An evolution–based approach for approximate parameterization of implicitly defined curves by polynomial parametric spline curves

Huaiping Yang, Bert Jüttler and Laureano Gonzalez–Vega

Abstract. We propose a novel approach for the approximate parameterization of an implicitly defined curve in the plane by polynomial parametric spline curves. The method generates the parameterization of the curve (which may consist of several open and closed branches) without using any a priori information about its topology. If needed the topology of the approximate parameterization can be certified against the initial curve in a simple way.

Keywords. Evolution, approximate parameterization, implicitly defined curves.

1. Introduction

Two major approaches for describing curves and surfaces in Computer Aided Design and Geometric Modeling exist: parametric representations, such as NURBS curves and surfaces, and implicit representations. In many geometric algorithms, such as the computation of surface-surface intersections, combining these two representations helps to improve the quality of results and computational performance.

This observation has motivated research on conversion algorithms between parametric representation and implicit representation. In particular, *approximate* techniques for implicitization [9, 10, 11, 15, 16, 17, 24, 36] and parameterization [3, 4, 5, 22, 30] may be useful for applications in Computer Aided Design, since they are able to deal with input specified by floating point numbers and they are more flexible than exact, symbolic–computation based methods (see, e.g., [12, 13, 33]). In addition, approximate techniques can be adapted so as to produce a representation within a certain region of interest, which is a very natural requirement in applications.

This paper is devoted to the case of implicitly defined planar curves, where the region of interest is a certain box. These curves arise naturally when discussing

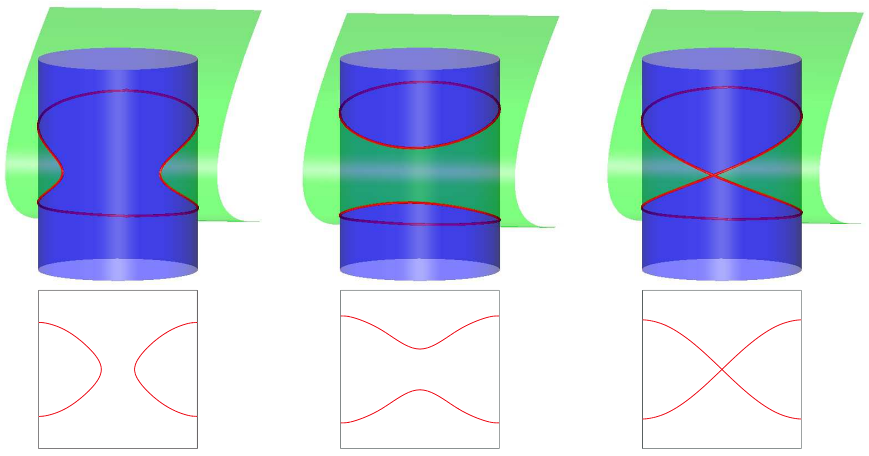


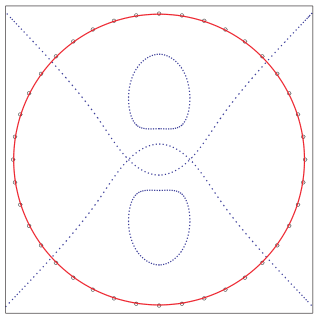
FIGURE 1. Surface-surface intersection.

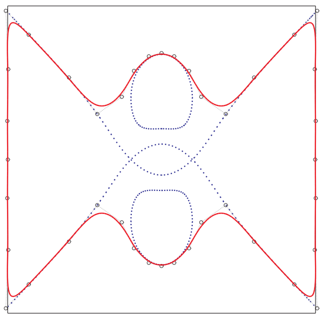
surface–surface intersections between an implicitly defined and a parametric surface. This configuration accounts for 90% of the surface–surface intersections occurring in practice [26, 29]. As an example, Fig. 1 shows the intersection between a cylinder and a biquadratic surface. Depending on the relative positions of the two surfaces, one obtains one or two components, or even a curve with a singular point. We used the algorithm described in this paper in order to represent the intersection curves.

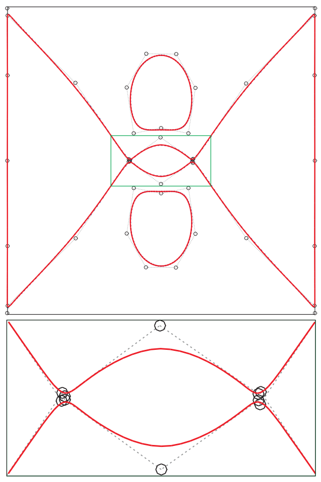
Typically, the methods for analyzing and parameterizing an implicitly defined curve consist of two stages. In the first step, the topology is analyzed, by detecting singular points, closed loops and intersections with the boundaries of the box [23, 26]. In particular, most algorithms identify a collection of characteristic points, which may include singularities and points with vertical and horizontal tangents. This step requires robust numerical solvers for systems of polynomial equations. In the next step, predictor–corrector methods are used to trace the segments in between the characteristic points. This produces a piecewise linear representation of the curve, which can then be approximated by splines.

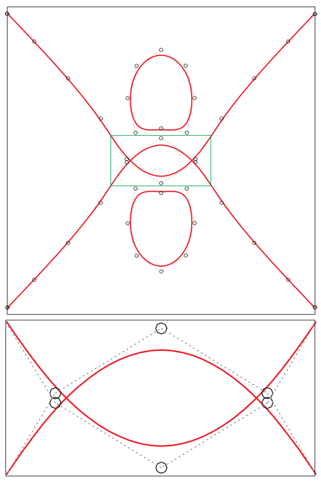
This paper proposes an all–at–once approach, which relies on an evolution process (see Figure 2): Starting from the bounding box of the domain of interest, a closed B-spline curve is moved gradually towards the given implicit curve. The evolution is governed by a differential equation which is derived from the function defining the given curve. The evolving parametric B-spline curve may split itself into multiple B-spline curves, thereby adapting its topology to the given implicitly defined curve. Our method can produce the parameterization result of a planar implicitly defined curve (not necessarily algebraic) within a bounding box, without knowing any a priori information about its topology.

The idea of breaking a parametric curve into multiple pieces for topological adaptation has also been used in other areas. In the field of image processing,


(a) Initial B-spline curve


(b) Intermediate result


(c) Before post-processing


(d) After post-processing

	(a)	(b)	(c)	(d)
# spline curves	1	1	3	4
# control points	40	40	42	46
Average error	1.753×10^{-1}	1.434×10^{-2}	6.687×10^{-4}	1.662×10^{-4}
Maximum error	3.632×10^{-1}	1.185×10^{-1}	5.453×10^{-3}	9.646×10^{-4}

FIGURE 2. Approximate parameterization of an implicitly defined curve by cubic spline curves. The computation took 57ms.

topologically adaptive snakes [27], or 'T-snakes' [28], are proposed to segment some complex-shaped biological structures from medical images. In [31, 32], B-spline active contours with handling of topology changes are used for video segmentation. To our knowledge, no one has yet applied this idea to the approximate parametrization of implicitly defined curves.

The remainder of this paper is organized as follows. The basic evolution process is given in Section 2. Then we discuss how to handle different components of the implicit curve in Section 3. The algorithm and some implementation details are given in Section 4, and the post-processing and the final refinement process are also described in this section. After presenting some test examples in Section 5, we conclude the paper with suggestions for further research.

2. The evolution process

We use an evolution process for generating an approximate parameterization of an implicitly defined curve by parametric spline curves. This process is inspired by similar techniques in the field of Image Processing and Computer Vision [6, 8, 25, 37].

