Dual Evolution of Planar Parametric Spline Curves and T–spline Level Sets

Robert Feichtinger[1], Matthias Fuchs[2], Bert Jüttler[1], Otmar Scherzer[2] and Huaiping Yang[1]

[1]Institute of Applied Geometry, Johannes Kepler University, Linz
[2]Institute of Computer Science, University of Innsbruck

Abstract. By simultaneously considering evolution processes for parametric spline curves and implicitly defined curves, we formulate the framework of dual evolution. This allows us to combine the advantages of both representations. On the one hand, the implicit representation is used to guide the topology of the parametric curve and to formulate additional constraints, such as range constraints. On the other hand, the parametric representation helps to detect and to eliminate unwanted branches of the implicitly defined curves. Moreover, it is required for many applications, e.g., in Computer Aided Design.

1 Introduction

Implicitly defined curves and surfaces, i.e., curves and surfaces which are described as the zero set of a scalar field, have been used, e.g., for geometric modeling [1, 2], for the reconstruction of geometric objects from unorganized points, see [3, 4, 5, 6, 7] and others. Several possible representations of the scalar fields have been explored, such as hierarchical combinations of simpler ones, radial basis functions, spline functions, and grid–based discretizations.

On the other hand, *parametric* curves and surfaces (such as NURBS representations) form the basis of the technology of Computer Aided Design [8]. In particular, the problem of (re–) constructing curves (and surfaces) from given point data has attracted a lot of attention during the last years. Due to artificial parameterization of the data, which is not a part of the described geometry, it produces non–linear optimization problems. Different strategies have been proposed, including 'parameter correction', quasi–Newton methods and geometrically motivated optimization strategies [8, 9, 10, 11, 12, 13, 14, 15, 16].

Since techniques for non–linear optimization rely on iterative methods, it is tempting to view the intermediate results as a time–dependent curve (or surface) which adapts itself to the target shape defined by the unorganized point data [12, 16]. This is similar to the notion of 'active (parametric) curves' which are used for image segmentation in Computer Vision and image processing [17, 18]. In order to perform segmentation, [18] introduced the idea 'active curves' which minimize an energy functional in a space of admissible curves. As shown in [19], this problem can be transformed to the problem of computing a geodesic curve in a Riemannian space with a metric determined by the image data, where solving this problem using the steepest-descent method defines an evolution of the curve.

Another related idea is the use of time–dependent discretizations of (approximations to) the signed distance function in the so–called Level Set method [20, 21]. As the main advantage of this implicit representation, it does not require a parameterization and it naturally adapts the topology during the evolution. Consequently, one may use it to detect complex topological structures, such as objects consisting of multiple components, without using prior knowledge.

This paper combines evolution processes for implicitly defined curves and parametric curves for geometry reconstruction and image segmentation. This leads to a new framework for evolution, which we call the *dual evolution*, since the two representations of geometry are dual to each other.

By simultaneously considering both representations of the geometry, we combine the advantages of the two representations. On the one hand, we obtain a parametric description, which is useful for many applications, e.g., in Computer–Aided Design. On the other hand, we use the implicit representation to identify the correct topology, and in particular to guide the shape of the parametric representations. Moreover, certain *constraints*, such as range or convexity constraints, can more efficiently be formulated in one of the two representations, and they can therefore be added to the framework of dual evolution. For instance, range constraints can be formulated as conditions on the sign of the function defining the implicit representation, as demonstrated in Section 4.

The remainder of the paper is organized as follows. The next section describes evolution process for parametric curves and for implicitly defined curves, and it formulates the framework of dual evolution. Section 3 is devoted to the interaction of the two representations. Section 4 discusses various constraints. Finally we conclude this paper.

2 Dual evolution of planar curves

After describing the idea of evolving or 'active' curves, we formulate them in the cases of parametric curves and implicitly defined curves. Finally, in order to combine the advantages of the two representations, we introduce the framework of dual evolution.

2.1 Evolving curves

Throughout this paper we assume that some data specifying one or more closed planar curves are given. The data, which will be referred to as the "target", can be an unorganized point cloud (e.g., generated by a measurement process), an image (e.g., in a medical application), or another curve (e.g., a polygon).

describing the target curve(s) from the data, by generating both implicitly defined and parametric curves which approximate the point data, or which detect the contours in the given image. In addition, it is possible to specify certain constraints, as will be discussed in Section 4. We assume that the user specifies a feature size ρ, representing the size and the resolution of the geometric objects in the target shape. Various constants that are needed during the evolution process are determined by it.

In order to detect or to reconstruct the geometric information contained in the data, we will consider an evolution process which drives the curve towards the target. More precisely, we consider a time–dependent family of curves $C = C_\tau$, which is sometimes called an "active" curve. The curve is described by certain parameters (e.g., control points or coefficients) which depend on a time variable τ. By continuously modifying these parameters, we move the curve towards its target shape, see Fig. 1. The data is used to derive some information about the expected normal speed of the curve. This will be described in the next section.

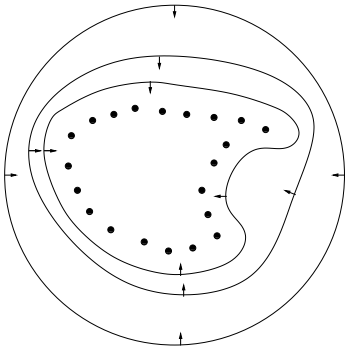


Figure 1: An "active" curve moving towards some data points (3 time steps).

Remark 1 We choose the initial position of the active curve (and similarly for surfaces) such that all data points lie within it. In many cases, the method also works when the initial curve lies within the target or if they intersect each other. We assume that the data contains neither self–intersections nor nested loops. Techniques for handling them are described in [22].

2.2 Speed functions

The evolution of the curve will be guided by the speed (or velocity) function v, which depends both on the curve and on certain geometric information (normals $\vec{\mathbf{n}}$ and the curvature κ) taken from the current instance of the evolving curve.

In the case of *image data* $D = D(x, y)$ we use the function

$$v = e(D)\,(\lambda + \kappa) - (1 - e(D))\,(\vec{\mathbf{n}}^T\,\nabla e(D)), \tag{1}$$

which was proposed in [19], where e is the edge detector function

$$e(D) = e^{-\eta\,|\nabla D|^2}. \tag{2}$$

In this speed function, λ is a constant velocity (also known as the balloon force) and η is a pre-described constant which depends on the range of the data. We refer to the extended version of [23].

In the case of data points, we use

$$v = e(d)\,(\lambda + \kappa) - (1 - e(d))\,(\vec{\mathbf{n}}^T\,\nabla d), \tag{3}$$

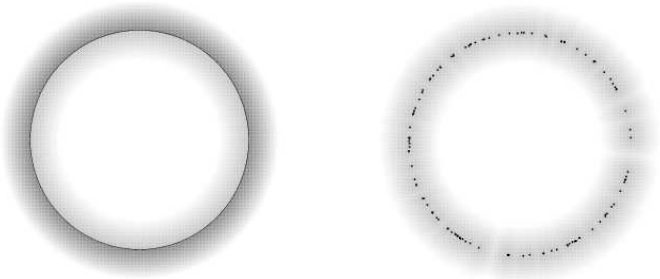
with the edge detector

$$e(d) = 1 - e^{-\eta\,d^2}. \tag{4}$$

Here, d is the unsigned distance function, and η is again a pre-described constant which depends on the range of the data.

Remark 2 The edge detector functions as well as the unsigned distance field will be pre–computed. To determine the unsigned distance field we use graphics hardware acceleration [24]. Therefore, $d(\mathbf{x})$ and $\nabla d(\mathbf{x})$ can be efficiently acquired by linear interpolation of the neighboring grid points. We use the pre–computation in the initialization step of the algorithms which will be described later.

Example 1 Fig. 2 shows two pre–computed distance fields. In the case of a closed curve (left), one may distinguish between interior (light gray) and exterior (dark gray) region. In the case of a point cloud (right), this is no longer possible.



closed curve point cloud

Figure 2: Precomputed distance fields.

