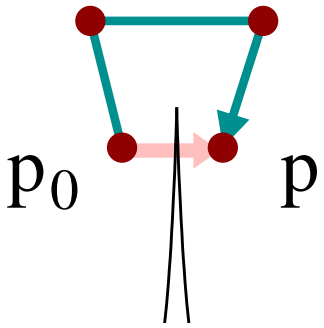


Figure 9. Alternative perturbation route around a cusp

all the flipping points of κ and κ', respectively; consequently, the cusp corresponding to a flipping κ' can be determined appropriately as either pointing *away* or *toward* to the local curve point, and consequently, the further or closer curvature center is the cusp point (for at least C^2 continuity, the cusp is simply the *unique* curvature center). On the other hand, for curves with C^0 continuity, things are a little different. Corresponding to each C^0 curve break point, there might be 0, 1 or 2 extended evolute cusps. Again, the implementation is based on the strategy of conversion to two C^1 breaks (cf. Section 9.3 and more details in [4]). Fig. 8 shows all the cusps of a C^0 B-spline curve.

2. Whenever the plane point p is detected to perturbed *into* and *stays* somewhere p' extremely close to some cusp point, it may happen that the later refinement algorithm (after the creation and evolution) has difficulty to converge. If this is the case, the plane point can be approximated to be *on* the cusp, and consequently degenerate critical distances can be assumed [2]. Furthermore, if the plane point is perturbed again later to p'', the new perturbation $p'p''$ is merged with the earlier pp' to make a single one pp'', and algorithms apply accordingly.

3. Whenever the plane point is detected to perturbed *into* a cusp and then *out* the same cusp, we replace the perturbation by an alternative yet equivalent route that goes around the cusp point, as illustrated in Fig. 9 [3]

Adaptive tracking There are two important control parameters to our tracking algorithm namely the marching step size for the evolution algorithm (cf. Remark 2), and the contouring height of the osculating parabola to the sectional curve $\mathcal{I}_\delta$ (cf. Remark 3). If the perturbation is detected to be inside a cusp area as detailed out in the previous paragraph, both control parameters can be adaptively refined based on the local approximate size of the cusp, which we know from the two intersection point of the perturbation pp' with the two cusp wings.

11 Examples

Fig. 10 gives an example of continuously tracking critical distances on a cubic B-spline curve, with 6 snapshots taken from the animation. The plane point is shown in dark square box, and foot points are shown in filled circles, while the corresponding points on the evolute (light gray colored) are shown in non-filled circles. There are 5 transition events occurred at some point between each pair of neighboring snapshots. The first transition annihilates a pair of critical distance, while the last one is actually two transition events, each of which annihilates a pair of critical distances. All the rest transitions create a pair of critical distances. The pair to be annihilated is shown in boxes, while the created pair in larger filled circles.

Fig. 11 shows the extremal distance tracking on a C^0 B-spline curve. Only part of the extended evolute curve is shown in very light color (cf. Fig. 8). Notice that Algorithm 4 will find out *automatically* if there is a transition

[2] However, we have not yet encountered such difficulty with all the demo programs we have run so far.

[3] Notice again, that this is only for robustness consideration – in most cases, our algorithms do the creation, evolution, and annihilation properly using the original perturbation.