The implicitly defined curve is given as the zero level set

$$Z(f) = \{\ (x, y) \in D \mid f(x, y) = 0\ \} \tag{1}$$

of a function f, which is assumed to be at least C^3 and to have at most finitely many singularities within the domain of interest $D \subset \mathbb{R}^2$. In order to simplify the method, we assume that the curve has only four kinds of order two singularities in the domain D: crunodes (ordinary double points), acnodes (isolated points), tacnodes and cusps. Moreover, in many cases the function f is assumed to be a polynomial, which then defines an algebraic curve $Z(f)$.

2.1. Evolution equation

The evolution of the spline curve $\mathbf{x}$ is governed by

$$\frac{\partial \mathbf{x}(u, \tau)}{\partial \tau} \cdot \vec{\mathbf{n}} \approx v(\mathbf{x}(u, \tau)), \tag{2}$$

where $\mathbf{x}(u, \tau)$ is any point on the spline curve, u is the corresponding parameter value, τ is the time variable, and $v(\mathbf{x}(u, \tau))$ is a scalar-valued *velocity function* (or *speed function*) along the normal direction $\vec{\mathbf{n}}$ of the spline curve at the point $\mathbf{x}$.

From the definition of a spline curve

$$\mathbf{x}(u, \tau) = \sum_{i=1}^{n} B_i(u) C_i(\tau), \tag{3}$$

or $\mathbf{x}(u, \tau) = \mathbf{c}(\tau)^\top \mathbf{b}(u)$, where $\mathbf{b} = [B_1, B_2, ..., B_n]^\top$ for all B-splines B_i, and $\mathbf{c} = [C_1, C_2, ..., C_n]^\top$ for all B-spline control points C_i, we get

$$\frac{\partial \mathbf{x}(u, \tau)}{\partial \tau} = \dot{\mathbf{c}}(\tau)^\top \mathbf{b}(u). \tag{4}$$

Note that the number and positions of control points, and therefore also the number of knots of the spline curve, are changed during the evolution process. We use uniform knots and (almost always) periodic spline curves in all examples presented in this paper.

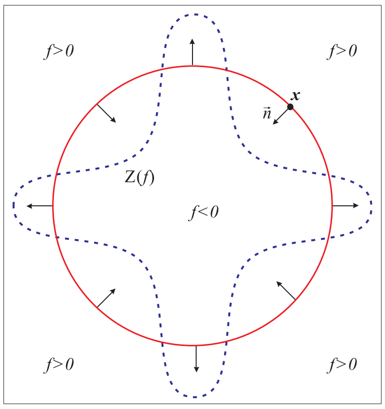


FIGURE 3. Type I speed function.

Since the condition (2) cannot be satisfied exactly in general, we adopt a least-squares approach to compute the time derivative $\dot{\mathbf{c}}$ of the B-spline control points. More precisely, we choose $\dot{\mathbf{c}}$ by solving

$$E = \int_I [(\dot{\mathbf{c}}(\tau)^\top \mathbf{b}(u)) \cdot \vec{\mathbf{n}} - v(\mathbf{x}(u,\tau))]^2 \, \mathrm{d}u \to \min_{\dot{\mathbf{c}}}, \tag{5}$$

where I is the parameter domain of the curve, which leads to a non-negative definite quadratic function of the unknowns $\dot{\mathbf{c}} = [\dot{C}_1, \dot{C}_2, ..., \dot{C}_n]^\top$. Numerical quadrature is used to evaluate the integral. For each value of $\mathbf{c}$, we compute the time derivatives by solving (5), or possibly a regularized version of this problem (e.g., one may use a simple Tikhonov regularization, see [20]).

2.2. Speed function

We use two types of evolution speed functions. The "type I" speed function drives the spline curve toward the implicit curve, which is used for approximately parameterizing each individual component (cf. Section 3). The "type II" speed function drives the spline curve towards the ridge/valley contour of the implicit curve, which is used for detecting nested components. In this section, we will describe the type I speed function only. The type II speed function will be discussed later in Section 3.1.

The type I speed function is defined as follows:

$$v_I(\mathbf{x}) = \xi \cdot f(\mathbf{x}), \tag{6}$$

vector ∇f on the implicit curve $Z(f)$. Except for isolated points, the zero level set $Z(f)$ is located at the boundary between positive and negative regions of f. During the evolution, all components of the implicitly defined curve are represented by simple closed curves (cf Section 3). Then, for each component Z_i, we choose $\xi = +1$ if ∇f is pointing to the outside of Z_i. Otherwise, we choose $\xi = -1$. Thus, the sign of the speed function is equal to the sign of $-f$ within Z_i.

Figure 3 shows an example of type I speed function with $\xi = +1$. By using this kind of speed function, the evolving spline curve will stop at the corresponding component of the implicitly defined curve.

In order to capture nested components of the implicitly defined curve, the type I speed function will be temporarily changed to type II, which will be discussed later in Section 3.1.

2.3. Time step size for the numerical integration

For each evolution step of the spline curve (see Equation (5)), the time derivatives $\dot{\mathbf{c}}(\tau)$ are computed by solving a sparse linear system of equations, $\nabla E = 0$. We then generate the updated B-spline control points

$$\mathbf{c}(\tau + \Delta\tau) = \mathbf{c}(\tau) + \dot{\mathbf{c}}\Delta\tau \tag{7}$$

simply by using an explicit Euler method with step size $\Delta\tau$. In order to prevent the spline curve from moving across the implicitly defined curve $Z(f)$, the step size is chosen as

$$\Delta\tau = \min_u \{\ \delta(\mathbf{x}(u,\tau))/|\dot{\mathbf{c}}(\tau)^\top \mathbf{b}(u)|\ \}, \tag{8}$$

where $\delta(\mathbf{x})$ is a lower bound for the Euclidean distance from the point $\mathbf{x}$ to the implicitly defined curve.

Here we use the method proposed by Taubin [34] to compute this lower bound. By using the kth-order Taylor series expansion and the Cauchy-Schwarz inequality, Taubin gives an exact lower bound δ_k for any polynomial f of degree $\leq k$. In practice, we found that the using of the second-order approximate lower bound δ_2 (cf. [34]) produces satisfying results for our method.

It is also proved in [34] that the lower bounds $\delta_2(\mathbf{x})$ and $\delta_k(\mathbf{x})$ converge to the exact Euclidean distance $d(\mathbf{x})$ between $\mathbf{x}$ and the implicitly defined curve $Z(f)$ when $d(\mathbf{x}) \to 0$, which ensures the convergence of the evolving spline curve to the implicitly defined curve.

3. Handling different domponents

The implicitly defined curve $Z(f)$ may have different types of components with various topological structures in the domain of interest D. These different components can be classified as follows:

1. **Open branches**, which are always due to the trimming by the boundary of the domain D.
2. **Closed curves**, which include both simple closed curves and self-intersecting closed curves.
3. **Isolated points**, which are corresponding to the isolated singularities.

Furthermore, these different components may be **nested**, i.e., one component can be inside another. Figure 4 illustrates the various possible components of an implicitly defined curve in a rectangular domain.

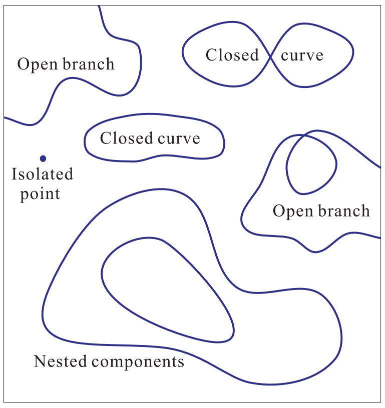


FIGURE 4. Different components of an implicitly defined curve.

During the evolution process for approximate parameterization of $Z(f)$, each component of $Z(f)$ will be represented by one or several simple closed spline curves. These closed spline curves will be denoted with Γ_i

For a simple closed component, one simple closed spline curve is used to represent it. For a closed component with self-intersections, one or more simple closed spline curves are used for the evolution, which will be reconfigured in a post-processing step (see Section 4.3) in order to produce a single self-intersecting spline curve. For an isolated point, a surrounding spline curve keeps evolving until it shrinks to this point. For open branches, a special treatment is needed, which makes it possible to treat them as closed components. This will be be discussed in Section 3.2.

