AAECC manuscript No.
(will be inserted by the editor)

M. Shalaby · B. Jüttler · J. Schicho

Approximate Implicitization of Planar Curves by Piecewise Rational Approximation of the Distance Function

Received: date / Revised: date

Abstract We present an approximate implicitization method for planar curves. The computed implicit representation is a piecewise rational approximation of the distance function to the given parametric curve.

The proposed method consists of four main steps: quadratic B–spline approximation of the given parametric curve, data reduction, segments–wise implicitization, multiplying with suitable polynomial factors. These segments are joined such that the collection generate a global C^r spline function which approximates the distance function, for $r = 0, 1$.

Keywords approximate implicitization · distance function

1 Introduction

The zero set of a bivariate polynomial $F(x, y)$ defines a planar algebraic curve. On the other hand, a parametric representation of the form $x = p(t)/w(t)$ and $y = q(t)/w(t)$, where $p(t), q(t)$, and $w(t)$ are (piecewise) polynomials, gives a planar rational (spline) curve. Both representations, *implicit* and *parametric*, play an essential role in Computer Aided Geometric Design. Each of them is particularly well suited for certain applications.

For example, for display purposes, it is easier to generate a large numbers of points if the parametric form is available. Also, finite segments of curves are

Mohamed Shalaby, Bert Jüttler
Institute of Applied Geometry, Johannes Kepler University, Linz, Austria.
Phone / Fax: +43 (0)732-2468-9178 / -9142
E-mail: {mohamed.shalaby, bert.juettler}@jku.at

Josef Schicho
Johannes Radon Institute of Computational and Applied Mathematics,
Austrian Academy of Sciences, Linz, Austria.
Phone / Fax: +43 (0)732-2468-5231 / –5412
E-mail: josef.schicho@oeaw.ac.at

easily defined by imposing parameter bounds. As another advantage, parametric curves can be pieced together with various degrees of derivative continuity. In addition, certain parametric curve forms (e.g., Bézier or B-spline curves) exhibit highly desirable properties such as the variation diminishing property and convex hull properties.

On the other hand, using the implicit representation one may easily decide whether a given point lies on the curve or not. As a theoretical advantage, the set of all algebraic curves is closed under offsetting (parallel curves are again algebraic curves), while parametric rational curves are not closed under this operation.

Any planar rational curve can be described as an algebraic one. The process of converting the parametric equation into implicit form is called *implicitization*. A number of established methods for *exact implicitization* exists, such as resultants, Gröbner bases, or moving curves and surfaces [2]. However, the implicitly defined curve may contain extraneous points which cannot be removed algebraically [3]. As an alternative one may use techniques for approximate implicitization [13].

In order to increase the flexibility of the implicit representation, one may work with algebraic spline curves. An algebraic spline curve is the zero contour of a spline function. A spline function is a collection of polynomials, joined together such that they form a globally C^r function. Similarly, a rational spline function is obtained by composing rational functions.

In [4,11], we discussed the problem of constructing an approximate implicit representation via *spline implicitization*. It is defined by a partition of the plane into polygonal segments along with a bivariate polynomial for each segment, such that the collection of the zero contours approximately describes the given curve. The polynomial pieces form a globally C^r spline function, for certain choice of r. The existing methods for spline implicitization are restricted to $r = 0, 1$.

This paper extends the methods of [4,11], in order to obtain a piecewise rational approximation of the (signed) distance function [1] to the given parametric curve. Distance function computation plays an essential role in several fields: computational geometry, geometric modeling, level set method, computer vision, and robotics. The graph surface of the distance function is a surface of constant slope. Its singularities correspond to the evolute of the curve. The distance function $d_F(\mathbf{X})$ can also be obtained as the viscosity solution of the so–called Eikonal equation $\|\nabla d_F(\mathbf{X})\| = 1$, (see, e.g., [10]). The problem of distance function computation has been studied by several authors from different fields [7,8,10,14].

In this paper, we propose a technique which generates a *piecewise rational* approximation of the distance function. The proposed method, which can deal with curves without singular points or self-intersections, consists of four steps, see Figure 1. First, a quadratic spline curve is used to approximate the given parametric curve. Second, a simple data reduction technique, which is based on spline wavelets, is applied. Then, the resulting quadratic segments

[1] The value of the distance function of an algebraic (spline) curve at a point $\mathbf{X} = (\bar{x}, \bar{y})$ is the smallest Euclidean distance of $\mathbf{X}$ to the zero contour of the defining (piecewise) polynomial.

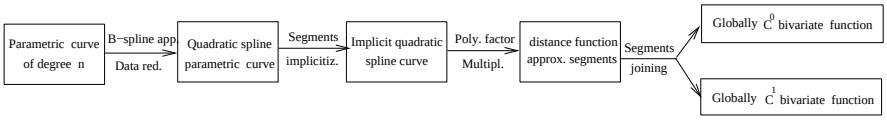

Fig. 1 Piecewise Polynomial Approximation of the Distance Function

are implicitized. Finally, by multiplying them with suitable polynomial factors, these implicitized segments are joined together. The collection of the segments forms a C^r function, for $r = 0, 1$, which approximates the distance function to the curve.

The remainder of the paper is organized as follows: In Section 2, we summarize the preprocessing steps of the spline implicitization, namely the first three steps of the proposed methods (curve approximation, data reduction and segments–wise implicitization). Sections 3 and 4 present four algorithms for generating a C^r implicit spline representation of the given parametric curve $\mathbf{g}(\mathbf{t})$ which approximate the distance function of $\mathbf{g}(\mathbf{t})$, for $r = 0, 1$ respectively.

2 Preprocessing: Conversion into quadratic spline form

Consider a given a planar parametric curve $\mathbf{g}(t)$, which does not contain singular points (cusps) or self-intersections. We assume that the coordinate functions belong to the Sobolev space $H^{2,2}$. This assumption is automatically satisfied by standard curve representations in Computer Aided Geometric Design, such as NURBS. Any curve satisfying these conditions will be called a *suitable* input curve.

First we convert the given curve into a polynomial quadratic spline curve. Following ideas of [9], we use an orthogonal projection in a suitably weighted Sobolev space, where the uniform quadratic B-Splines form an orthogonal basis. Consequently, the coefficients of the approximating curve are found by evaluating inner products (with respect to the weighted Sobolev inner product) $\langle g_i, N_j^2 \rangle$ between the coordinate functions of the given curve and the quadratic B-Splines. In order to obtain a highly accurate representation, we choose a uniform knot vector with respect to a small stepsize h. Consequently, this conversion introduces virtually no error. The approximation order is 3.