2.3 Evolution of parametric curves

We consider a closed parametric spline curve

$$\mathbf{f}(u, \tau) = \sum_{i=1}^{n} B_i(u)\,\mathbf{c}_i(\tau) \tag{5}$$

with B–splines B_i, curve parameter $u \in [0, 1]$, time–dependent control points $\mathbf{c}_i = \mathbf{c}_i(\tau)$, time parameter τ, and uniform periodic knots. The curve is assumed to be C^2 (e.g., a cubic spline curve with single knots).

We shall use the prime $'$ in order to indicate differentiation with respect to the curve parameter u,

$$\frac{\partial \mathbf{f}(u,\tau)}{\partial u} = \mathbf{f}',\ \frac{\partial^2 \mathbf{f}(u,\tau)}{\partial u^2} = \mathbf{f}'', \quad \text{etc.}, \tag{6}$$

while the dot $\dot{}$ represents differentiation with respect to the time parameter τ,

$$\frac{\partial \mathbf{f}(u,\tau)}{\partial \tau} = \dot{\mathbf{f}},\ \frac{\partial \mathbf{c}_i(\tau)}{\partial \tau} = \dot{\mathbf{c}}_i(\tau). \tag{7}$$

with the normal velocity

$$\vec{\mathbf{n}}(u,\tau)\cdot\,\dot{\mathbf{f}}(u,\tau), \tag{8}$$

where $\vec{\mathbf{n}}(u,\tau)$ is the unit normal vector of the curve at $\mathbf{f}(u,\tau)$. This normal velocity is to match the velocity field

$$v = v(\mathbf{f},\mathbf{f}',\mathbf{f}''), \tag{9}$$

see Eqns. (1) and (3), which is determined by the given data and by the current instance of the curve. In order to satisfy this condition approximately, we formulate a least–squares problem

$$E_0(\dot{\mathbf{c}}) = \int_0^1 (\vec{\mathbf{n}}\cdot\dot{\mathbf{f}} - v)^2\,\mathrm{d}u \to \min\,. \tag{10}$$

where $\mathbf{c} = (\mathbf{c}_1,\ldots,\mathbf{c}_n)$ is obtained by gathering all control points into a single vector and $\dot{\mathbf{c}} = (\dot{\mathbf{c}}_1,\ldots,\dot{\mathbf{c}}_n)$ is used for the derivatives of the control points. After replacing the integral by a numerical quadrature with N sample points u_j, we arrive at

$$E(\dot{\mathbf{c}}) = \sum_{j=1}^{N} (\vec{\mathbf{n}}\cdot\dot{\mathbf{f}}_j - v_j)^2 \to \min \tag{11}$$

with

$$\begin{array}{rclcrcl} \mathbf{f}_j & = & \mathbf{f}(u_j,\tau), & & \dot{\mathbf{f}}_j & = & \dot{\mathbf{f}}(u_j,\tau), \\ \vec{\mathbf{n}}_j & = & \vec{\mathbf{n}}(u_j,\tau), & & v_j & = & v(\mathbf{f}_j,\mathbf{f}_j',\mathbf{f}_j''). \end{array} \tag{12}$$

Typically we chose $N = 5\,n$, where n is the number of control points. Finally, by using the B-spline representation for $\mathbf{f}$, this can be rewritten as

$$E(\dot{\mathbf{c}}) = \sum_{j=1}^{N} \left(\vec{\mathbf{n}}_j\cdot\left(\sum_{i=1}^{n} B_i(u_j)\,\dot{\mathbf{c}}_i(\tau)\right) - v_j\right)^2 \to \min\,. \tag{13}$$

The solution $\dot{\mathbf{c}}(\tau)$ of this problem is obtained by solving the sparse linear system with a symmetric positive definite matrix, which is obtained from

$$\frac{\partial}{\partial\dot{\mathbf{c}}_i}E(\dot{\mathbf{c}}) = 0,\ i = 1,\ldots,n. \tag{14}$$

Very efficient algorithms for solving such systems exist [25].

The system (14) defines an ordinary differential equation which specifies an evolution process for the curve. The time derivatives of the control points can be computed from their current values.

Since we are mostly interested in the final position of the evolving curve, but not in the path of the evolution, we integrate the differential equation by an explicit Euler method. The updated control points are chosen as

$$\mathbf{c}(\tau+\Delta\tau) = \mathbf{c}(\tau) + \dot{\mathbf{c}}\Delta\tau. \tag{15}$$

The step size $\Delta\tau$ is chosen as $\min(1, L/v_j,\, j = 1,\ldots,N)$ where L is a user-defined value. This value specifies the maximum allowed displacement of a point in normal direction per iteration step. It should be chosen according to the expected size ρ of the geometry features in the target shape.

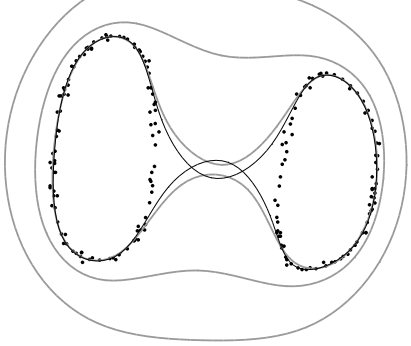


Figure 3: Evolution of a parametric curve towards a point cloud.

Example 2 Several time steps of the evolution of a parametric curve towards a target point clouds consisting of two parts are shown in Fig. 3. The final time step, where the curve reaches a stationary state with two self–intersections, is shown in black.

Remark 3 In order to avoid numerical instabilities, the system (14) has to be regularized. In our implementation, we use a simple Tikhonov regularization, by adding a damping term $\omega||\dot{\mathbf{c}}||$ with a small positive weight ω. See [26] for more information on this type of regularization, in particular concerning the choice of the weight ω.

Remark 4 In the case of given point or curve data, after the evolution reaches the stopping criterion (the norm of $\dot{\mathbf{c}}$ falls below a user–defined threshold), one may improve the solution by solving the following non–linear least-squares problem

$$\sum_{j=1}^{N} |(\mathbf{x}_j - \mathbf{Q}_j)\cdot\vec{\mathbf{n}}_j|^2 \to \min, \tag{16}$$

e.g., by using a Gauss–Newton method, such as the method of normal distance minimization described in [16, 17]. Here $\mathbf{Q}_j$ are the given data points and $\mathbf{x}_j$ is the closest point to $\mathbf{Q}_j$ on the active curve. $\vec{\mathbf{n}}_j$ are the unit normals corresponding to $\mathbf{x}_j$. As observed in [27, 28], this can also be seen as an evolution of a curve, where the normal velocities of the closest points $\mathbf{x}_j$ are equal to the oriented distances to the data.