We assume that the evolution process is initialized with a spline curve which contains all components of $Z(f)$. The next three sections discuss how we deal with

- nested components, i.e., components of $Z(f)$ which are contained within other components,
- open branches, and
- the adaptation of the spline curves to several non-nested components of $Z(f)$.

Based on these preparations, the entire parameterization algorithm will be formulated later in Section 4.

3.1. Handling nested components

We consider the following situation: During an evolution process driven by the type I speed function, a spline curve Γ_i has stopped at a component of the zero level set $Z(f)$. In the general case, $Z(f)$ may have some interior components inside this component contour (Fig. 5 shows such an example). In order to capture these interior components, we make a copy $\hat{\Gamma}_i$ of the stopped spline curve Γ_i, and apply the following "type II" speed function for the evolution of $\hat{\Gamma}_i$,

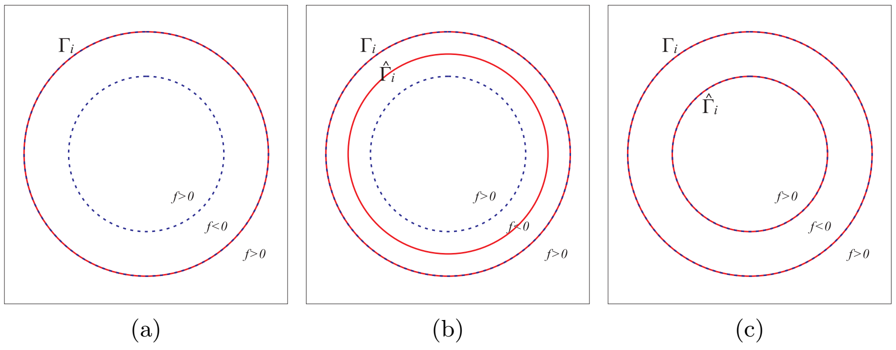


FIGURE 5. Handling nested components. In (a), the spline curve Γ_i stops at the outer contour after the type I evolution. In (b), the duplicated curve $\hat{\Gamma}_i$ stops at the valley contour after the type II evolution. In (c), $\hat{\Gamma}_i$ stops at the inner contour after resuming the type I evolution again.

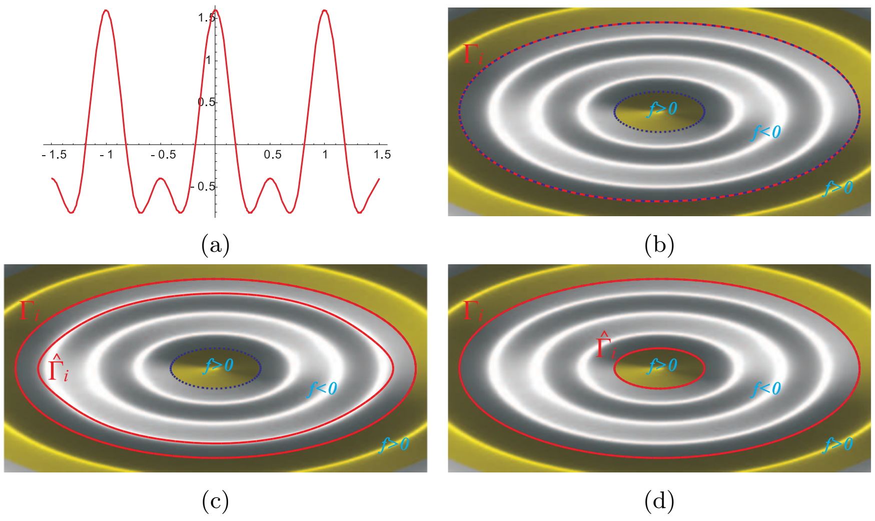


FIGURE 6. Handling nested components with multiple ridge/valley contours in between. The implicit curve is defined the function $f = \cos(2\pi\sqrt{x^2+5y^2}) + 0.6\cos(4\pi\sqrt{x^2+5y^2})$ with the cross-section view shown in (a). In (b), the spline curve Γ_i stops at the outer contour after the type I evolution. In (c), the duplicated curve $\hat{\Gamma}_i$ stops the type II evolution as soon as its area is not decreased any more. In (d), $\hat{\Gamma}_i$ stops at the inner contour after resuming the type I evolution again.

$$v_{II}(\mathbf{x}) = \hat{\xi}_i \nabla f(\mathbf{x}) \cdot \vec{\mathbf{n}}, \;\; \mathbf{x} \in \hat{\Gamma}_i. \tag{9}$$

Note that $\hat{\xi}_i = -\xi_i$, since the orientation of ∇f changes between neighboring nested components (cf. Section 2.2 and Figure 5). By using the type II speed function, usually $\hat{\Gamma}_i$ will stop at some ridge/valley contour of $Z(f)$, as illustrated in Figure 5 (b). Then we change back to the type I evolution for $\hat{\Gamma}_i$, such that $\hat{\Gamma}_i$ will stop at the desired interior contour (cf. Figure 5 (c)). Due to choice of the sign of the type 1 speed function, the evolution process drives the spline curve towards the interior contour.

When there are multiple loops or branches of ridge/valley contours between neighboring components of $Z(f)$, the type II evolution stops at the first ridge/valley contour. But that does not matter, since an arbitrary closed curve in between the two neighboring components of $Z(f)$ is sufficient to capture the inner component. We then turn back to the type I evolution. In practice, we do so as soon as the area of $\hat{\Gamma}_i$ is not decreased any more. Figure 6 shows an example of nested components with multiple ridge/valley contours in between.

This strategy of alternating between type I and type II speed functions can be iteratively applied to the evolution process, until the spline curve shrinks to a point. In this way, we can always capture all interior components in the domain of interest, no matter how many nested layers exist in $Z(f)$.

The step size should be chosen carefully for the type II evolution, such that the updated spline curve will not move into the interior component. In our case, we assume that the distance between nested components has a lower bound β, which is called the *feature size* of nested components of $Z(f)$. We then choose the time step size $\Delta\tau = \min_u\{\ \beta / |\dot{\mathbf{c}}(\tau)^\top \mathbf{b}(u)|\ \}$, in order to make sure that no component will be missed.

In our method, some self-intersecting components are also regarded as a special case of nested components, as shown in Figure 7. For this case, we first detect singular points on the stopped spline curve Γ_i (this detection process is done for all nested components before the type II evolution starts, and Section 4.2 will describe how to detect singularities). For example, we get a crunode $\mathbf{p}_0$ (the self-intersection point) in Figure 7 (a). Then we fix the point $\mathbf{p}_0$ during the type II evolution of $\hat{\Gamma}_i$. Here some points near $\mathbf{p}_0$ may easily move into the interior component, but that does not matter since the type II speed function will pull them back to the ridge/valley contour (cf. Figure 7 (b)).