As an example, we consider a piecewise polynomial parametric curve consisting of three segments of degree 10. Figure 2a shows the error (grey) between the original curve (black) and its quadratic spline approximation with 128 segments. The error had to be amplified by a factor of 2000, since it is very small. Secondly, in order to reduce the number of segments, one may apply a simple data reduction technique which is based on spline wavelets [12]. After running the filters of wavelet analysis, we set all wavelets coefficients which are below a user–defined threshold τ to zero. After synthesis, most of the knots can be removed in general (depending on the choice of τ). Clearly, this is a fast but non–optimal technique for knot removal, and more sophisticated techniques such as [5] could be used instead.

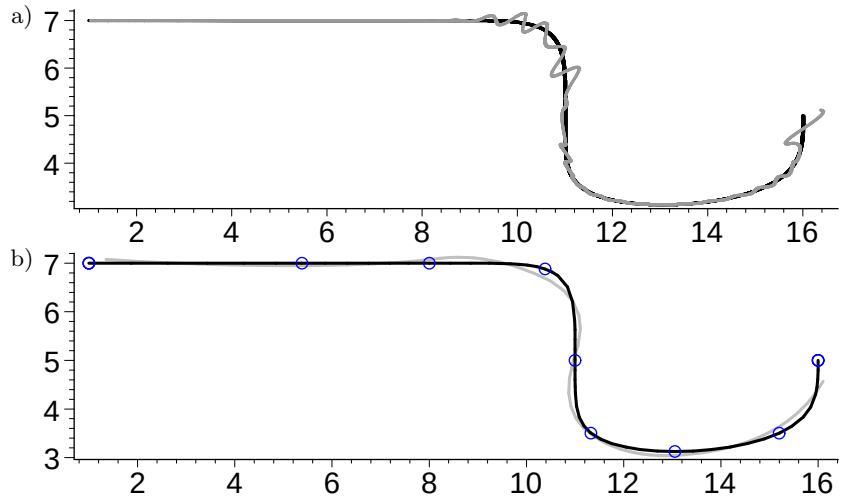

Fig. 2 a) The original curve (black) and the error (grey, exaggerated) introduced by approximating it with a quadratic spline curve with 128 knots. b) The error (grey, exaggerated) after data reduction.

In the example we chose $\tau = 0.001$. Figure 2b shows the original curve (black) and the error (grey) introduced by the data reduction. The 9 remaining knots are shown as circles. In order to make it visible, the error has been amplified by a factor of 5.

As the third step, we split the quadratic spline curve into its Bézier segments and implicitize them using Bezout resultants[2]. This produces a sequence of quadratic bivariate polynomials $(G_i)_{i=0,\ldots,m}$. Finally, in order to generate a globally C^r $(r = 0, 1)$ spline function, we join these polynomial segments G_i, $i = 1, \ldots, m$ along suitable lines called the transversal lines. This will be discussed in the next two sections.

Remark 1 The conversion process can easily be generalized to yield spline curves of degree higher than two. However, while polynomial quadratic spline curves consist of parabolic arcs, and therefore produce a relatively well–behaved implicit representation, this is not automatically true for higher degrees. For instance, the implicit representations of the Bézier segments of cubic spline curves may have singularities.

3 The C^0 Case

In order to generate a globally C^0 spline function, we join the polynomial segments G_i, $i = 1, \ldots, m$ along suitable lines, which will be called the *transversal lines*.

[2] Potential robustness problems – which we did not experience so far – could be dealt with by using. e.g., Dokken's SVD–based approximate implicitization [13]

3.1 Transversal Lines

Consider two neighboring Bézier segments of the quadratic spline curve with implicit representations $G_i(x, y) = 0$, $i = 1, 2$. These segments are parabolas which meet with tangent continuity at their junction point $\mathbf{p}$. Moreover, they intersect in two additional points $\mathbf{p}_1$ and $\mathbf{p}_1'$, see Figure 3. These two points can be both real (b) or conjugate–complex (c), at infinity (d) or one of them may coincide with $\mathbf{p}$ (a). If both coincide with $\mathbf{p}$ then the two parabolas are identical.

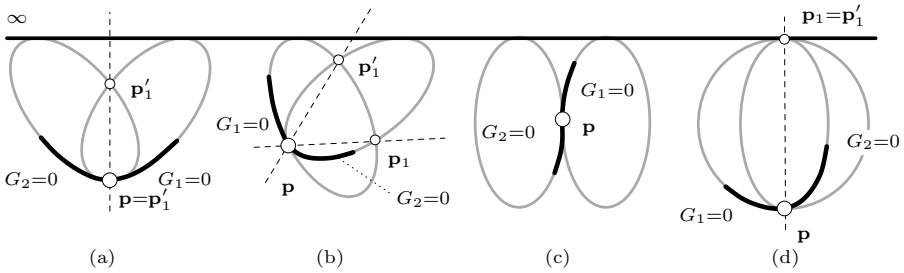

Fig. 3 Choosing the transversal line L (dashed). The line at infinity is shown as a finite line (∞), hence the parabolas appear as conic sections which are tangent to it.

The bivariate polynomials G_1 and G_2 are to be joined continuously along a transversal line L through $\mathbf{p}$. Obviously, this implies that L passes through $\mathbf{p}_1$ or $\mathbf{p}_1'$, since the restriction of either G_i to L gives a univariate polynomial, and the roots of both polynomials have to coincide. The behavior of these lines is governed by the following result.

Proposition 1 *Assume that the original parametric curve has no inflection points and is given by a C^3 arc length parameterization. If the stepsize h for converting it into quadratic spline form via orthogonal projection in a weighted Sobolev space, according to [9], is sufficiently small, then the neighboring parabolas intersect in two real points. In addition, one of the two possible transversal lines at each junction point converges to the normal of the curve as $h \to 0$.*

The proof can be found in [4]; it is based on the so–called canonical Taylor expansion of a planar curve with respect to its arc length parameter, which is a consequence of Frenet's formulas in elementary differential geometry.

Consequently, under the assumptions of the proposition, non–inflected curve segments alway produce suitable transversal lines $L = L_i$ between neighboring segments G_i and G_{i+1}. Hence, after multiplying the implicitized segments by suitable constants, the collection of these bivariate polynomials $G_i(x, y)$ – each restricted to a tile bounded by the transversal lines – forms a continuous function $G(x, y)$. This function is defined within a certain neighborhood of the curve. In the limit $h \to 0$, this neighborhood is bounded by the evolute, since it is the envelope of the normals of the curve.

Remark 2 The above construction fails at inflections of the quadratic spline curve, as the additional intersection points $\mathbf{p}_1$, $\mathbf{p}'_1$ do not exist in this case. In this situation, the two implicitized segments can still be joined continuously, provided that they can be seen as a graph of a univariate quadratic spline *function* with respect to a suitable coordinate system. If this is the case, then the two segments belong to case (d) in Figure 3 (but one of the two conics is a hyperbola which touches the line ∞ from above). A local modification of the B-spline control points, as described in [4] Section 5.3, can be used to obtain this situation.