2.4 Evolution of implicitly defined curves

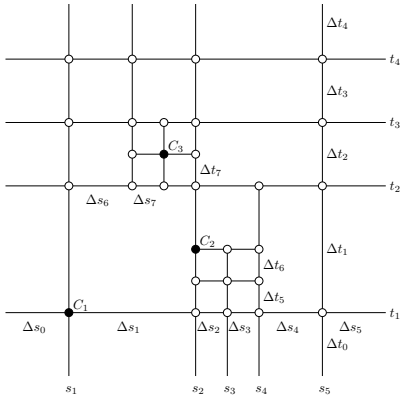
We consider a T–spline (see [29]) of the form

$$g(\mathbf{x},\tau) = \sum_{i=1}^{n} T_i(\mathbf{x})\,c_i(\tau)\ \ \mathbf{x}\in\Omega\subset\mathbb{R}^2, \tag{17}$$

with the bivariate T-spline basis functions T_i and time–dependent real coefficients $c_i = c_i(\tau)$, where the domain Ω is an axis–aligned boxed containing the region of interest. The basis functions

$$T_i(\mathbf{x}) = \frac{B^3_{si}(x_1)B^3_{ti}(x_2)}{\sum_{i=1}^{n} B^3_{si}(x_1)B^3_{ti}(x_2)}$$

certain knot vectors $s_i = (s_{i0},\ s_{i1},\ s_{i2},\ s_{i3},\ s_{i4})$ and t_i which are determined with the help of the so–called T-spline grid (which generalizes the knot vectors of tensor–product splines). This is illustrated by Fig. 4. See [29] for more information.



control points	x-knots y-knots
C_1	$[s_1 - \Delta s_0, s_1 - \Delta s_0, s_1, s_2, s_3]$ $[t_1 - \Delta t_0, t_1 - \Delta t_0, t_1, t_2, t_3]$
C_2	$[s_1 - \Delta s_0, s_1, s_2, s_3, s_4]$ $[t_1, t_1 + \Delta t_5, t_1 + \Delta t_5 + \Delta t_6, t_2, t_2 + \Delta t_7]$
C_3	$[s_1, s_1 + \Delta s_6, s_1 + \Delta s_6 + \Delta s_7, s_2, s_5]$ $[t_1, t_2, t_2 + \Delta t_7, t_3, t_4]$

Figure 4: Top: A T-spline grid. We use 4–fold knots at boundaries. Bottom: The knot vectors for three selected control points.

Since T–splines admits T-junctions, they can be refined locally. Clearly, this is not the case for tensor product B-splines. If the T–spline grid does not contain any T-junctions, then the T-spline simplifies to a tensor–product spline.

The zero level set of the T-spline g,

$$\Gamma(g, \tau) = \{\, \mathbf{x} \in \Omega \subset \mathbb{R}^2 \mid g(\mathbf{x}, \tau) = 0 \,\}, \tag{18}$$

defines a time–dependent planar curve. Similar to the case of a parametric curve, we use the speed function v to derive an evolution process.

Recall that the normal velocity of a point $\mathbf{x}$ of Γ equals $-\dot{g}(\mathbf{x})/|\nabla g(\mathbf{x})|$, where the unit normal vector has been chosen as $\vec{\mathbf{n}} = \nabla g(\mathbf{x})/|\nabla g(\mathbf{x})|$. Similar to (10), we formulate a least squares problem

$$E_0(\dot{\mathbf{c}}) = \int_{\mathbf{x} \in \Gamma(g)} (\dot{g}(\mathbf{x}, \tau) + v\ |\nabla g(\mathbf{x}, \tau)|\ )^2\ \mathrm{d}s \to \min \tag{19}$$

where s represents the arc length of the T–spline level set, and $\mathbf{c} = (c_1, \ldots, c_n)$. The value of the speed function depends on the point $\mathbf{x} \in \Gamma$ and on the first and second derivative of the T–spline g at $\mathbf{x}$. Again, we use numerical integration in order to approximate the integral,

$$E(\dot{\mathbf{c}}) = \sum_{j=1}^{N} (\dot{g}(\mathbf{x}_j, \tau) + v(\mathbf{x}_j, \tau)\ |\nabla g(\mathbf{x}_j, \tau)|)^2 \to \min \tag{20}$$

the zero level set, where $N \gg n$.

The initial T-spline g is chosen as an approximation to the signed distance function of its zero level set. During the evolution, g will gradually loose this property, which is characterized by $|\nabla g| = 1$. Most existing level set evolutions use a re-initialization step to restore the signed distance property, e.g., by using a Fast Marching technique [30]. Following ideas described in [23, 31, 32], we avoid the re-initialization by introducing a *distance field constraint*.

More precisely, we add the constraint term

$$S_0 = \int_{\Omega} (\frac{\partial\, |\nabla g(\mathbf{x}, \tau)|}{\partial \tau} + |\nabla g(\mathbf{x}, \tau)| - 1)^2 \mathrm{d}\mathbf{x} \to \min \tag{21}$$

as a penalty function, which penalizes the deviation of g from a signed distance function. Again, we use numerical integration (but now with sample points distributed in Ω, and not just on the zero level set) in order to derive a discretized version S of this constraint term. Typically we use 25 uniform distributed sample points per cell of the T-spline grid.

For each step of the T-spline evolution the time derivatives $\dot{\mathbf{c}}(\tau)$ are computed by minimizing the weighted linear combination

$$F(\dot{\mathbf{c}}) = E(\dot{\mathbf{c}}) + \omega_s\ S(\dot{\mathbf{c}}) \to \min, \tag{22}$$

with a certain positive weight ω_s. Similar to the parametric case, this results in a sparse symmetric positive definite linear system defining an evolution of the curve. Once again we use explicit Euler steps in order track the evolution path. More details, including information concerning the choice of the weight ω_s, the discretization of the signed distance constraint term, and the selection of the T–spline grid, have been presented in [23]. An example for the adaptive choice of the T-spline grid will be given later (Example 7).

2.5 Dual evolution

On the one hand, a parametric spline representation of the curve is needed, e.g., in Computer Aided Design. However the evolving parametric curves have some difficulties to deal with changes of the topology, i.e., with targets which consisting of more than one component.

On the other hand, an implicit representation is clearly not a standard representation. Moreover, it may produce additional branches during the evolution. However, as a major advantage, it is able to adapt its topology to the target in a natural way. This is one of the main reasons for the increasing popularity of the level–set method. Additionally we need sample points on the zero–level to solve the evolution equation. These sample points can be provided by the parametric curve.

Consequently, it is a natural idea to combine the two evolution processes for the two representations. We propose the following algorithm for what we call "dual evolution":

Algorithm 1

1. *Initialization:* Pre-compute the evolution speed function and choose initial position of both curves.

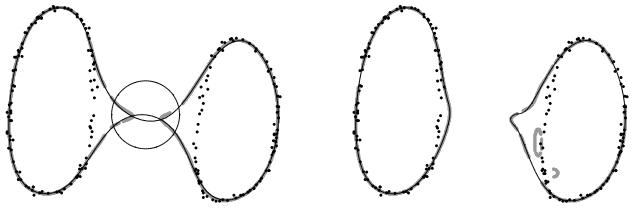
ric curves for one time step.

3. *Synchronization:* Detect and deal with occurring problems, such as additional branches and topological changes, and ensure that the two representations stay close. See section 3.

4. *Termination:* Check whether the stopping criterion is satisfied, cf. Remark 4. Continue with step 2 (no) or 5 (yes).

5. *Refinement* of the parametric curve, see Remark 4.

Example 3 We continue the previous example. Fig. 5, left, shows again the self intersection. Now we use the dual evolution, which combines a parametric spline curve (black) and an implicitly defined one (grey). By combining these two representations, we may now adapt the topology of the spline curve and split it into two components (right). At the same time, the implicitly defined curve develops two phantom branches.



before splitting after splitting

Figure 5: Adapting the topology by dual evolution of an implicitly defined (grey) and a parametric curve (black).

Remark 5 Following the assumptions made in Remark 1 about the target shape, and if we assume that the initial position of the active curve enclosed all the data, then only one type of topological changes is possible, namely splitting events. The methods of this paper can easily be extended to the deal with merging events. For instance, these events may occur if the initial position of the target curve does not enclose all data. Note that more sophisticated methods are needed in oder to deal with targets possessing self-intersections, see [22].

Remark 6 Instead of coupling the evolutions of implicitly defined and parametric curves, one may first use the implicit evolution to capture all components of the target, and then approximate the different branches of the implicit curve by an parametric representation. Two additional difficulties arise when following this approach. First, it requires a topology analysis of the implicitly defined curve, see e.g., [33, 34]. Second, after generating sample points on the various branches, they have to be approximated by parametric spline curves. Various fitting techniques are available [10, 16]. While these two problems are certainly solvable, they would become even more challenging in the case of surfaces. By coupling the two representations, we obtain an "all at once" approach. In addition, certain constraints, such as range constraints, can much easier be formulated for implicitly defined curves.