3.2. Handling open branches

For open branches of the implicitly defined curve $Z(f)$, which are always due to the trimming by the boundary of the domain D, we also use simple closed spline curves to represent them. The basic idea is to connect each open branch with the trimmed segments of the boundary ∂D, such that a closed curve is obtained. Then during the evolution process, in order to attract the corresponding spline curve to

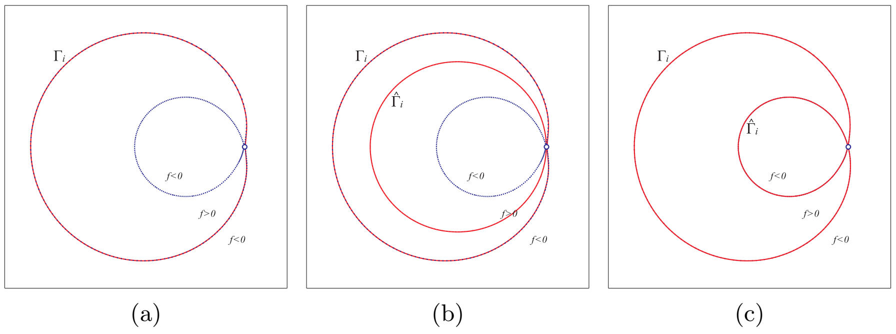


FIGURE 7. Handling self-nested components. The implicitly defined curve is defined by the zero set of $f = (x^2 + y^2 - 1)(0.1 - (x-0.3)^2 - y^2) - 0.0564$ in the bounded domain $D = [-1.2, 1.2] \times [-1.2, 1.2]$, which has a crunode. In (a), the spline curve Γ_i stops at the outer contour after the type I evolution. In (b), the duplicated curve $\hat{\Gamma}_i$ stops at the ridge contour after the type II evolution. In (c), $\hat{\Gamma}_i$ stops at the inner contour after resuming the type I evolution again.

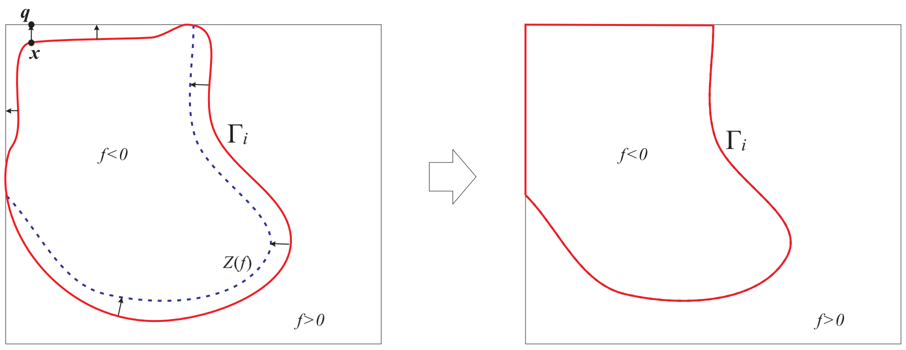


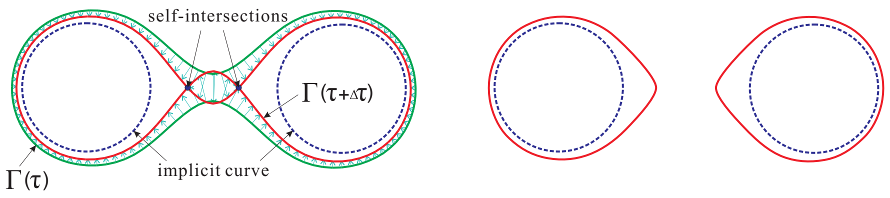
FIGURE 8. An open branch is represented by a closed spline curve.

this closed curve, we modify the type I speed function as follows

$$\tilde{v}_I(\mathbf{x}) = \begin{cases} (\mathbf{q} - \mathbf{x}) \cdot \vec{\mathbf{n}}, & \text{if } \mathbf{x} \text{ is outside of D} \\ \operatorname{sign}(v_I(\mathbf{x})) \cdot \min(|\delta(\mathbf{x})|, \|\mathbf{x} - \mathbf{s}\|), & \text{otherwise} \end{cases} \tag{10}$$

where $\mathbf{q}$ is the closest point on the boundary ∂D to the point $\mathbf{x}$, $\mathbf{s}$ is the first intersection point between the ray $R(\Delta t) = \mathbf{x} + \operatorname{sign}(v_I(\mathbf{x}))\Delta t \cdot \vec{\mathbf{n}}$, $\Delta t \geq 0$ and the boundary ∂D, and $\delta(\mathbf{x})$ is as given in Section 2.3.

As shown in Figure 8, by using the modified version of type I speed function, the evolving spline curve will stop at the closed contour, which is consisting of an open branch and partial boundary segments. The included boundary segments



(a) Before the topology change (b) After the topology change

FIGURE 9. Handling topology changes.

will be removed from the result in a post-processing step, which is described later in Section 4.3.

3.3. Handling topological changes

In general, the implicitly defined curve divides the region of interest is into multiply connected regions. Thus, the evolving spline curve has to split itself in order to adapt its topology to the implicitly defined curve. This is done when self-intersections occur on the evolving spline curve. After each evolution step, the following procedure is applied to check and handle topology changes for each spline curve

1. Check if the spline curve has self-intersections. If no self-intersections, exit.
2. Split the self-intersecting spline curve into multiple simple closed curves, which are separated by the self-intersection points.
3. Remove the redundant curves, whose interior is completely scanned by the movements of spline curves.

Figure 9 shows an example of the topology change. Due to the time step constraint, the interior of a redundant curve does not contain any zero point of the function f, and thus can be safely removed without missing any component of the implicitly defined curve $Z(f)$. After the change of topology, the original spline curve is split into multiple ones, which correspond to the different components of $Z(f)$.

4. Algorithm and implementation

In this section, we present the algorithm and discuss some implementation details. The algorithm takes as input an implicit function f, the domain of interest D and a user-specified error tolerance ϵ_0. The given function f is at least C^3 and the implicitly defined curve has at most a finite number of four kinds of order two singularities (crunodes, acnodes, tacnodes and cusps) in the domain D. The implicit curve $Z(f)$ has the *feature size* β (cf. Section 3.1) in the domain D.

The algorithm produces a set of spline curves, which approximately parameterize the implicit curve $Z(f)$ in the domain D, and the approximation error is bounded by ϵ_0. Here the approximation error ϵ is measured by the one-sided

Hausdorff distance from the sample points on the spline curves to the implicit curve

$$\epsilon = \max_{i=1,2,\ldots,N} \|\mathbf{x}_i - \mathbf{p}_i\|, \tag{11}$$

where $\{\mathbf{x}_i\}_{i=1,2,\ldots,N}$ are a set of uniformly sampled points on the spline curves, and $\mathbf{p}_i$ is the closest point on $Z(f)$ to $\mathbf{x}_i$.

4.1. Algorithm

The entire parameterization algorithm consists of 6 steps.

1. Initialize the spline curve Γ_0 as the boundary contour ∂D of the domain D. Initialize the set of spline curves as $\Gamma = \{\Gamma_0\}$. Initialize the output as $\tilde{\Gamma} = \emptyset$.
2. Check if Γ_0 intersects the implicit curve. If not, then set $\xi_0 = \text{sign}(f(\mathbf{x}_0))$ ($\mathbf{x}_0 \in \Gamma_0$) for the type I evolution of Γ_0, with the speed function v_I (cf. Equation (6)). Otherwise, set $\xi_0 = +1$ for the type I evolution of Γ_0, with the speed function $\tilde{v}_I$ (cf. Equation (10)) for handling open branches.
3. While $\Gamma \neq \emptyset$ do
   - 3.1. Choose a spline curve $\Gamma_i \in \Gamma$.
   - 3.2. Apply one step of the type I evolution of Γ_i. If topology changes happen (cf. Section 3.3), then add each generated non-redundant curve into the set Γ, and delete Γ_i from it, go to step 3.1.
   - 3.3. Repeat step 3.2. until Γ_i stops evolution.
   - 3.4. Detect singularities in the vicinity of Γ_i, and identify the type of each singular point (cf. Section 4.2). The detected crunodes as shown in Figure 7 will be fixed for the type II evolution of $\hat{\Gamma}_i$ in step 3.7.
   - 3.5. If the area of Γ_i is sufficiently small ($< \pi\beta^2$), then delete Γ_i from Γ, and continue with step 3.
   - 3.6. Make a copy $\hat{\Gamma}_i$ of Γ_i with $\hat{\xi}_i = -\xi_i$. Set $\Gamma = (\Gamma \setminus \Gamma_i) \cup \{\hat{\Gamma}_i\}$, and $\tilde{\Gamma} = \tilde{\Gamma} \cup \Gamma_i$.
   - 3.7. Apply the type II evolution of $\hat{\Gamma}_i$, until it stops evolution (cf. Section 3.1) and continue with step 3.1.
4. Apply the post-processing of $\tilde{\Gamma}$, which produces the desired singular features and open branches of $Z(f)$. (cf. Section 4.3)
5. Apply the final refinement of $\tilde{\Gamma}$, such that the approximation error is smaller than the user-specified tolerance ϵ_0. (cf. Section 4.4)
6. Output $\tilde{\Gamma}$ as the parameterization result.