3.2 Distance Function Approximation (DFA)

The collection of the bivariate polynomials $G_i(x, y)$, $i = 1, \ldots, m$ forms a continuous function $G(x, y)$. Unfortunately, $G(x, y)$ does not approximate the distance function. In order to obtain a function $\hat{G}(x, y)$ which approximates the distance function to the curve $G(x, y)$, we multiply $G(x, y)$ by a polynomial factor $\lambda_i(x, y)$ such that $||\nabla \hat{G}(x, y)|| \approx 1$ within the vicinity of the curve, where $\hat{G} = \lambda G$.

The quality of approximation depends on the degree of the polynomial factor $\lambda_i(x, y)$. We propose two approaches.

Approach 1. This approach works without introducing additional transversal lines. We simply multiply by constant polynomial factors $\lambda_i(x, y)$. We use the original transversal lines and follow Algorithm 1.

Algorithm 1 Basic DFA

Input A suitable parametric curve (see Section 2).

Output An approximate implicit representation by a continuous (C^0) piecewise rational approximation $\hat{G}(x, y)$ of degree 3/1 (numerator/denominator) of the distance function to the given parametric curve. It is defined within a certain neighborhood of the curve.

1: Run the preprocessing steps, Section 2.
2: Choose the transversal lines L_i through the junction points $\mathbf{p}_i$ as described in Section 3.1, see Fig. 4, and multiply the polynomials G_i by suitable constants in order to avoid the jump of the gradient at the junction points.
3: Describe the transversal lines by linear functions $L_i = a_i x + b_i y + c_i$ with normalized coefficients, i.e. $a_i^2 + b_i^2 = 1$.
4: At each junction point $\mathbf{p}_i$, compute the constant polynomial factor λ_i from the equation

$$\|\nabla(\lambda_i \, G_{i-1})\|\Big|_{(x,y)=\mathbf{p}_i} = 1 \tag{1}$$

5: For each patch, blend λ_i and λ_{i+1} using rational linear factors d_i, d_{i+1} , where $d_i = \pm L_i$ with a suitable choice of the sign. This leads to $\phi_i = (\lambda_i \, d_{i+1} + \lambda_{i+1} \, d_i)/(d_i + d_{i+1})$. Each ϕ_i is defined over the patch bounded by the two transversal lines L_i and L_{i+1}.
6: The collection of the bivariate polynomials $\phi_i \, G_i$ – each restricted to a tile bounded by the transversal lines – forms the continuous function $\hat{G}(x, y)$.

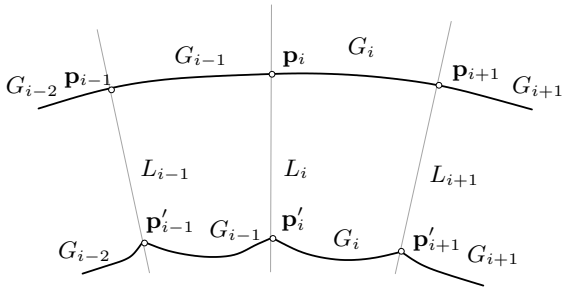

Fig. 4 Basic DFA: No additional transversal lines are needed

An example is shown in Figure 5. The piecewise rational approximation consists of 8 bivariate polynomials of degree 3/1. They are pieced together along 7 transversal lines (grey). The quality of the implicit representation is visualized by the level curves $\hat{G}(x, y) = \text{constant}$ ("algebraic offsets"). The level curves are not tangent continuous, since $\hat{G}$ is not differentiable (see the right figure).

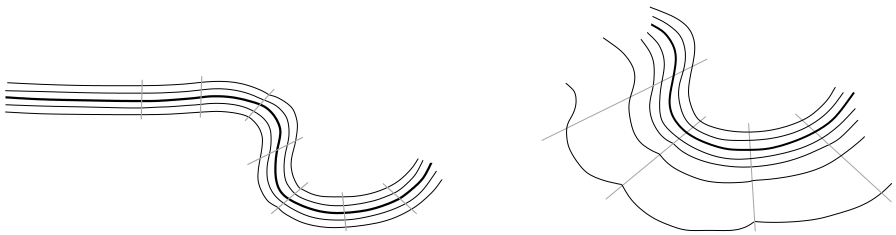

Fig. 5 Continuous approximation of the distance function obtained by using Algorithm 1 (left), enlarged (right)

Approach 2. We introduce additional transversal lines $\bar{L}_i$ and use polynomial factors of higher degree. The new transversal lines $\bar{L}_i$ serve as new partition lines. Let $\bar{\mathbf{p}}_i$ be the intersection points of $\bar{L}_i$ and $\mathcal{Z}(G(x, y))$ [3]. The gradients $\nabla G_i(x, y)$ at $\bar{\mathbf{p}}_i$ will be normalized to 1. This can be achieved by following Algorithm 2.

Figure 7 shows an example for an approximation produced by Algorithm 2 with factors of degree $n = 1$. The piecewise rational approximation consists of 16 bivariate polynomials of degree 4. They are pieced together along 15 transversal lines (grey). The quality of the implicit representation is visualized by the level curves $\hat{G}(x, y) = \text{constant}$. The level curves are not tangent continuous, since $\hat{G}$ is not differentiable (see the right figure).

In Algorithm 2, the user may choose the degree n and the number of segments (the number of additional transversal lines). Using a high degree

[3] $\mathcal{Z}(G) = \{(x, y) \mid G(x, y) = 0\}$ denotes the zero level set of a function G.

Algorithm 2 Enhanced DFA

Input A suitable parametric curve (see Section 2) and a user defined degree n.

Output An approximate implicit representation by a continuous (C^0) piecewise rational approximation $\hat{G}(x,y)$ of degree $(n+3)/1$ of the distance function.