The section is devoted to the synchronization step of the algorithm for dual evolution. Firstly we discuss the detection of possible topological changes (splitting events), secondly the synchronization in the case of no changes, and finally the adaptation of the parametric curve in the case of topological changes.

3.1 Detection of topological changes

We describe and compare three different approaches.

Method 1: Self–intersections on the parametric curve

This method does not use any information from the implicitly defined curve. Instead, it simply tries to detect self–intersections of the parametric curve via sampling. More precisely, we approximate the parametric curve by an inscribed polygon and check for self–intersections.

Method 2: Comparing normals

After each evolution step one may compare the unit normal vectors of the parametric curve $\vec{\mathbf{n}}_{\mathbf{f}}$ and of the implicitly defined curve $\vec{\mathbf{n}}_g$. More precisely, we may define a unit normal $\vec{\mathbf{n}}_g = \nabla g/|\nabla g|$ for almost *all* points in the domain (except for points with vanishing gradients), not only for points on the zero level set Γ.

In our experiments, we observed that the following two events are closely related:

(1) The implicit curve has changed its topology.

(2) There exists a parameter value u_j such that

$$\vec{\mathbf{n}}_{\mathbf{f}}(u_j) \cdot \vec{\mathbf{n}}_g\left(\mathbf{f}_j\right)) \leq 0. \tag{23}$$

This observation allows us to detect self–intersections without explicitly computing them. If N sample points are used, then the complexity is $\mathcal{O}(N)$.

This observation is justified by the following simple result (see Fig. 6 for an illustration).

Lemma 1 *Consider a subdomain $S \subset \Omega$ with boundary ∂S, which is assumed to be contained in another open subset $D \subseteq \Omega$. We assume that the C^1 function g has no local extrema[1] in D. We assume that the boundary ∂S consists of segments of the parametric curve $\mathbf{f}$, where all normals $\vec{\mathbf{n}}_{\mathbf{f}}$ are either pointing away or pointing towards S. In addition, we assume that g is not constant. Then the sign of the inner product*

$$\nabla g \cdot \vec{\mathbf{n}}_{\mathbf{f}} \tag{24}$$

changes on ∂S, or it everywhere equals 0 on ∂S.

[1]Here, a point $\mathbf{p}$ is said to be a local maximum (and similar for a local minimum) of g on D if $g(\mathbf{p}) \geq g(\mathbf{x})$ holds for all points $\mathbf{x}$ in an open neighborhood N of $\mathbf{p}$. This includes points where the value of g is constant in N.

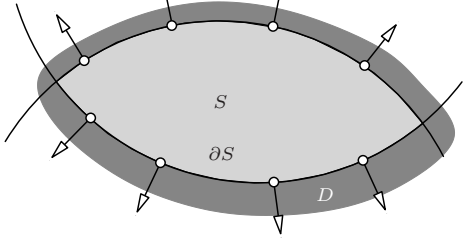


Figure 6: The assumptions of Lemma 1.

Proof: We assume that the normals $\mathbf{n_f}$ point away form S, as shown in Fig. 6. Since $g : \Omega \to \mathbb{R}$ is continuous and $S \subseteq \Omega$ is compact, the restriction of g to S takes its maximum and minimum values at two points $\mathbf{x}_{\max}, \mathbf{x}_{\min} \in S$, respectively. As we assumed that g has no local extrema on the open set D and $S \subset D$, these two points belong to the boundary ∂S. We consider the minimum of g on $\mathbf{S}$ which is attained at $\mathbf{x}_{\min} \in \partial S$.

Case 1: $\mathbf{x}_{\min}$ is a regular point of ∂S, see Figure 7, left. Let $\mathbf{n}$ be the normal of ∂S at this point. As $\mathbf{x}_{\min}$ is the global minimum of g on S, it is also the global minimum of the restriction of g to the boundary ∂S. Using standard arguments from constrained optimization (see e.g. [35]) we conclude that the gradient vector ∇g and the normal vector of ∂S at $\mathbf{x}_{\min}$ are linearly dependent. In addition, the directional derivative of g in the direction of the outward-pointing normal $\mathbf{n_f}$ is non-positive,

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{n_f}} = \nabla g(\mathbf{x}_{\min}) \cdot \mathbf{n} \leq 0. \tag{25}$$

Case 2: $\mathbf{x}_{\min}$ is a vertex of ∂S, i.e., a double point of the parametric curve $\mathbf{f}$, see Figure 7, right. Let $\mathbf{n}_1, \mathbf{n}_2$ be the normals of the two branches of ∂S at this point. Again, using standard arguments from constrained optimization, we conclude that the gradient ∇g at $\mathbf{x}_{\min}$ lies in the intersection of the two half–planes

$$\nabla g(\mathbf{x}_{\min}) \cdot \mathbf{n}_1 \leq 0, \qquad \nabla g(\mathbf{x}_{\min}) \cdot \mathbf{n}_2 \leq 0 \tag{26}$$

which are shown by the dashed lines in the figure.

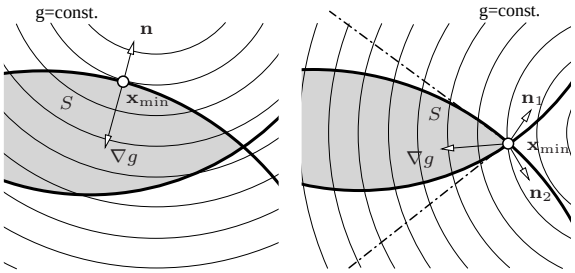


Figure 7: Conditions for the minimum of g on S. Case 1 (left) and Case 2 (right).

Similarly we may conclude that

$$\nabla g(\mathbf{x}_{\max}) \cdot \mathbf{n} \geq 0, \tag{27}$$

consider the inner products

$$\nabla g(\mathbf{x}) \cdot \mathbf{n_f}(\mathbf{x}) \geq 0, \quad \mathbf{x} \in \partial S, \tag{28}$$

where $\mathbf{n_f}(\mathbf{x})$ is the normal of the parametric curve $\mathbf{f}$ at the point $\mathbf{x} \in \partial S$. We consider this as a function on the union of the arcs that form the boundary ∂S, where each vertex appears twice[2]. Again, this is a continuous function on a compact set, and it therefore has a minimum and a maximum. Due to (25) resp. (26), the minimum is non-positive, and due to (27), the maximum is non-negative. This completes the proof. □

Note that the assumption concerning the non–existence of local extrema is likely to be satisfied by the domain enclosed by two branches of a self–intersecting curve, such as the black curve in Fig. 3. On the one hand, the global distribution of the normals of the parametric curve entails that they point either towards S or away from S. On the other hand, the function defining g the curve Γ is likely to have a saddle point, but not an extremal point, in this region.

Remark 7 In practice we replace the right–hand side in (23) with a small positive constant ε, in order to make the criterion more sensitive.