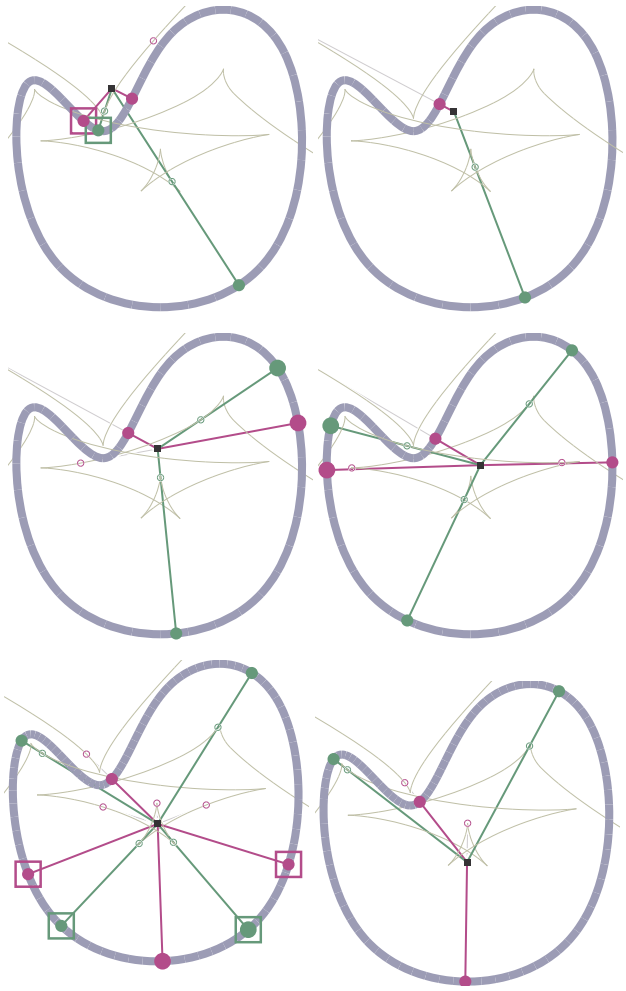


Figure 10. Example 1: Tracking Critical Distances on a C^2 B-spline Curve

(and its type) when perturbing the plane point across normal lines at C^1 or C^0 break points.

The videos of these demos are accessible by following http://www.cs.utah.edu/ xchen/papers/more.html

12 Conclusion

In this paper, we have presented a problem-formulation of an implicit surface $\mathcal{I}$ in the augmented parametric space for dynamic point-curve critical distances. The evolution and the transition of critical distances are achieved by first and second order differential computation on $\mathcal{I}$, respectively. The detection of transition is robustly and efficiently implemented using bounding volume tree of the curve evolute. Extra transition events, at curve break points, are also computed.