1: Run the preprocessing steps, Section 2.
2: Choose the transversal lines as described in Section 3.1 and multiply the polynomials G_i by suitable constants, in order to obtain a globally continuous function.
3: Choose a number of additional points $\bar{\mathbf{p}}_i$ on $\mathcal{Z}(G(x,y))$, see Figure 6. For instance, one may choose the midpoints of each curve segment.
4: Introduce additional arbitrary transversal lines $\bar{L}_i$ through $\bar{\mathbf{p}}_i$. For instance, one may use the normals to the curve. Describe the transversal lines through the junction points $\bar{\mathbf{p}}_i$ by linear functions $\bar{L}_i = \bar{a}_i x + \bar{b}_i y + \bar{c}_i$ with normalized coefficients, i.e. $\bar{a}_i^2 + \bar{b}_i^2 = 1$.
5: At each point $\bar{\mathbf{p}}_i$, compute the polynomial factor λ_i of degree n from the equations

$$\|\nabla(\lambda_i\, G_{i-1})\|\Big|_{(x,y)=\bar{\mathbf{p}}_i} = 1, \quad \Big(\frac{\partial^m}{\partial^{m-\ell}(x)\partial^{\ell}(y)}\|\nabla(\lambda_i\, G_{i-1})\|\Big)\Big|_{(x,y)=\bar{\mathbf{p}}_i} = 0 \quad (2)$$

for $m = 1,\ldots,n$, $\ell = 0,\ldots,m$.
6: Blend λ_i, λ_{i+1} using rational linear factors $\bar{d}_i$, $\bar{d}_{i+1}$ where $\bar{d}_i = \pm\bar{L}_i$ with a proper choice of the sign. This leads to $\bar{\phi}_i = (\lambda_i\,\bar{d}_{i+1} + \lambda_{i+1}\,\bar{d}_i)/(\bar{d}_i + \bar{d}_{i+1})$. Each factor $\bar{\phi}_i$ is defined over the patch bounded by the two 'additional' transversal lines $\bar{L}_i$ and $\bar{L}_{i+1}$. The 'original' transversal line L_i divides this patch into two sub-patches where both G_{i-1} and G_i are defined. Hence both G_{i-1} and G_i are multiplied with $\bar{\phi}_i$.
7: The collection of the bivariate polynomials $\phi_i\, G_{i-1}$, $\phi_i\, G_i$ forms the continuous function $\hat{G}(x,y)$.

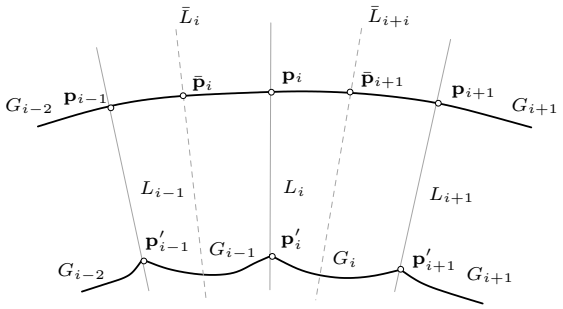

Fig. 6 Enhanced DFA: additional transversal lines $\bar{L}_i$ are introduced

polynomial factor and/or large number of transversal lines improves the quality of approximation. On the other hand, this will increase the degree of the implicit representation and/or increase the number of segments of the spline curve. This will be discussed in more detail in Section 5.

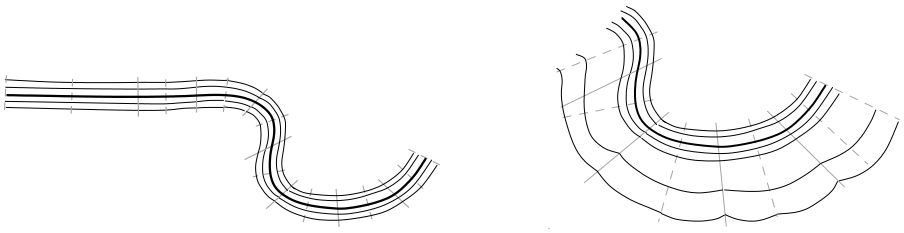

Fig. 7 Continuous approximation of the distance function obtained by using Algorithm 2 with factors of degree $n = 1$ (left), enlarged (right)

4 The C^1 Case

Differentiability (i.e., C^1 smoothness) is needed for many applications. In order to generate a globally C^1 spline function, we join the polynomial segments G_i, $i = 1, \ldots, m$ along suitable transversal lines. In addition, we multiply again by suitable quadratic polynomial factors.

In this construction, we multiply by polynomial factors *twice*. First, we multiply by suitable quadratic polynomial factors $f_{2,1}(x, y)$ and $f_{1,2}(x, y)$ in order to achieve a C^1–joint implicit representations along the transversal lines. Second, we multiply by polynomial factors λ_i in order to have obtain distance function approximation.

4.1 Transversal lines

The transversal lines are now chosen as arbitrary lines through the junction points $\mathbf{p_i}$, e.g., as the normals to the curve.

Two segments. Consider two neighboring Bézier segments of the quadratic spline curve with implicit representations $G_i(x, y) = 0$, $i = 1, 2$. Now we choose the transversal line L as an arbitrary line passing through $\mathbf{p}$, which is different from the tangent $T_{\mathbf{g}}$. Let $\mathbf{q}$ and $\mathbf{r}$ be the second intersection points of L and $\mathcal{Z}(G_1)$, see Figure 8.

We choose an arbitrary point $\mathbf{s}$ on L, which is different from $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$ (see Figure 9). In order to achieve a C^1–joint of the implicit representations along L, we multiply $G_1(x, y)$ and $G_2(x, y)$ by two quadratic polynomial factors $f_{2,1}(x, y)$ and $f_{1,2}(x, y)$, such that the following conditions are satisfied.

- Neither $f_{2,1}(x, y)$ nor $f_{1,2}(x, y))$ vanish identically along L.
- $\nabla f_{2,1}(\mathbf{r})$ and $\nabla G_2(\mathbf{r})$ are linearly dependent.
- $\nabla f_{1,2}(\mathbf{q})$ and $\nabla G_1(\mathbf{q})$ are linearly dependent.
- $\nabla f_{1,2}(\mathbf{s})$ and $\nabla f_{2,1}(\mathbf{s})$ are linearly dependent.

Consider the bivariate polynomials

$$F_1(x, y) = G_1(x, y) f_{2,1}(x, y) \qquad \text{and} \qquad F_2(x, y) = G_2(x, y) f_{1,2}(x, y).$$

As observed in [11], the two bivariate polynomials $F_1(x, y)$ and $F_2(x, y)$ meet with C^1 continuity along L after multiplying one of them with a suitable constant.

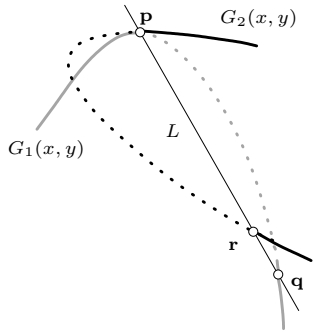

Fig. 8 Choosing the transversal line

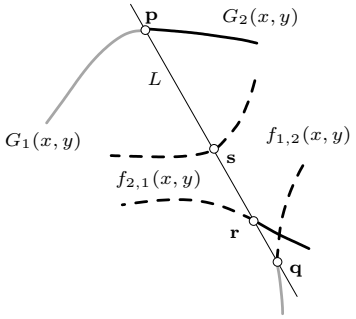

Fig. 9 Multiplication by auxiliary quadratic factors

This construction has two degrees of freedom: it is possible to choose the location of $\mathbf{s}$ on the line L and the direction of the tangent at this point.