Method 3: Distance check

Finally we may check whether the parametric curve $\mathbf{f}$ and the implicitly defined curve Γ are "sufficiently close" to each other. More precisely, for each point $\mathbf{f}(u)$ of the parametric curve we try to find the corresponding point on Γ, by intersecting the normal with Γ,

$$g(\mathbf{f}(u) + \mu(u)\, \vec{\mathbf{n}}_f(u)) = 0. \tag{29}$$

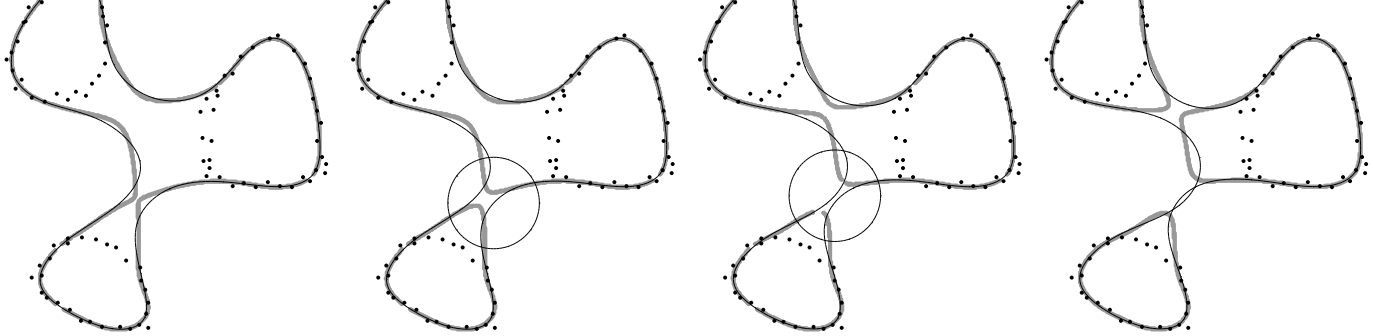
By differentiation we obtain a differential equation for μ,

$$\mu' = -\frac{\nabla g \cdot (\mathbf{f}' + \mu\, \vec{\mathbf{n}}')}{\nabla g \cdot \vec{\mathbf{n}}} \tag{30}$$

Using a predictor–corrector method we trace the parametric curve and the corresponding points on the implicitly defined curve. If the corrector (a Newton method for root finding along the normal) changes the predicted value of μ too much, or even fails to find a corresponding point, or if the distance between the point $\mathbf{f}(u)$ and its corresponding point on Γ exceeds a certain threshold, then we report that a change of topology is likely. For this threshold we use the feature size ρ.

Remark 8 In order to speed up the computation one may instead simply check whether the sign of g changes in a tubular ϵ–neighborhood around the parametric curve. We again use $\epsilon = \rho$. If the zero level is close to the parametric curve, then the sign changes in this neighborhood.

[2]We may get two different values for each vertex, therefore we choose this union as the domain of this function.



Initial position | Step 1: Method 3 detects change | Step 4: Method 2 detects change. | Step 7: Method 1 detects change.

Figure 8: Detection of topological changes.

Comparison

The three methods have been implemented and tested. The first method (Section 3.1) is rather time–consuming, in particular if a large number of sampled points is used. Furthermore, this condition does not guarantee that the topology of the implicit curve has changed as well. Moreover, topological changes are detected relatively late, as will be demonstrated by Example 4. On the other hand not using the implicit curve helps in some cases where the T-spline has a very flat shape. In this case, the other two methods have problems.

The third method detects the changes as soon as possible, but with more computational effort. The second method can be seen as a compromise. Methods 2 and 3 have problems if the implicit curve is very flat, since then the function g does not represent the curve well.

Note that it is not a serious problem if the method is too sensitive, i.e., if it reports topological changes if no such events have actually taken place. As we will see later, the method for modifying the topology (see Section 3.3) will adapt the topology of the parametric curve to the current shape of the implicitly defined one, and it will identify cases where no change of topology took place.

We apply the three methods to an example:

Example 4 Fig. 8 shows the dual evolution of a curve which experiences a change of its topology. The implicitly defined curve is shown in grey and the corresponding parametric curve in black. The third method (distance check) is the first one to detect the topological change in the first time step (top right). Four time steps later, the second method (comparing normals) reports the change (bottom left). Finally, after three more time steps, the parametric curve develops a self–intersection, which is then duly reported by the first method (bottom right).

3.2 Synchronization without topological changes

If no change of the topology has been reported, then we try to make sure that the two representations of the curve stay close to each other. Two possibilities exist:

Fitting the implicitly defined curve to the parametric one

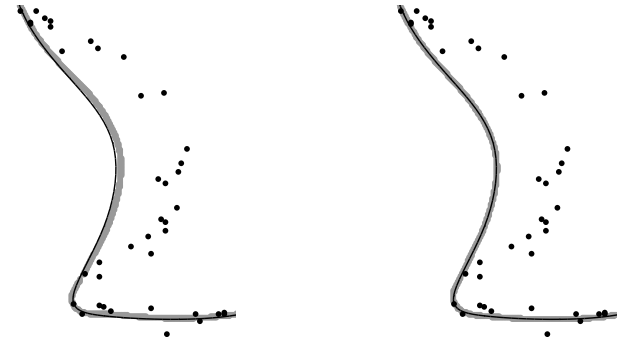
In most cases, we fit the implicitly defined curve to the parametric one, by solving a least-squares problem

$$\sum_{j=1}^{N} e^{-\eta\phi(\mathbf{y}_j)} \left(g(\mathbf{y}_j) - \phi(\mathbf{y}_j)\right)^2 \rightarrow \min . \tag{31}$$

The sample points $\mathbf{y}_j$ are uniformly distributed in the domain Ω of g. The function ϕ is the signed distance field of the parametric spline curve (again obtained using graphics hardware) and η is a constant as defined in (Section 2.2).

We chose this approach because it allows us to eliminate additional branches of the implicit representation and it guarantee that the two representations of the curve stay close to each other.

Example 5 Fig. 9 shows an example.



before synchronization | after synchronization

Figure 9: Synchronization without topological changes.

Fitting the parametric curve to the implicitly defined one

In some cases, e.g., if additional constraints acting on the implicitly defined curve are used (see Section 4), the implicitly defined curve takes the leading role. For a sequence of uniformly distributed sample points on the parametric curve we create the closest points on Γ. Then we solve again a linear–least–squares problem, in order to fit the parametric curve to them.

If a change of topology has been reported, we have to create a new parametric curve with the the correct topology. We use the implicit curve to guide this process. More precisely, if $\mathbf{p}_i$ is a point which has been identified by one of the three methods in Section 3.1, then we apply the following algorithm.

Algorithm 2

1. Create a circle C with a predefined radius (which should be chosen in dependence of the feature size, for the examples we used 20ρ) around $\mathbf{p}_i$.

2. Compute the set P of intersections between C and the parametric curve $\mathbf{f}$. Compute the set I of intersections between C and the implicitly defined curve Γ. In order to compute I, the circle C is represented as a parametric quadratic spline curve. On the other hand, in order to compute P, its parametric representation is used. In both cases this leads to root–finding problems in one variable.

3. Check if $|I| = |P|$. For each $\mathbf{x} \in I$ find the nearest $\mathbf{y} \in P$. This should define a one–to–one correspondence between the points of P and I. If this fails, we increase the radius of the circle C and continue with step 2.

4. Trace the implicitly defined curve within C between its intersection points in I and use this information to create new parametric spline curves between the corresponding points of P.

Some steps of the algorithm will now be discussed in more detail:

In step 2, if $|P| = |I|$ then for every $\mathbf{x} \in P$ there exists a point $\mathbf{y} \in I$ which is close to $\mathbf{x}$, due to the synchronization step in algorithm 1. Otherwise, if $|P| \neq |I|$ we may increase the radius of C. Alternatively, one may filter out the unused points in I later. In order to compute the intersections between the parametric curve, the implicitly defined curve and the circle we approximate both curves by polygons and compute their intersections. Alternatively one may formulate both tasks as univariate polynomial root-finding problems and use methods such as Bézier-clipping, cf. [36].

Clearly, it is essential to identify the correct radius of the circle C. If the radius is too small, then we may not find all informations we need. One the other hand, if we choose the radius too large, then we may lose some parts of the target. There will be no perfect fully automatic solution to this problem. In the algorithm we use a binary search strategy to determine the radius.

In order to trace the implicitly defined curve within C (step 4), we use a predictor–corrector method [8] with a curvature–dependent step–size control. This tracing should establish pairs of intersection points. If this fails, then we increase the accuracy of the tracing method (i.e., we decrease the step size).

In order to create the new parametric spline curve (step 4), we split it at its intersections with the circle C and fill in new segments. The control points of the new segments are obtained by uniformly distributing points on the corresponding segments of the implicit curve. Alternatively, one might try to fit another B-spline curve to the traced segment, but the result of the simpler method are sufficient.

arbitrary number of branches during each event. While it is very unlikely that one curve splits in more than two branches, it can easily happen that several branches get close to each other. See the following example.