4.2. Detection and classification of singularities

A singularity of f is a point $\mathbf{x}^*$ which satisfies $f(\mathbf{x}^*) = f_x(\mathbf{x}^*) = f_y(\mathbf{x}^*) = 0$. In order to detect and classify the singularities of f in the domain D, we propose the following process.

Firstly, we uniformly sample a set of points $\{\mathbf{x}_i \mid i = 1, 2, ..., N\}$ on each spline curve. We choose those points $\{\mathbf{x}_k \mid 1 \leq k \leq N, \text{ and } |\nabla f(\mathbf{x}_k)| < \varepsilon_g\}$ (ε_g is a predefined small constant) as potential singular points, in which consecutive sample points are counted only once.

Then for each potential singular point $\mathbf{x}_k$, we use it as the initial value for applying the Newton-Raphson method to compute the root of equation $f^2(\mathbf{x}) + f_x^2(\mathbf{x}) + f_y^2(\mathbf{x}) = 0$. If a root $\mathbf{x}_k^*$ is found, then we add it to the list of singularities. The points representing the same singularity are merged together.

Finally, by analyzing the determinant $\|H(f)\|$ of the Hessian matrix and

$$g(f) = 1/6 f_{xxx} f_{yy}^3 + 1/2 f_{xxy} f_{xy} f_{yy}^2 + 1/2 f_{xyy} f_{xy}^2 f_{yy} + 1/6 f_{yyy} f_{xy}^3$$

at each singular point $\mathbf{x}_k^*$, the detected singularities are classified into four types:

1. **Crunodes**, if $\|H(f)\| < 0$. (cf. Figure 10 (a))
2. **Acnodes**, if $\|H(f)\| > 0$. (cf. Figure 10 (b))
3. **Tacnodes**, if $\|H(f)\| = 0$ and $g(f) = 0$. (cf. Figure 10 (c))
4. **Cusps**, if $\|H(f)\| = 0$ and $g(f) \neq 0$. (cf. Figure 10 (d))

The expression $g(f)$ is used to distinguish between tacnodes and cusps. Note that the computation of third derivatives of f is needed if and only if $\|H(f)\| = 0$ at the considered singular point.

4.3. Post-processing

After the singularities have been detected and classified, we reconfigure the corresponding spline curves for each singular point $\mathbf{x}_k^*$, according to the type of $\mathbf{x}_k^*$:

1. If $\mathbf{x}_k^*$ is a crunode, then we reconfigure the corresponding spline curves, such that the desired self-intersection feature is produced. (cf. Figure 10 (a)).
2. If $\mathbf{x}_k^*$ is an acnode, then we replace the surrounding spline curve with this point. (cf. Figure 10 (b)).
3. If $\mathbf{x}_k^*$ is a tacnode, then we measure the difference between the tangent $\mathbf{t}_s$ of the spline curve and the tangent $\mathbf{t}_f$ of the two osculating branches of $Z(f)$. If $\mathbf{t}_s$ is quite different from $\mathbf{t}_f$, then we change the connections between the corresponding spline curve branches accordingly. (cf. Figure 10 (c)).
4. If $\mathbf{x}_k^*$ is a cusp, then we use multiple control points at this point to produce the sharp feature. (cf. Figure 10 (d)).

For those spline curves which correspond to open branches of $Z(f)$, we split them along the intersection points with the boundary contour ∂D, and remove the curve segments on the boundary, such that the resulting open spline curves are representing the open branches only.

4.4. Final refinement

The final refinement is also achieved by using an evolution process, which is driven by the speed function

$$v_{\text{refine}}(\mathbf{x}) = (\mathbf{p} - \mathbf{x}) \cdot \vec{\mathbf{n}}, \tag{12}$$

where $\mathbf{p}$ is the closest point (foot point) on the implicit curve to $\mathbf{x}$. Robust methods for computing foot points on implicitly defined curves have been studied in [1].

After each step of evolution, we compute the approximation error ϵ (cf. Equation (11)). If ϵ is not reduced any more but still $\geq \epsilon_0$, then we add new control points into the spline segments having a large approximation error. The iterative refinement continues until the user-specified error tolerance is satisfied, i.e., $\epsilon < \epsilon_0$.

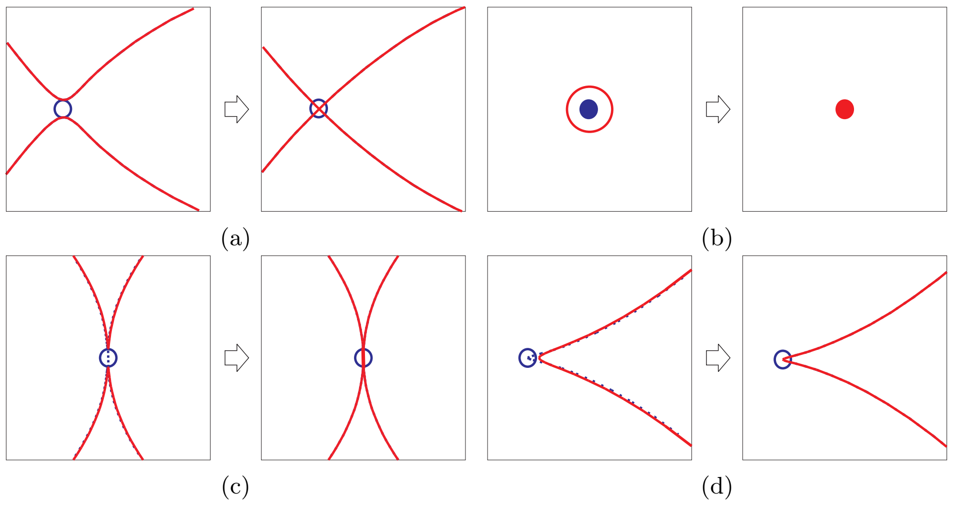


FIGURE 10. Handling singularities.

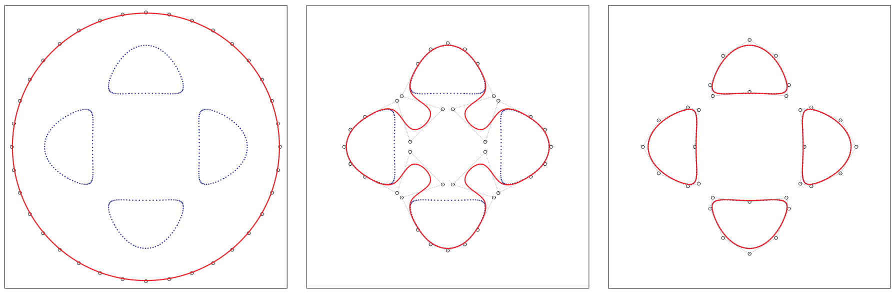
5. Experimental results and Discussion

We present some test examples to demonstrate the effectiveness and discuss the limitations of our method. The *maximum error* mentioned below is the maximum of the distances of the sample points $\{\mathbf{x}_i\}_{i=1,2,\ldots,N}$ on the spline curves to the implicit curve, i.e., the one-sided Hausdorff distance given in (11). The *average error* is the average of the distances of the sample points $\mathbf{x}_i$ to the implicit curve. The error tolerance is set to $\epsilon_0 = 1 \times 10^{-3}$ for all the presented examples. All experiments were performed on a PC with AMD Opteron(tm) 2.20GHz CPU and 3.25G RAM. The execution time shown does not include that of final refinement process in Section 4.4.