Several segments. In order to deal with more than two segments, we propose the following algorithm:

Algorithm 3 Global Smoothing

1: Split the plane into patches by choosing arbitrary lines L_i through the junction points $\mathbf{p}_i$ of the segments. For instance, one may choose the normals to the curve.
2: Each inner patch ($i = 2, \ldots, m-1$, where m is the number of the segments) is split into two sub–patches by an arbitrary line l_i. For instance, one may choose the normal through the midpoint of the curve segments.
3: First patch: we multiply $G_1(x, y)$ by a quadratic polynomial factor $f_{2,1}$ such that $\nabla f_{2,1}(\mathbf{r}_1)$ and $\nabla G_2(\mathbf{r}_1)$ are linearly dependent.
4: Inner patches: we multiply $G_i(x, y)$ by a piecewise quadratic multiplier, defined as quadratic polynomials $f_{1,i}$ and $f_{2,i}$ on each of the two sub-patches, see Figure 10. The multiplier has to satisfy the following conditions:
   - It is C^1 on the whole patch.
   - $\nabla f_{1,i}(\mathbf{q}_{i-1})$ and $\nabla G_{i-1}(\mathbf{q}_{i-1})$ are linearly dependent.
   - $\nabla f_{1,i}(\mathbf{s}_{i-1})$ and $\nabla f_{2,i}(\mathbf{s}_{i-1})$ are linearly dependent.
   - $\nabla f_{2,i}(\mathbf{r}_i)$ and $\nabla G_{i+1}(\mathbf{r}_i)$ are linearly dependent.

   These conditions lead to a homogeneous linear system of equations which has at least one non–trivial solution.
5: The last patch is dealt with similarly to Step 3.

This construction provides two degrees of freedom. When joining the first two patches, one may choose the location of the point $\mathbf{s}_1$ and the slope of tangent T_1 (the tangent of $f_{2,1}$ at $\mathbf{s}_1$). In general, the multipliers which are applied to the other patches are then determined up to scalar constants.

'Bad' patches. In the above construction, the coordinates of the point $\mathbf{s}_i$ and the tangent direction at this point depend on the coordinates of the previously generated point $\mathbf{s}_{i-1}$ and tangent directions. After the first patch,

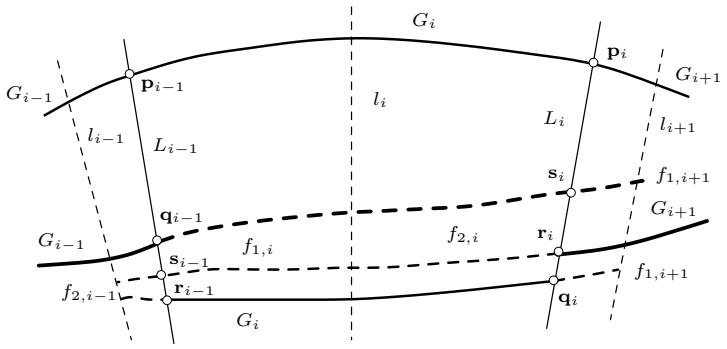

Fig. 10 The zero sets of the original curve (solid) and of the quadratic factors (dashed)

we do not have any control on the coordinates of the points $\mathbf{s}_l$, $l = 2, \ldots, m-1$ where m is the number of the patches[4]. Hence, the polynomial factors may intersect the original curve within the area of interest.

As described in [11], if there is an intersection between the polynomial factors and the original curve at any patch (for instance patch i), patch i will be split into 4 sub-patches. This will introduce 2 additional degrees of freedom which are used to choose the coordinate of $\mathbf{s}_i$ and the slope of T_i at this patch to avoid the intersection. Any such modification will act locally and affect only two patches (patch i and patch $i+1$), see Figure 11.

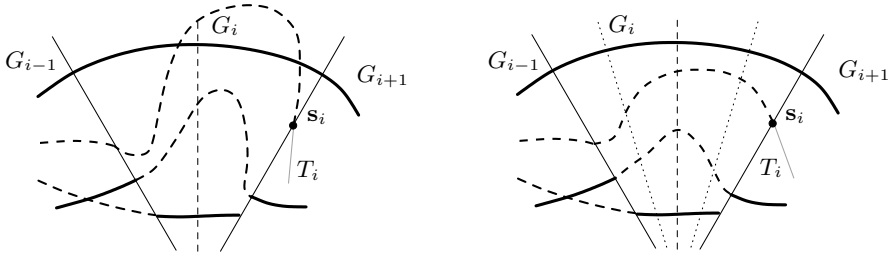

Fig. 11 Choosing $\mathbf{s}_i$, T_i to control the curve locally

Clearly, subdividing into 4 sub–patches is more expensive than subdividing into 2 sub–patches, and it leads to a higher data volume. In practice, one may use the following hybrid method. First, divide each patch into 2 sub–patches. Only if a singularity is introduced at patch i, then we discard the 2 sub-patches and subdivide into 4 sub–patches.

The proposed algorithm produces C^1 global function whose zero contour $G(x, y) = 0$ is the quartic spline curve.

[4] This is similar to C^1 interpolation with quadratic splines. Once we fix the tangent direction for the first quadratic function we do not have freedom to choose any of the tangent directions afterwards.

4.2 Distance Function Approximation (DFA)

The collection of the bivariate polynomials $G_i(x, y)$, $i = 1, \ldots, m$ forms a C^1 smooth global function $G(x, y)$. Unfortunately, $G(x, y)$ does not approximate the distance function. Once again, we multiply $G(x, y)$ by a polynomial factors $\lambda_i(x, y)$, such that $||\nabla \hat{G}(x, y)|| \approx 1$. This can be achieved by two algorithms: Algorithm 4 and Algorithm 5 which are two modified versions of the Algorithms 1 and 2. The main difference is that the blending functions d_i take the form $d_i = L_i^2$, in order to preserve the C^1 smoothness.

The first algorithm *Basic Smooth DFA* (see Algorithm 4) works without introducing additional transversal lines and uses constant factors.

Algorithm 4 Basic Smooth DFA

Input A suitable parametric curve (see Section 2).

Output An approximate implicit representation by a C^1 smooth piecewise rational approximation of degree 6/2 (numerator/denominator) of the distance function to the given parametric curve. It is defined within a certain neighborhood of the curve

1: Run the preprocessing steps, Section 2.
2: Choose the transversal lines and describe them by normalized linear equations, as in Steps 2, 3 of Algorithm 1. Join the segments with C^1 smoothness as described in Algorithm 3.
3: Algorithm 1 (Step 4).
4: Algorithm 1 (Step 5), but with $d_i = L_i^2$.
5: The collection of the bivariate polynomials $\phi_i\, G_i$ – each restricted to a tile bounded by the transversal lines – forms a C^1 smooth function $\hat{G}(x, y)$ of degree 6/2.