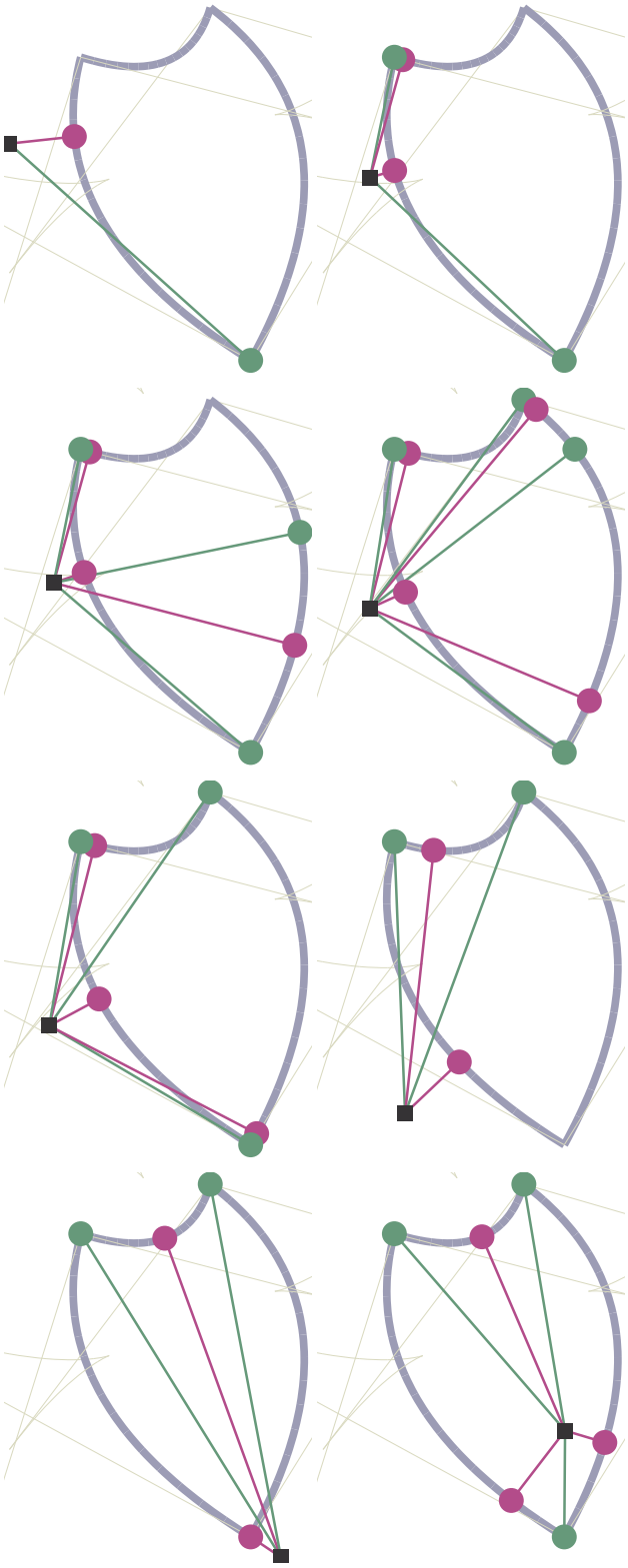


Figure 11. Example 2: Tracking Critical Distances on a C^0 B-spline Curve

So far we have not mentioned global minimal/maximal distance tracking at all. This is because, to robustly track *global minimal/maximal* distance, all the *local critical* distances have to be tracked. Because critical distances always comes with alternating types [4](cf.Section 9.3), the global minimal/maximal distance can be computed at run time by searching and comparing every other current critical distance.

Notice that global *minimal* distance tracking might also be done using medial axis transformation (MAT) on the curve, and probably more efficiently. This is a possible topic for our future work.

With this continous tracking of point-curve critical distances efficiently achieved using local marching, robust transition detection, and occasional transition computation, *without any traditional global searching*, our future task, as pointed out in Section 2, is to extend the approach and design robust algorithms for the point-model case and model-model case.

References

[1] V. Arnold, *Catastrophe Theory*, 3 edition, Springer-Verlag, 1992.

[2] H. Blum, "A transformation for extracting new descriptors of shape," *Models for the perception of speech and visual forms*. 1967, pp. 362–380, MIT Press.

[3] X. Chen, R. Riesenfeld, and E. Cohen, "Degree Reduction Strategies for NURBS Symbolic Computation," *submitted*, Dec. 2005.

[4] X. Chen, R. Riesenfeld, and E. Cohen, "Extended Curve Evolute as the Transition Set of Distance Functions," *Under submission*, 2006.

[5] J. J. Chou, "Voronoi diagrams for planar shapes," *IEEE Computer Graphics and Applications*, vol. 15, no. 2, 1995, pp. 52–59.

[6] D. E.Johnson, P. Willemsen, and E. Cohen, "A Haptic System for Virtual Prototyping of Polygonal Models," *DETC 2004*, 2004.

[7] G. Elber and M.-S. Kim, "Geometric constraint solver using multivariate rational spline functions," *Symposium on Solid Modeling and Applications*, 2001, pp. 1–10.

[8] A. Gregory, M. C. Lin, S. Gottschalk, and R. Taylor, "A Framework for Fast and Accurate Collision Detection for Haptic Interaction," *IEEE VR 1999*, 1999.

[9] T. V. T. II and E. Cohen, "Direct Haptic Rendering Of Complex Trimmed NURBS Models," *ASME Proc. 8th Annual Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Nov. 1999.

[4] a degenerate critical distance of neither minimum nor maximum could never happen in practical implementation due to numerical error or by intentional ϵ-perturbation

[10] D. Johnson and E. Cohen, "A framework for efficient minimum distance computations," *Proc. IEEE Intl. Conf. Robotics and Automation*, May 1998, pp. 3678–3684.

[11] D. Johnson and E. Cohen, "Bound Coherence for Minimum Distance Computations," *IEEE Proc. International Conference on Robotics and Automation*, 1999.

[12] J.W.Bruce and P.J.Giblin, *Curves And Singularities*, 2 edition, Cambridge University Press, 1992.

[13] J. J. Koenderink, *Solid Shape*, MIT press, 1990.

[14] P. Koparlar and S. P. Mudur, "A new class of algorithms for the processing of parametric curves," *Computer-Aided Design*, vol. 15, 1983, pp. 41–45.