5.1. Examples

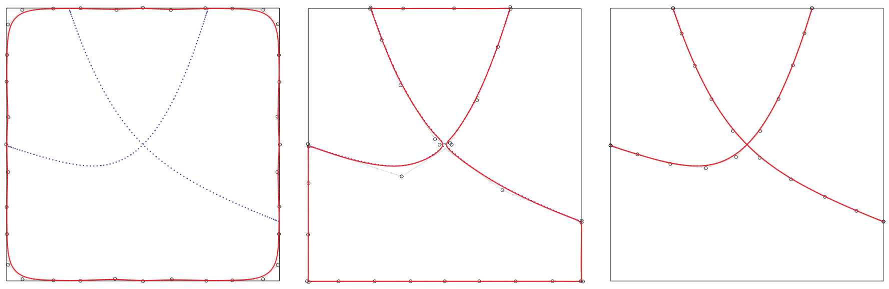
Example 1. (cf. Figure 11) The implicit curve is defined by the zero set of $f = 4y^4 + 17x^2y^2 - 20y^2 + 4x^4 - 20x^2 + 17$, which has four components in the bounded domain of $[-2.75, 2.75] \times [-2.75, 2.75]$. Figure 11 (a) shows an initial order-4 spline curve which is containing all the four components of the implicit curve. Then the spline curve is guided by the evolution to get the intermediate result in (b), and the final result in (c) (after topology changes). The approximation errors are reported in the table below the figure.

Example 2. (cf. Figure 12) The implicit curve is defined by the zero set of $f = x^3 + 3x^2y + x^2 - y^2$, which has a crunode in the bounded domain of $[-1, 1] \times [-1, 1]$. Figure 12 (a) shows the initial order-4 spline curve. Then the evolution result after 14 iterations is shown in (b). Finally, after post-processing, the improved result is given in (c). The approximation errors are reported in the table.


(a) 0 iterations (b) 11 iterations (c) 15 iterations

	(a)	(b)	(c)
# spline curves	1	1	4
# control points	36	36	32
Average error	3.178×10^{-1}	3.504×10^{-2}	3.484×10^{-4}
Maximum error	4.530×10^{-1}	2.161×10^{-1}	8.027×10^{-4}

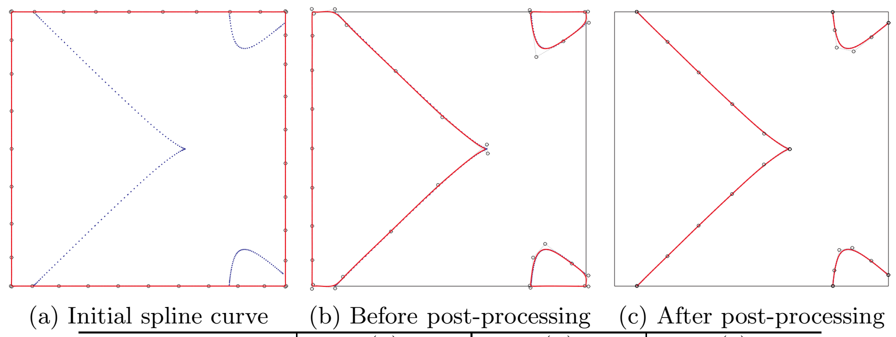
FIGURE 11. Approximate parameterization of the implicit curve with multiple components. The execution time is 32ms.


(a) Initial spline curve (b) Before post-processing (c) After post-processing

	(a)	(b)	(c)
# spline curves	1	1	1
# control points	36	33	28
Average error	1.769×10^{-1}	6.502×10^{-4}	4.435×10^{-5}
Maximum error	7.351×10^{-1}	7.943×10^{-3}	8.361×10^{-4}

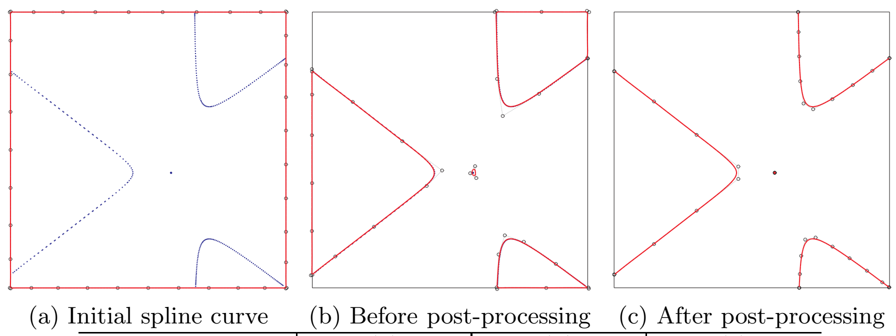
FIGURE 12. Approximate parameterization of the implicit curve with a crunode. The execution time is 15ms.

Example 3. (cf. Figure 13) The implicit curve is defined by the zero set of $f = x^3 - xy^2 - 3x^2 + 2y^2 + 3x - 1$, which has a cusp in the bounded domain of $[-3.55, 3.55] \times [-3.55, 3.55]$. Figure 13 (a) shows the initial order-4 spline curve. Then the evolution result after 36 iterations is shown in (b). Finally, after post-processing, the improved result is given in (c).



	(a)	(b)	(c)
# spline curves	1	3	3
# control points	40	30	37
Average error	1.643×10^{-1}	7.136×10^{-4}	1.681×10^{-4}
Maximum error	7.269×10^{-1}	4.586×10^{-3}	9.092×10^{-4}

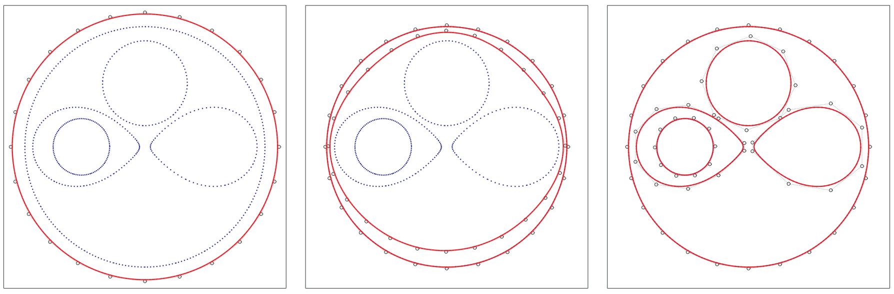
FIGURE 13. Approximate parameterization of the implicit curve with a cusp. The execution time is 47ms.



	(a)	(b)	(c)
# spline curves	1	4	4
# control points	40	34	43
Average error	1.974×10^{-1}	2.000×10^{-3}	8.692×10^{-5}
Maximum error	7.157×10^{-1}	2.816×10^{-2}	9.342×10^{-4}

FIGURE 14. Approximate parameterization of the implicit curve with an acnode. The execution time is 31ms.

Example 4. (cf. Figure 14) The implicit curve is defined by the zero set of $f = 3x^3 - 5xy^2 - 4x^2 - 10xy + 10y^2 - 6x + 20y + 12$, which has an acnode in the bounded domain of $[-6, 6] \times [-6, 6]$. Figure 14 (a) shows the initial order-4 spline curve. Then the evolution result after 23 iterations is shown in (b). Finally, after post-processing, the improved result is given in (c).