The second algorithm *Enhanced Smooth DFA* (see Algorithm 5) works with introducing additional transversal lines and uses general polynomial factors.

Algorithm 5 Enhanced Smooth DFA

Input A suitable parametric curve (see Section 2) and a user defined degree n.

Output An approximate implicit representation by a C^1 smooth piecewise rational approximation, of degree $(n+6)/2$, of the distance function to the given parametric curve. It is defined within a certain neighborhood of the curve.

1: Run the preprocessing steps, Section 2.
2: Choose the transversal lines and describe them by normalized linear equations, as described in Algorithm 2. Join the segments with C^1 smoothness as described in Algorithm 3.
3: Steps 3, 4, 5 of Algorithm 2.
4: Algorithm 2 (Step 6) but let $d_i = \bar{L}_i^2$. .
5: The collection of the bivariate polynomials $\phi_i\, G_{i-1}$, $\phi_i\, G_i$ forms a C^1 function $\hat{G}(x, y)$ of degree $(n + 6)/2$.

Figures 12 and 13 show the rational approximation of the distance function of the spline curve $\mathbf{g}(t)$ shown in Figure 2. The algorithms 4 and 5 with

factors of degree $n = 1$ were used. In order to assess the quality of the result, we enlarged a part of the curve and drew some additional algebraic offsets (right figures). It can clearly be seen that the algebraic offsets are C^1 smooth.

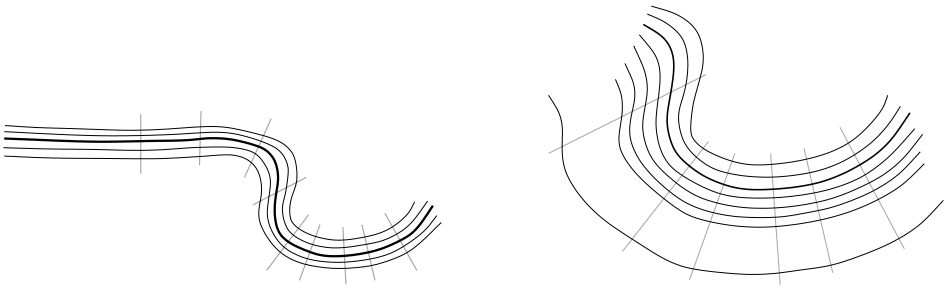

Fig. 12 C^1 smooth approximation of the distance function obtained by using Algorithm 4 (left), enlarged (right)

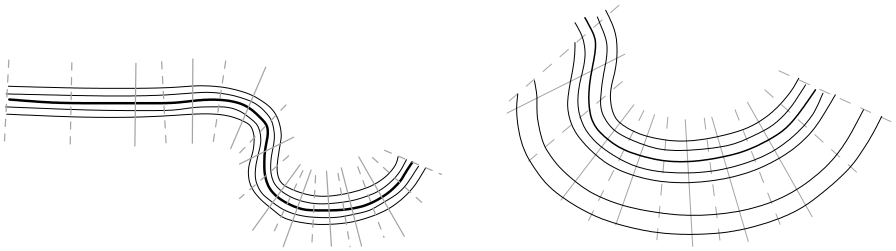

Fig. 13 C^1 smooth approximation of the distance function obtained by using Algorithm 5 with $n = 1$ (left), enlarged (right)

Figure 14 visualizes the quality of the approximation, for the previous two examples shown above. We compare the level sets of the rational approximation of the distance function (grey curves) with the corresponding offsets (which are level sets of the distance function). One may observe that the quality of the second approximation is slightly better. The quality of approximation depends on the degree of polynomial factors and the number of segments (number of transversal lines). This will be analyzed in the next section.

5 Experimental error analysis

We study the behavior of the error for the approximation of the distance function generated by the previous algorithms. Two kinds of error can be considered: First, one may analyze the *geometric error*, i.e., the maximum

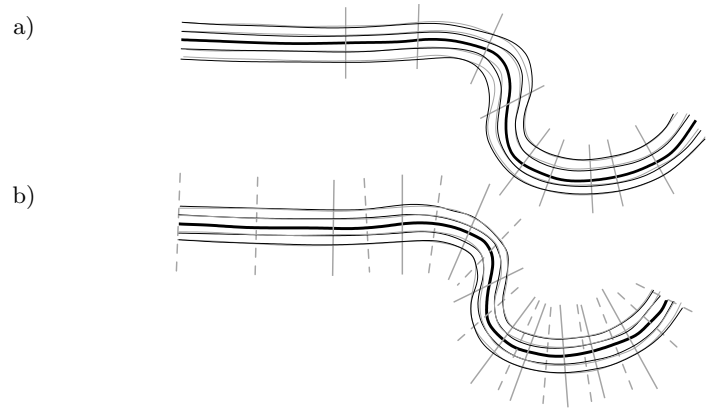

Fig. 14 Comparison of level sets of the rational approximation of the distance function (grey) with offset curves (black). (a) Basic Smooth DFA, (b) Enhanced Smooth DFA with $n = 1$.

distance between the original curve and its approximate implicit representation. Second, the *error in the distance function* should be considered, i.e., the deviation of the true signed distance function from its approximation. For the latter error, two user–defined parameters influence the result: the *degree* of the auxiliary factors, and the *step size*, i.e., the distance between the transversal lines. The influence of both parameters will be analyzed separately.

5.1 Geometric error

The geometric error is introduced only during the preprocessing stage: approximation by a quadratic spline curve via orthogonal projection in a weighted Sobolev space (step 1) and adaptive knot removal (step 2).

The approximation order of step 1 is known to be three. Virtually no error is introduced in this step, provided that the step-size is sufficiently small. On the other hand, the error introduced by the knot removal (step 2) can be bounded using standard techniques of Computer Aided Geometric Design. Several techniques for adaptive knot removal with prescribed tolerance exist, see [5].

5.2 Distance function error and the degree of the auxiliary factors

The accuracy of the approximation to the distance function decreases with the distance to the given curve. For any given tolerance, the approximation is valid only within a certain tubular neighborhood.

More precisely, consider a given planar curve $\mathbf{p}(t)$ with associated unit normals $\mathbf{n}(t)$. The mapping

$$(t, d) \mapsto \mathbf{q}(t, d) = \mathbf{p}(t) + d\ \mathbf{n}(t) \tag{3}$$

parameterizes a certain neighborhood of the curve. It is non-singular (i.e., locally bijective) for $|d| \leq |1/\kappa|$, where $\kappa = \kappa(t)$ is the curvature of the curve, and the parameter d represents the exact signed distance of $\mathbf{q}(t, d)$ to the curve.