Example 6 Fig. 10 (top) shows a complicated example which can be handled by our method. A curve evolves towards a target which consists of four pieces. In one step, the parametric curve has to split into four components.

We conclude this section by another example.

Example 7 Fig. 10 (bottom) shows some steps of the evolution process towards a target defined by two point clouds. In addition to the data and the curves, the figures visualize the T-spline grid, which is refined in the vicinity of the data.

In this example, the parametric curve starts with 14 control points in the beginning and increases this number to 17 after the splitting step. The T-spline is defined by 160 coefficients. One step of the evolution of the parametric curve needs less than 1 milliseconds, and one step of the evolution of the implicitly defined curve requires about 60 milliseconds. Most of the computation time is needed for the synchronization: 300 milliseconds without splitting, and 700 milliseconds with splitting.

Reducing the number of T-spline coefficients also reduces the time per iteration. E.g. with 86 coefficients we need about 150 milliseconds per time step. The final result is reached after 40 evolution steps. For very simple objects like a circle with 12 B-spline control points and 25 T-spline coefficients one evolution step can be done in 30 milliseconds.

4 Constraints

We present three different constraints which can be applied to the evolution. The range constraints are used to define regions which should lie inside or outside of the final curve. The area constraints force the curve to enclose a certain pre-defined area. Finally the convexity constraint allows us to define a region where the active curve becomes convex.

4.1 Range constraints

The implicitly defined active curve decomposes the domain into an *inner* region, where $g(\mathbf{x}) \leq 0$, and an *exterior* region, where $g(\mathbf{x}) \geq 0$. (Note that one of these regions can be empty, though this does not make sense in our framework.) Based on this observation one may add *range constraints* to the the framework of dual evolution. More precisely, if the domain bounded by the final curve should contain a certain set of points $\{\mathbf{x}_i\}_{i=0,\ldots,N_0}$, and it should not contain another set of points $\{\mathbf{y}_j\}_{j=0,\ldots,N_1}$, then we have to ensure that the evolving implicitly defined curve satisfies

$$g(\mathbf{x}_i) \leq 0, \quad \text{and} \quad g(\mathbf{y}_j) \geq 0, \tag{32}$$

respectively.

In the first case, this can be achieved by adding a penalty term to the objective function (22) which implies that the time derivative satisfies $\dot{g}(\mathbf{x}_i) < 0$ if the function value $g(\mathbf{x}_i)$ is positive.

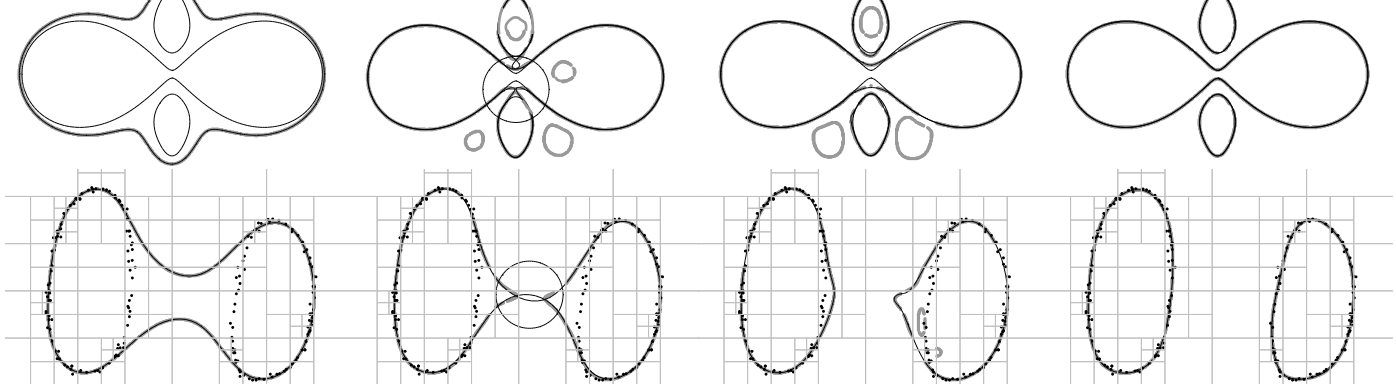


Figure 10: Synchronization with topological changes. The target is defined by a smooth curve (top) or by a point cloud (bottom).

(The second case can be dealt with similarly.) We propose to use

$$C(\dot{\mathbf{c}}) = \sum_{j=1}^{N_0} \left(\dot{g}(\mathbf{x}_j, \tau) + g(\mathbf{x}_j, \tau) + \delta\right)^2 \alpha_\varepsilon(g(\mathbf{x}_j, \tau)) \qquad (33)$$

where δ is chosen by the user (see below for examples). The 'activator' function α controls the influence of this term,

$$\alpha_\varepsilon(g) = \begin{cases} 1 & g > -\varepsilon \\ 0 & g < -2\varepsilon \\ C^2 - \text{blend} & \text{in between} \end{cases} \qquad (34)$$

where ε is a user–defined positive constant (e.g., the feature size ρ can again be used). The optimization problem

$$\widehat{F}(\dot{\mathbf{c}}) = E(\dot{\mathbf{c}}) + \omega_s\, S(\dot{\mathbf{c}}) + \omega_c\, C(\dot{\mathbf{c}}) \to \min . \qquad (35)$$

leads to a sparse linear system of equations with a symmetric positive definite matrix, which can be dealt with efficiently.

The constraint term acts only on the implicitly defined curve, but not on the parametric one. However, the parametric curve 'inherits' the constraint through the synchronization, provided that the second synchronization method (guided by the implicitly defined curve) is used.

Depending on the choice of δ, the evolution stops at a certain offset of the given shape. We will demonstrate this by several examples. In all examples, the points on the target are simultaneously used to define the constraints. More precisely, we look for approximating curves which are circumscribed or inscribed to the given data.

Example 8 In Fig. 11, the target is defined by a noisy point cloud taken from a circle. We show the approximation without constraints (left) and the approximation (right) obtained by adding the constraint term (33) with $\delta = -0.5$.

Example 9 We consider the two branches of the curve defined by the implicit equation

$$((x - \frac{3}{4})^2 + y^2)\,((x + \frac{3}{4})^2 + y^2) = 0.316. \qquad (36)$$

Fig. 12 shows the dual evolution for a target defined by this curve. Depending on the choice of δ, we obtain offsets of the algebraic curve.

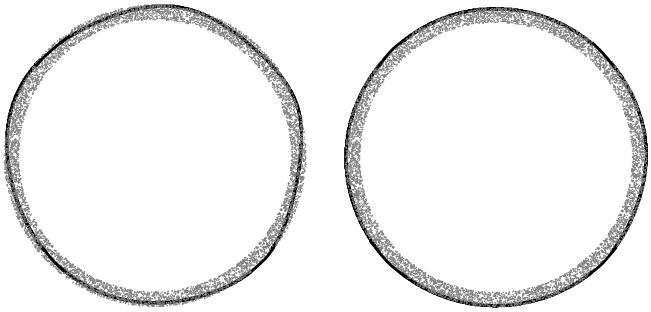


Figure 11: Approximation with (right) and without (left) range constraints.

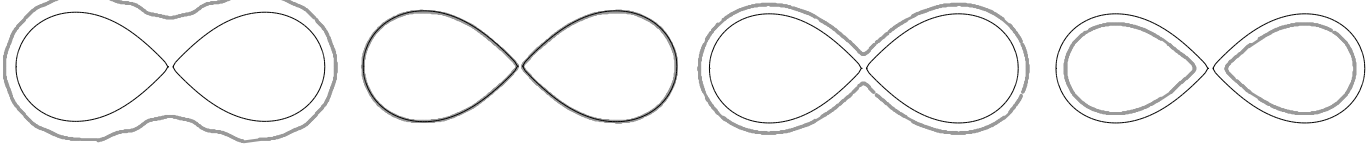
Example 10 In this final example (see Fig. 13) we consider a target which consists of three parts. Evolution without constraints produces three curves lying within the data set. For $\delta = -0.5$ we obtain curves which represent outer boundaries of the data set.