(a) 0 iterations (b) 22 iterations (c) 100 iterations

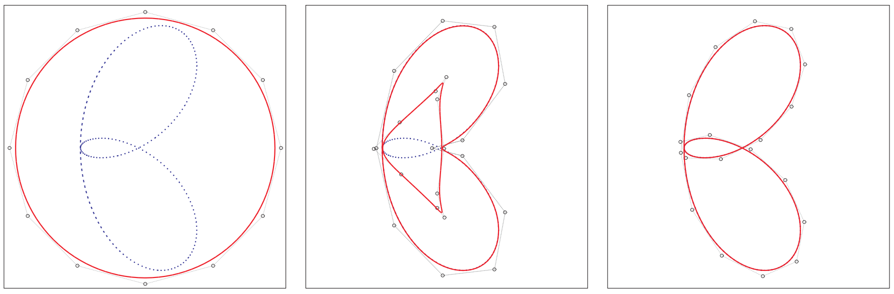
	(a)	(b)	(c)
# spline curves	1	2	5
# control points	24	48	60
Average error	8.921×10^{-2}	3.415×10^{-2}	6.232×10^{-5}
Maximum error	9.173×10^{-2}	1.182×10^{-1}	7.367×10^{-4}

FIGURE 15. Approximate parameterization of the implicit curve with nested components. The execution time is 125ms.

Example 5. (cf. Figure 15) The implicit curve is defined by the zero set of $f = (x^2+y^2-0.7225)\times[(x+0.45)^2+y^2-0.04]\times[x^2+(y-0.45)^2-0.09]\times[((x-0.75)^2+y^2)((x+0.75)^2+y^2)-0.3136]$, which has three-layered nested components in the bounded domain of $[-1,1]\times[-1,1]$. Figure 15 (a) shows the initial order-4 spline curve, which will first stop at the outermost contour by using the type I evolution for 7 iterations. Then the spline curve stops at the valley contour after applying the type II evolution for 15 iterations, as shown in (b). Figure 15 (c) shows the approximation result after the whole evolution stops (100 iterations in total). The feature size we used is $\beta = 0.01$ for the type II evolution.

Example 6. (cf. Figure 16) The implicit curve is defined by the zero set of $f = (x^2+y^2-3x)^2-4x^2(2-x)$, which has both a crunode and a tacnode in the bounded domain of $[-1.25, 3.75]\times[-2.5, 2.5]$. Figure 16 (a) shows the initial order-4 spline curve, which will first stop at the outermost contour by using the type I evolution for 21 iterations. Then the spline curve stops in (b) after applying the type II evolution for 24 iterations. Figure 16 (c) shows the final result after handling singularities. The feature size we used is $\beta = 0.1$ for the type II evolution.

Example 7. (cf. Figure 2) The implicit curve is defined by the zero set of $f = -3+12y^2+2y^4-12y^6+y^8+12x^2-28y^2x^2+12y^4x^2+4y^6x^2-18x^4+20y^2x^4+2y^4x^4+12x^6-4x^6y^2-3x^8 = 0$, which has two closed components and two open branches (including two crunodes) in the domain of $[-5,5]\times[-5,5]$. Figure 2 (a) shows the initial order-4 spline curve. An intermediate result after 8 iterations is shown in (b). The evolution result after 51 iterations is shown in (c). Finally, after post-processing, the improved result is given in (d).


(a) Initial spline curve (b) Type II evolution (c) After post-processing

	(a)	(b)	(c)
# spline curves	1	2	1
# control points	12	22	19
Average error	6.270×10^{-2}	4.715×10^{-2}	1.203×10^{-4}
Maximum error	1.249×10^{-1}	1.688×10^{-1}	4.357×10^{-4}

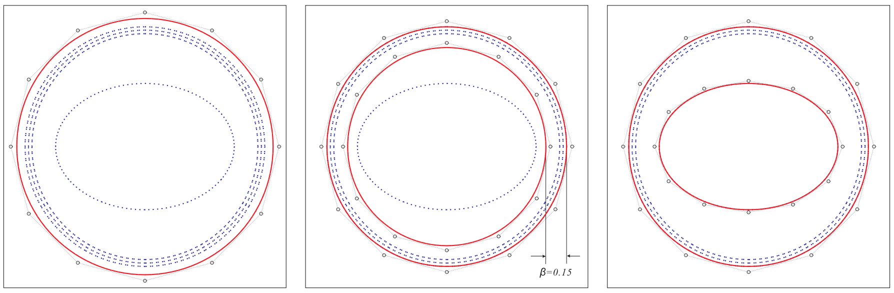
FIGURE 16. Approximate parameterization of the implicit curve with both a crunode and a tacnode. The execution time is 67ms.

5.2. Limitations

Currently, our algorithm is only able to handle second order singularities, since we use the second-order approximation of Taubin's lower bound for efficient determination of time step sizes during the evolution (cf. Section 2.3). Handling higher order singularities is possible by using higher order approximations of the lower bound, but there is still much work to do for dealing with different types of more complicate singularities based on the classification.

The feature size β (cf. Section 3.1) of the implicit curve $Z(f)$ has to be known a priori such that all nested components of $Z(f)$ can be correctly detected. However, in practice, this information is not easy to be exactly obtained in advance. If β is not small enough, then some components of $Z(f)$ will not be detected and the approximation result will be topologically wrong only because of these missing components. Figure 17 shows a failure example of this kind.

Example 8. (cf. Figure 17) In this failure example, the implicit curve is defined by the zero set of $f = (x^2+y^2-0.72)\times(x^2+y^2-0.68)\times(x^2+y^2-0.64)\times(x^2+2y^2-0.4)$, which has four-layered nested components in the bounded domain of $[-1,1] \times [-1,1]$. Figure 17 (a) shows the initial order-4 spline curve, which will first stop at the outermost contour by using the type I evolution. Since the parameter ($\beta = 0.15$) we used is much larger than the real feature size of the implicit curve $Z(f)$, the type II evolution has overshot the two components between the outermost and the innermost contours, as shown in (b). Finally, the approximation result only contains two-layered components, as shown in (c). By using a sufficiently small value $\beta = 0.01$, the correct result with all components can be obtained.



(a) Initial spline curve (b) Type II evolution (c) Failure result

FIGURE 17. A failure example with a bad feature size.

6. Conclusion and future work

We have proposed an evolution–based approach for approximately parameterizing an implicitly defined curve by parametric B-spline curves. The proposed algorithm does not require in advance any information about the topology of the considered curve (as required in the approach presented in [22]).

In order to guarantee that the final result is topologically correct and that no component of the considered curve is missing, more information is needed about how to define the so–called feature size which moreover guides the time step size choice of the evolution, cf. Section 3.1. Thus if a posteriori analysis detects that some component is missing then the feature size will be decreased and the algorithm applied back again.

One possibility for dealing in advance with this question about the right feature size to start with will be to explore the use of the root bounds presented in [38] where an explicit lower bound for the feature size is presented in terms of the degree and coefficient vector 2–norm of the polynomial defining the considered implicit curve. Nevertheless, this lower bound is in practice very small and, if used, the evolution will be very slow. More useful lower bounds could be probably obtained by analyzing locally this lower bound (the one in [38] is global) and revising how it is derived since the above mentioned lower bound is presented concerning the minimal distance between the components of two different implicit curves while, in our case, only one implicit curve is involved.

A different possibility for providing a guarantee for the topology of the approximate parametrization obtained is to use the algorithms in [23, 18] to verify a posteriori that the topology of the obtained output agrees with the topology of the considered implicit curve. Again, the application of the whole algorithm is not required: it is enough to know, for example, the number of connected components and the topological nature of the singular points in order to certify that the topology of the computed approximate parameterization fits with the topology of the considered implicit curve. Note that the approach presented in [22] requires to know in advance the topology of the considered implicit curve in order to compute the approximate parametrization, in this case, by rational quadratic B–spline curves (arcs of conic sections).