The difference between the true signed distance and its approximation $G(\mathbf{x})$ can easily be expressed in this parameterization,

$$\delta(t, d) = |G(\,\mathbf{q}(t, d)\,) - d|. \tag{4}$$

Using an example, we analyze the behavior of this difference for various degrees of the auxiliary factor. We consider an arc of an ellipse (see Figure 15) and generate a rational quadratic parameterization $\mathbf{p}(t)$ covering the vicinity of the origin.

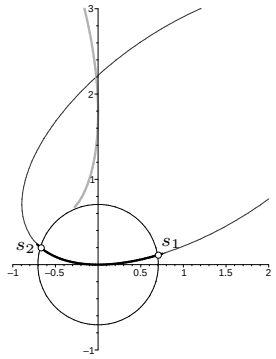

Fig. 15 The example used for analyzing the influence of the degree: An arc of an ellipse (black) and its evolute (grey).

Starting from the implicit equation of the ellipse, we generate approximations to the distance function via multiplication with auxiliary polynomials, such that (2) is satisfied at the origin.

Figure 16 shows the level sets $\delta = 0.05$ and $\delta = 0.1$ of the error (4) for various degrees of the auxiliary factor. It can be seen that the accuracy of the approximation to the signed distance function improves by using higher degrees. We conjecture that the error can be made arbitrarily small at all points which are closer to the origin than the evolute of the curve segment. These points are contained in the circle in Figure 15.

The blending procedure used in the four DFA algorithms does not increase the error, since it is based on a convex combination.

5.3 Distance function error and the step–size

Finally we study the influence of the step–size (the distance between adjacent transversal lines). A first example is shown in Figure 17, which compares the piecewise rational approximations of the distance function obtained by algorithm *Enhanced Smooth DFA* with $n = 0$ before (left) and after (right)

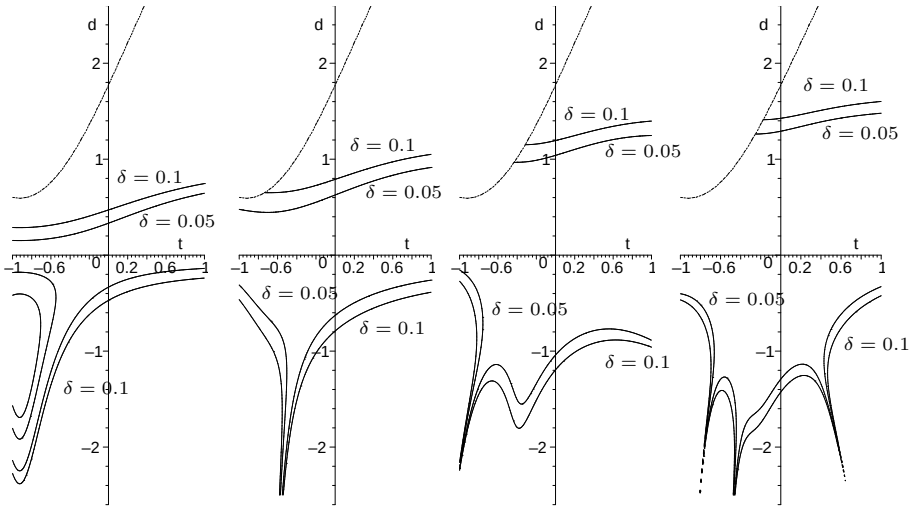

Fig. 16 Starting from left: The error of the approximated distance function using polynomial factors of degrees 0, 1, 3 and 5. The vertical and the horizontal axes correspond to the parameters (t, d) in (3). The dashed line is the graph of $d = 1/\kappa$, which corresponds to the evolute of the curve. The parameterization (3) is singular there.

adding transversal lines. Using more transversal lines improves the distribution of the level sets of the rational approximation of the signed distance function along the curve.

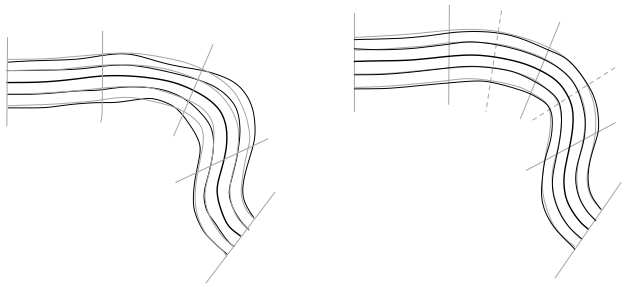

Fig. 17 Distance function approximation (black) and exact (grey), before (left) and after (right) adding new lines (dashed)

In order to analyze this phenomenon, we choose again a parameterization $\mathbf{p}(t)$ of the given curve and consider the length L of the gradient of the rational approximation of the distance function along this curve,

$$L(t) = ||\nabla G(x, y)||\Big|_{(x,y)=\mathbf{p}(t)} . \tag{5}$$

Figure 18 compares the smooth DFA obtained by using constant and linear factors, with 1, 2, 4 and 8 segments. Reducing the step–size decreases the

deviation from the (ideal) value 1 one the gradient length. In the case of constant factors, L matches the exact values (i.e., 1) at the parameters that correspond to the transversal lines, and for linear factors it also matches the first derivatives (i.e., 0). It is expected that the deviation $||L-1||_\infty$ behaves as h^n, where n is the degree of the auxiliary factors, and h is the step-size.

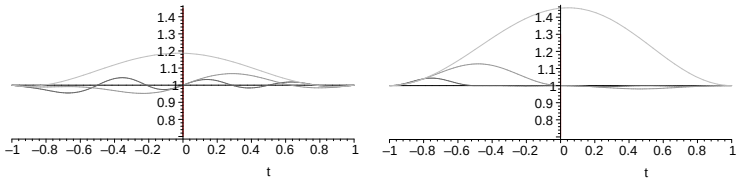

Fig. 18 Gradient length $||\nabla G||$ along the curve for decreasing step size, using constant (left) and linear (right) factors.

6 Numerical vs. graphical results

Throughout the paper, we used the same example to illustrate the proposed ideas. Due to space limitation and the size of equations, all results were presented graphically. In this section we provide some numerical values for one segment of this curve, see Figure 19. All computations used floating point numbers (note that many square root computations were needed).