[15] J. M. Lane and R. F. Riesenfeld, "A theorectical development for the computer generation and display of piecewise polynomial surfaces," *IEEE Trans. PAMI*, vol. 2, 1980, pp. 35–46.

[16] M. C. Lin, "Efficient Collision Detection for Animation and Robotics," *Ph.D. thesis,* University of California, Berkeley, 1993.

[17] M. C. Lin and D. Manocha, "Collision and proximity queries," *Handbook of discrete and computational geometry*, 2004, pp. 787–807.

[18] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *SIGGRAPH 1999*, 1999, pp. 401–408.

[19] D. D. Nelson, D. Johnson, and E. Cohen, "Haptic Rendering of Surface-to-Surface Sculpted Model Interaction," *ASME Proc. 8th Annual Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Nov. 1999.

[20] I. R. Porteous, *Geometric Differentiation: For the Intelligence of Curves and Surfaces*, 2 edition, Cambridge University Press, 2001.

[21] P. T. Saunders, *An Introduction to Catastrophe Theory*, 2 edition, Cambridge University Press, 1980.

[22] T. W. Sederberg and T. Nishita, "Curve intersection using Bézier clipping," *Computer-Aided Design*, vol. 22, 1990, pp. 538–549.

[23] T. W. Sederberg and S. R. Parry, "A comparison of curve-curve intersection algorithms," *Computer-Aided Design*, vol. 18, 1986, pp. 58–63.

[24] E. C. Sherbrooke and N. M. Patrikalakis, "Computation of the solutions of nonlinear polynomial systems," *Computer Aided Geometric Design*, vol. 10, no. 5, 1993, pp. 379–405.

[25] P. Shirley and R. K. Morley, *Realistic Ray Tracing*, 2 edition, A K Peters Ltd., 2003.

M Appendix: Proof of Eq. (6.2)

By Eq. (3.2),

$$\begin{aligned}\mathfrak{D}' &= \Big((\gamma - p)\cdot\gamma'' + |\gamma'|^2\Big)' \\ &= 3\gamma'\cdot\gamma'' - (p-\gamma)\cdot\gamma'''\end{aligned}$$

At a transition point,

$$p - \gamma = \frac{1}{\kappa}\ \frac{\gamma'_{\mathfrak{r}}}{|\gamma'|},$$

and consequently,

$$\mathfrak{D}' = 3\gamma'\cdot\gamma'' - \frac{1}{\kappa}\ \ \frac{\gamma'_{\mathfrak{r}}\cdot\gamma'''}{|\gamma'|},$$

or,

$$\mathfrak{D}'|\gamma'| = 3(\gamma'\cdot\gamma'')\ |\gamma'| - \frac{[\gamma'\ \ \gamma''']}{\kappa}. \qquad \text{(M.1)}$$

On the other hand,

$$\kappa = \frac{[\gamma'\ \ \gamma'']}{|\gamma'|^3},$$

or,

$$\kappa|\gamma'|^3 = [\gamma'\ \ \gamma''].$$

Taking the derivatives yields,

$$\kappa'|\gamma'|^3 + \kappa(|\gamma'|^3)' = [\gamma'\ \ \gamma'''].$$

Substituting

$$\begin{aligned}(|\gamma'|^3)' &= 3|\gamma'|^2(|\gamma'|)'\\ &= 3|\gamma'|^2(\sqrt{\gamma'\cdot\gamma'})'\\ &= 3|\gamma'|(\gamma'\cdot\gamma''),\end{aligned}$$

yields,

$$\frac{\kappa'}{\kappa}|\gamma'|^3 + 3|\gamma'|\ (\gamma'\cdot\gamma'') = \frac{[\gamma'\ \gamma''']}{\kappa}.$$

Therefore, by Eq. (M.1),

$$\mathfrak{D}' = -|\gamma'|^2\frac{\kappa'}{\kappa}, \qquad \text{(M.2)}$$

Also, substituting curvature equation into Eq. (M.1) yields,

$$\mathfrak{D}' = 3(\gamma'\cdot\gamma'') - \frac{[\gamma'\ \ \gamma''']}{[\gamma'\ \ \gamma'']}|\gamma'|^2. \qquad \text{(M.3)}$$