Finally, one may use the techniques described in [2] in order to obtain a bound on the (one–sided) Hausdorff distance between a curve and its approximation. Essentially, the maximum distance between the points of the spline curve and the corresponding closest points of the algebraic curve $Z(f)$ can be bounded by V/G, where V is the maximum value of f on the spline curve and G is the minimum length of the gradient in the (suitably chosen) subregion of the region of interest. Both an upper bound of V and a lower bound of G can be generated by exploiting the convex–hull property of the Bernstein–Bézier representation. Clearly, this bounds fails if the subregion contains singularities of the curve.

Acknowledgments. The first author has been supported by a Marie Curie Incoming International Fellowship of the European Commission (project no. 022073 ISIS). The second author has been supported by the Austrian Science Fund through the national research network on “Industrial Geometry” (subproject S9202). The third author has been supported by the Spanish Ministry of Science and Innovation through the research grant MTM2008-04699-C03-03. The second and third authors were also supported by the 7th Framework Programme of the European Commission through the Marie Curie Initial Training Network “Shapes, Geometry and Algebra” (SAGA, PITN-GA-2008-214584). All authors would like to thank the anonymous referees for their comments which have helped to improve the paper.

References

[1] M. Aigner and B. Jüttler, Robust computation of foot points on implicitly defined curves, in: M. Dæhlen, K. Mørken, and L. Schumaker, eds., *Math. Meth. for Curves and Surfaces*, 1–10. Nashboro Press, 2005.

[2] M. Aigner, I. Szilagyi, J. Schicho, and B. Jüttler, Implicitization and distance bounds, in Algebraic Geometry and Geometric Modelling (B. Mourrain, M. Elkadi, R. Piene, eds.), Springer, 2006, 71–86.

[3] D. A. Aruliah and R. M. Corless. Numerical parameterization of affine varieties using ODEs. In J. Gutierrez, editor, *Proc. ISSAC*, pages 12–18. ACM Press, New York, 2004.

[4] C. L. Bajaj and A. V. Royappa. Parameterization in finite precision. *Algorithmica*, 27 (2000), 100–114.

[5] C. L. Bajaj and G. L. Xu. Piecewise rational approximations of real algebraic curves. *J. Comput. Math.*, 15 (1997), 55–71.

[6] A. Blake and M. Isard, Active contours, Springer, 2000.

[7] J. Caravantes, L. Gonzalez-Vega, Computing the topology of an arrangement of quartics, in IMA Conference on the Mathematics of Surfaces, *LNCS*, 4647 (2007), 104–120.

[8] V. Caselles, R. Kimmel, and G. Sapiro, Geodesic active contours, *Int. J. of Computer Vision*, 22 (1997), 61–79.

[9] F. Chen and L. Deng. Interval implicitization of rational curves. *Comput. Aided Geom. Design*, 21 (2004), 401–415.

[10] J. Chuang, C. Hoffmann: On local implicit approximation and its applications. ACM Trans. Graphics, 8 (1989), 298–324.

[11] R. Corless, M. Giesbrecht, I. Kotsireas, and S. Watt: Numerical implicitization of parametric hypersurfaces with linear algebra. In: AISC'2000 Proceedings, Springer, LNAI 1930.

[12] D. Cox, J. Little, D. O'Shea: Ideals, Varieties and Algorithms, Springer, New York 1997.

[13] D. Cox, J. Little, D. O'Shea: Using algebraic geometry, Springer, New York 1998.

[14] D. Cox, R. Goldman and M. Zhang: On the validity of implicitization by moving quadrics for rational surfaces with no base points, J. Symbolic Computation, 29 (2000), 419–440.

[15] T. Dokken: Approximate Implicitization, in: Lyche, T., Schumaker, L. (eds.), Mathematical methods in CAGD, Nashboro Press, 2001, 1-25.

[16] T. Dokken and J. Thomassen, Overview of Approximate Implicitization, in: Topics in Algebraic Geometry and Geometric Modeling, AMS Cont. Math. 334 (2003), 169–184.

[17] T. Dokken and J. Thomassen: Weak approximate implicitization, in Algebraic Geometry and Geometric Modelling (B. Mourrain, M. Elkadi, R. Piene, eds.), Springer, 2006.

[18] A. Eigenwillig, M. Kerber, and N. Wolpert: Fast and Exact Geometric Analysis of Real Algebraic Plane Curves. Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (2007), 151–158, ACM Press.

[19] M. Elkadi and B. Mourrain: Residue and Implicitization Problem for Rational Surfaces, Applicable Algebra in Engineering, Communication and Computing, (2004), 361-379.

[20] H. Engl, M. Hanke and A. Neubauer: Regularization of inverse problems, Kluwer, Dordrecht 1996.

[21] G. Farin: Curves and Surfaces for Computer Aided Geometric Design, Academic Press, 2002.

[22] X.-S. Gao and M. Li. Rational quadratic approximation to real algebraic curves, *Comput. Aided Geom. Des.*, 21 (2004), 805–828.

[23] L. Gonzalez-Vega, I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. Comput. Aided Geom. Design **19** (2002), 719–743.

[24] B. Jüttler, A. Felis: Least-squares fitting of algebraic spline surfaces, Advances in Computational Mathematics, 17 (2002), 135–152.

[25] M. Kass, A. Witkin, and D. Terzopoulus, Snakes: active contour models, *Int. J. of computer vision*, 1 (1988), 231–233.

[26] Kunwoo Lee: Principles of CAD/CAM/CAE Systems, Prentice Hall, 1999.

[27] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. ICCV '95: Proceedings of the Fifth International Conference on Computer Vision (1995), 840–845.

[28] T. McInerney and D. Terzopoulos. T-snakes: topology adaptive snakes. Medical Image Analysis **4** (2000), 73–91.

[29] N. M. Patrikalakis and T. Maekawa. Chapter 25: Intersection problems. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of computer aided geometric design.* Elsevier, 2002.

[30] S. Perez-Diaz, J. R. Sendra, J. Sendra. Parametrization of approximate algebraic curves by lines. Theoret. Comput. Sci. **315** (2004), 627–650.

[31] F. Precioso and M. Barlaud. B-spline active contour with handling of topology changes for fast video segmentation. EURASIP J. Appl. Signal Process. **2002** (2002), 555–560.

[32] F. Precioso, M. Barlaud, T. Blu, M. Unser. Robust real-time segmentation of images and videos using a smooth-spline snake-based algorithm. IEEE Transactions on Image Processing **14** (2005), 910–924.

[33] J. R. Sendra. Normal parametrizations of algebraic plane curves. *J. Symb. Comp.*, 33 (2002), 863–885.

[34] G. Taubin, Distance approximation for rasterizing implicit curves, *ACM Transactions on Graphics*, 13 (1994), 3–42.

[35] T.W. Sederberg and F. Chen: Implicitization using moving curves and surfaces. Proc. Siggraph 1995, 301–308.

[36] L. Gonzalez-Vega: Implicitization of parametric curves and surfaces by using multidimensional Newton formulae. J. Symb. Comput. 23 (1997), 137-151.

[37] H. Yang, M. Fuchs, B. Jüttler, and O. Scherzer, Evolving T-spline level sets, Shape Modeling and Applications 2006, IEEE, pp. 247–252. Extended version available as an FSP report at `http://www.ig.jku.at`

[38] C. K. Yap. Complete subdivision algorithms, I: intersection of Bezier curves. SCG '06: Proceedings of the twenty-second annual symposium on Computational Geometry (2006), 217–226, ACM Press.

Huaiping Yang
Institute of Applied Geometry
Johannes Kepler University
Altenberger Str. 69
4040 Linz, Austria
e-mail: `yang.huaiping@jku.at`

Bert Jüttler
Institute of Applied Geometry
Johannes Kepler University
Altenberger Str. 69
4040 Linz, Austria
e-mail: `bert.juettler@jku.at`

Laureano Gonzalez–Vega
Departamento de Matematicas
Universidad de Cantabria
Avenida de los Castros s/n
39005 Santander, Cantabria, Spain
e-mail: `laureano.gonzalez@unican.es`