Remark 10 In the case of parametric curves, several approaches to range constraints exist. For instance, a tension–based technique to constrained interpolation by parametric spline curves is described in [37], the use of tight piecewise linear enclosures have been proposed in [38], and certain optimization techniques are explored in [39].

4.2 Area constraints

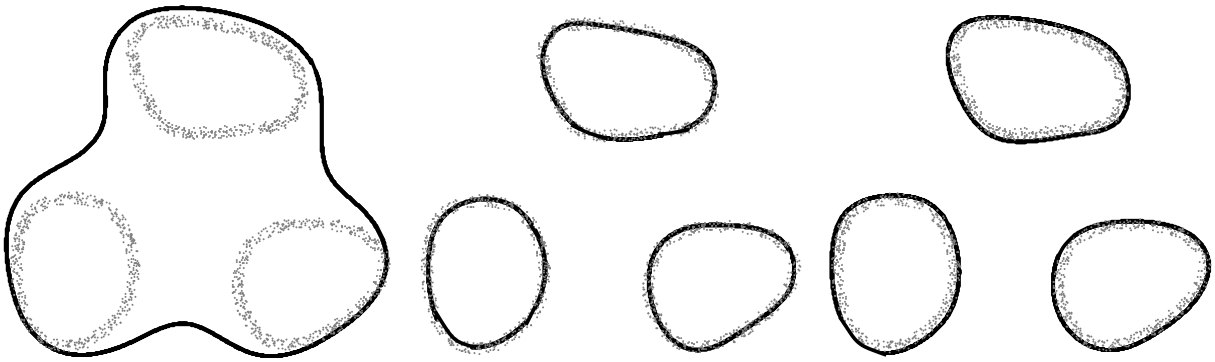
In some applications, e.g. when dealing with noisy data containing holes, one may wish to specify the area of the target object. Starting from an initial value, the current area A_C enclosed by the curve should adjusted until the final area A_F, which is specified by the user, has been reached. This can be achieved by adding a constraint of the form

$$\int_\Gamma v_n \mathrm{d}s = k \qquad (37)$$



initial position $\delta = 0$ $\delta = -0.05$ $\delta = +0.05$

Figure 12: Dual evolution with range constraints for a target defined by an algebraic curve.



initial position result without constraints with constraint, $\delta = -0.5$

Figure 13: Dual evolution with range constraints for a target defined by three point clouds.

to the least squares problems (11) resp. (19), where

$$v_n(u,\tau) = \vec{\mathbf{n}}(u,\tau) \cdot \dot{\mathbf{f}}(u,\tau) \quad \text{resp.} \quad v_n(\mathbf{x},\tau) = -\frac{\dot{g}(\mathbf{x},\tau)}{|\nabla g(\mathbf{x},\tau)|}$$

is the normal velocity of the parametric / implicitly define curve, respectively. s is the arc length of the curve, and k is the rate of area change (with respect to the time τ),

$$k = \begin{cases} \frac{A_C}{A_F} - 1 & A_C > A_F \\ -\frac{A_F}{A_C} + 1 & \text{otherwise} \end{cases}$$

The constraint (37) is linear in the time derivatives of the control points resp. T-spline coefficients. Again we use numerical integration to obtain a linear constraint, which is to be considered along with the quadratic objective function. This leads to a quadratic optimization problem with linear constraints, which is solved using Lagrangian multipliers.

Example 11 We consider a set of data points sampled from a square after removing the top edge, see, see Fig. 14. The area of the square equals 0.5. The figure shows the result of the dual evolution obtained by specifying different values of the target area A_F. If the correct value is specified, then the dual evolution recovers the original shape. In this example, the area constraint was added to the T-spline level set evolution, and it was inherited by the parametric curve via synchronization.

4.3 Convexity constraints

Similar to range constraints one may also add convexity constraints to the framework of dual evolution. In the following we assume that the user has specified a region Ω_d where the curve is to be convex. This can be dealt with by adding a penalty term of the form

$$D(\dot{\mathbf{c}}) = \int_{\Gamma \cap \Omega_d} (\dot{\kappa}(\mathbf{x},\tau) - |\kappa(\mathbf{x},\tau)| - \sigma)^2 \beta_\sigma(\kappa(\mathbf{x},\tau)) \mathrm{d}\mathbf{x} \quad (38)$$

to the quadratic objective function, which is again evaluated via numerical integration. If the curve is concave, i.e., if $\kappa(\mathbf{x}) < 0$, then the time derivative of the curvature is forced to be positive, $\dot{\kappa}(\mathbf{x}_i) > 0$, thereby increasing the curvature until it gets positive.

Similar to (34), the 'activator' function β_σ controls the influence of the penalty term,

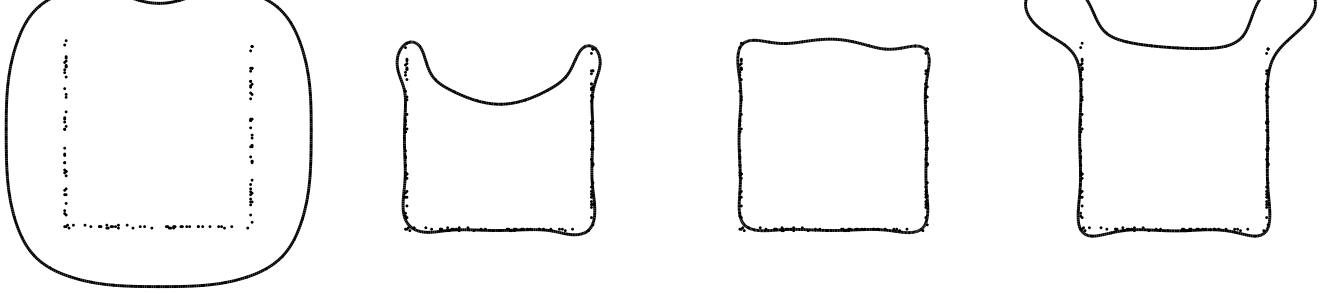
$$\beta_\sigma(\kappa) = \begin{cases} 1 & \kappa < 0 \\ 0 & \kappa > \sigma \\ C^2 - \text{blend} & \text{in between} \end{cases} \quad (39)$$

where σ is a user-defined positive constant. A larger value of σ increases the curvature of the curve segment in the constrained region Ω_d. By incorporating the convexity constraint, the optimization problem becomes

$$\widetilde{F}(\dot{\mathbf{c}}) = E(\dot{\mathbf{c}}) + \omega_s S(\dot{\mathbf{c}}) + \omega_d D(\dot{\mathbf{c}}) \to \min. \quad (40)$$

Since $\dot{\kappa} = \sum_{i=1}^{n} \dot{c}_i \partial\kappa/\partial c_i$, which means $D(\dot{\mathbf{c}})$ is also quadratic to $\dot{\mathbf{c}}$. Hence the solution to (40) can be found by solving a least squares problem. Usually we choose a very large weighting coefficient ($\omega_d = 1000$) in our experiments.

Example 12 In Fig. 15, the target shape (defined by unorganized points) has some concave features. On the left, we show the approximation result without convexity constraints. On the right, we show the approximation result obtained by applying the convexity constraints ($\sigma = 0.05$). The region Ω_d is the box which is shown in the figure.



initial position | volume set to 0.4 | volume set to 0.5 | volume set to 0.6

Figure 14: Dual evolution with volume constraints.