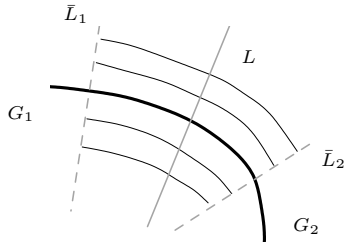

Fig. 19 Distance function approximation of the middle segment of the curve in Fig. 2

- The given curve is a polynomial curve of degree 10.
- After the preprocessing steps, Section 2 we obtain two quadratic segments with control points $[8.2, 7.1]$, $[9.2, 7.2]$, $[10.2, 6.8]$ and $[10.2, 6.8]$, $[11.2, 6.3]$, $[10.8, 5.2]$. Implicitization gives: $G_1(x, y) = 8.3061 + 2.3697\ y - 5.8337\ x + 0.3374\ x^2$ and $G_2(x, y) = -52.211 - 1.5847\ y + 5.5791\ x - 1.8154\ xy + 0.44273\ x^2 + 1.8611\ y^2$.
- We choose the transversal line $L = y + 16.27355 - 2.26351\ x$ (solid grey in the figure) and apply Algorithm 3. The joining polynomial factors are $f_1 = 15.389 - 8.3368\ x + x^2 + xy - 14.781\ y - 0.79714\ y^2$ and $f_2 = -226.90 + 25.476\ x - 0.6735\ x^2 + xy - 18.783\ y - 0.37794\ y^2$. The two

segments $\bar{G}_1(x,y) = G_1(x,y)\ f_1(x,y)$ and $\bar{G}_2(x,y) = G_2(x,y)\ f_2(x,y)$ are C^1 along the transversal line L.

- We introduce two transversal lines $\bar{L}_1 = y + 25.31258 - 3.32888\ x$ and $\bar{L}_2 = y + 0.36392 - 0.61775\ x$ (dashed gey in the figure). Algorithm 5 with $n = 1$ gives $\lambda_1 = 0.6685 - 0.03039\ x + 0.00436\ y$ and $\lambda_2 = 0.237933 + 0.011680\ x - 0.2121\ y$.
- We compute

$$\phi = \frac{\lambda_1\ \bar{L}_2^2 + \lambda_2\ \bar{L}_1^2}{\bar{L}_1^2 + \bar{L}_2^2}.$$

The bivariate rational functions $\phi\ \bar{G}_1$ and $\phi\ \bar{G}_2$ can be joined along L to form a C^1 continuous function which approximates the signed distance function.

7 Conclusions

This paper presented four algorithms for finding an approximate implicit representation for a given suitable planar parametric curve $\mathbf{g}(t)$. The computed implicit representation is a continuous or a C^1 smooth piecewise rational approximation of the signed distance function to the curve. The algorithms are computationally simple, since all computations act locally. As a matter of future research, one may generalize them to the surface case.

Among many possible applications, the proposed methods can be used to compute the bisector of two planar curves. This bisector, which consists of the centers of all circles which touch the two curves, are closely related to the so–called medial axis [1,6].

If two functions $\hat{F}(x,y)$, $\hat{G}(x,y)$ approximate the signed distance functions of two curves $\mathbf{f}(t)$, $\mathbf{g}(t)$, then the implicitly defined curves defined by $\hat{F}_i(x,y) \pm \hat{G}_j(x,z) = 0$ describe an approximation of the bisector curves. A first example is shown in Fig. 20.

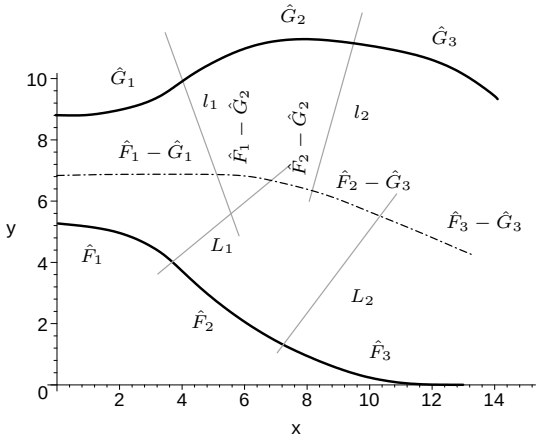

Fig. 20 Bisector approximation via approximation of the signed distance function.

Acknowledgements The authors were supported by the European Union through project IST 2001-35512, entitled "Intersection Algorithms for Geometry-Based IT Applications Using Approximate Algebraic Methods (GAIA II)", and by the Austrian Science Fund through the Special Research Area (SFB) F013 "Numerical and Symbolic Scientific Computing" `www.sfb013.uni-linz.ac.at`. The authors would like to thank the referees for their comments which have helped to improve the manuscript.

References

1. H. Blum, Biological shape and visual science, Journal of Theoretical Biology, **45**, pp. 205–287, (1973).
2. D. Cox, J. Little, and D. O'Shea, Ideals, Varieties and Algorithms, Springer, New York, (1997).
3. M. Fioravanti, L. Gonzalez-Vega, On the geometric extraneous components appearing when using implicitization, in: Daehlen, M., Mørken, K., Schumaker, L. (eds.), Mathematical Methods for Curves and Surfaces, Nashboro Press, pp. 157–168, (2005).
4. B. Jüttler, J. Schicho and M. Shalaby, Spline Implicitization of Planar Curves, in: Curves and Surface Design: Saint–Malo 2002, T. Lyche et al., (eds.), Nashboro Press, Brentwood, pp. 225–234, (2003).
5. T. Lyche, and K. Mørken, Knot removal for parametric curves and surfaces, Computer Aided Geometric Design **4**, pp. 217–230, (1987).
6. N. M. Patrikalakis and T. Maekawa, Shape Interrogation for Computer Aided Design and Manufacturing. Springer (2002).
7. H. Pottmann and M. Hofer, Geometry of the squared distance function to curves and surfaces, in: Hege, H., Polthier, K., eds. Visualization and Mathematics III, Springer, pp. 223–244, (2003).
8. S. J. Osher and J. A. Sethian, Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation. Journal of Computational Physics, **79**, pp. 12–49, (1988).
9. U. Reif, Orthogonality Relations for Cardinal B-Splines over Bounded Intervals, in: Geometric Modeling: Theory and Practice (W. Strasser et al., eds), Springer, pp. 56–69, (1998).
10. J. A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, (1999).
11. M. Shalaby, B. Jüttler and J. Schicho, C^1 Spline Implicitization of Planar Curves, in: Automated Deduction in Geometry (F. Winkler, ed.), Springer Lecture Notes in Artificial Intelligence 2930, Springer, Berlin, pp. 161–177, (2004).
12. E. J. Stollnitz, T. D. De Rose and D. H. Salesin, Wavelets for Computer Graphics, Morgan-Kaufman, San Francisco (1996).
13. E. Wurm, J.B. Thomassen, B. Jüttler, T. Dokken, Comparative Benchmarking of Methods for Approximate Parameterization, in: Geometric Modeling and Computing: Seattle 2003 (M. Neamtu u. M. Lucian, Eds.), Nashboro Press, Brentwood 2004, 537–548.
14. J. Wu and L. Kobbelt, Piecewise Linear Approximation of Signed Distance Fields, Vision, Modeling and Visualization 2003 Proceedings, pp. 513–520, (2003).