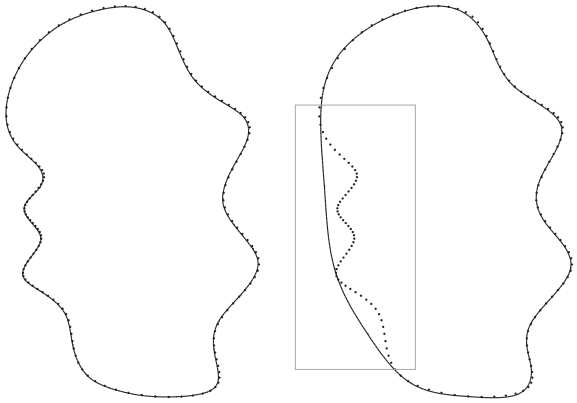


Figure 15: Approximation with (right) and without (left) convexity constraints.

5 Concluding remarks

We formulated the novel framework of dual evolution, by simultaneously considering evolution processes for parametric spline curves and implicitly defined curves. As the main advantage of this framework, it combines the advantages of both representations. On the one hand, the implicit representation is used to guide the topology of the parametric curve and to formulate additional constraints, such as range constraints, volume constraints or curvature constraints. On the other hand, the parametric representation helps to detect and to eliminate unwanted branches of the implicitly defined curves in the synchronization step. Clearly, the parametric representation is preferred in many applications, e.g., in Computer Aided Design.

Currently we study the extension of the results to surfaces, where we need to define evolution processes and to discuss the interaction of the two representations. The evolution of T–spline Level sets in 3D has already been discussed in [23]. The application to practical problems, such as reverse engineering, has produced promising results [40]. However, the choice of a suitable surface representation is still open, and we are currently considering triangular meshes, manifold splines, or point–set surfaces.

Acknowledgments. The financial support of the Austrian Science fund through the Joint Research Programme S92 'Industrial Geometry', subproject 2, and by the European Union through the Marie Curie IIF Fellowship for Huaiping Yang (project 22073 ISIS) is gratefully acknowledged. The authors thank the referees for their suggestions.

References

[1] Cartwright R., et al. Web-based shape modeling with HyperFun. IEEE Computer Graphics and Applications 2006; 25:60-9.

[2] Raviv A, Elber G. Three dimensional freeform sculpting via zero sets of scalar trivariate functions. In Proc 5th ACM Symposium on Solid Modeling and Applications; ACM; 1999, p. 246-57.

[3] Carr JC, etal. Reconstruction and representation of 3D objects with radial basis functions. In Proc. SIGGRAPH'01, 2001, p. 67–76.

[4] Jüttler B, Felis A. Least–squares fitting of algebraic spline surfaces. Adv Comput Math 2002; 17: 135-52.

[5] Ohtake Y, et al. Multi-level partition of unity implicits. ACM Transactions on Graphics (Proc. Siggraph) 2003; 22: 463-70.

[6] Yang ZW, Deng JS, Chen FL. Fitting unorganized point clouds with active implicit B-spline curves. The Visual Computer 2005; 21: 831-9.

[7] Zhao HK, Osher S, Fedkiw R. Fast surface reconstruction using the level set method. In Proc 1st IEEE Workshop on Variational and Level Set Methods in Computer Vision, IEEE; 2001, p. 194-201.

[8] Hoschek J, Lasser D, Fundamentals of Computer Aided Geometric Design, Wellesley: AK Peters; 1996.

[9] Alhanaty M, Bercovier M, Curve and surface fitting and design by optimal control methods, Computer-Aided Design 2001; 33: 167-82.

[10] Speer T, Kuppe M, Hoschek J. Global reparametrization for curve approximation. Computer Aided Geometric Design 1998; 15: 869-77.

[11] Pottmann H, Leopoldseder S. A concept for parametric surface fitting which avoids the parametrization problem. Computer Aided Geometric Design 2003; 20: 343-62.

[12] Pottmann H, Leopoldseder S, Hofer, M. Approximation with active B-spline curves and surfaces. Proc. Pacific Graphics IEEE; 2002, p. 8–25.

[13] Pottmann H, et al. Industrial geometry: recent advances and applications in CAD. Computer-Aided Design 2005; 37: 751-66.

[14] Rogers DF, Fog NG. Constrained B-spline curve and surface fitting. Computer Aided Design 1989; 21: 641-8.

[15] Sarkar B, Menq CH. Parameter optimization in approximating curves and surfaces to measurement data. Computer Aided Geometric Design 1991: 8, 267-80.

[16] Wang W, Pottmann H, Liu Y. Fitting B-spline curves to point clouds by squared distance minimization. ACM Transactions on Graphics 2006; 25: 214-38.

[17] Blake A, Isard M, Active contours, Springer; 2000.

[18] Kass M, Witkin A, Terzopoulus D. Snakes: active contour models, Int. J. of computer vision 1998; 1: 231-3.

[19] Caselles V, Kimmel R, Sapiro G. Geodesic active contours, Int. J. of computer vision 1997; 22: 61-79.

[20] Osher S, Sethian J, Fronts propagating with curvature dependent speed, algorithms based on a Hamilton-Jacobi formulation. J. Comp. Phys. 1988; 79: 12-49.

Springer; 2003.
[22] Yang H, Jüttler B, Gonzalez-Vega L. An evolution-based approach for approximate parameterization of implicitly defined curves by polynomial parametric spline curves. 2006: FSP report no. 28, available at www.ig.jku.at.
[23] Yang H, Fuchs M, Jüttler B, Scherzer O. Evolving T-spline level sets. In: Shape Modeling and Applications 2006, IEEE; 2006, p. 247-52. Extended version available as FSP report no. 1 (2005) at www.ig.jku.at.
[24] Hoff KE, Culver T, Keyser J, Lin M, Manocha D. Fast computation of generalized Voronoi diagrams using graphics hardware. Proc. SIGGRAPH'99, 1999: p. 277-86.
[25] Botsch M, Bommes D, Kobbelt L, Efficient linear system solvers for mesh processing. In Martin R, et al, editors. The Mathematics of Surfaces XI, volume 3604 of LNCS. London: Springer; 2005: 62-83.
[26] Engl H, Hanke M, Neubauer A, Regularization of inverse problems. Dordrecht: Kluwer; 1996.
[27] Aigner M, Šír Z, Jüttler B, Least-Squares Approximation by Pythagorean Hodograph Spline curves via an Evolution Process. In Kim M-S, Shimada K, editors. Geometric Modelling and Processing, Springer LNCS 4077; 2006, p. 45-58.
[28] Aigner M, Jüttler B. Hybrid curve fitting, Computing 2007; 79: 237-47.
[29] Sederberg TW, Zheng J, Bakenov A, Nasri A. T-splines and T-NURCCs ACM Transactions on Graphics 2003; 22: 477-84.
[30] Sethian J, A fast marching level set method for monotonically advancing fronts, Proceedings of the National Academy of Sciences 1996; 93: 1591-5.
[31] van den Doel K, Ascher U, On level set regularization for highly ill-posed distributed parameter estimation problems. Journal of Computational Physics 2006; 216: 707-23.
[32] Li C, Xu C, Gui C, Fox M. Level set evolution without re-initialization: a new variational formulation. Proc. Comp. Vision and Pattern Recognition vol. 1. IEEE: 2005, p. 430-6.
[33] Gonzalez-Vega L, Necula I. Efficient topology determination of implicitly defined algebraic plane curves. Comput. Aided Geom. Design 2002; 19: 719-743.
[34] Lee K. Principles of CAD/CAM/CAE systems. Boston: Addison-Wesley; 1999.
[35] Fletcher R, Practical Methods of Optimization, 2nd edition. London: Wiley; 2000.
[36] Bartoň M, Jüttler B. Computing roots of polynomials by quadratic clipping. Computer Aided Geometric Design 2007; 24: 125-41.
[37] Meek DS, Ong BH, Walton DJ. Constrained interpolation with rational cubics. Computer-Aided Design 2003; 20: 253-75.
[38] Peters J, Wu X, SLEVEs for planar spline curves. Comput. Aided Geom. Design 2004; 21: 615-35.
[39] Flöry S. Fitting curves and surfaces to point clouds in the presence of obstacles. 2006; Technical Report no. 160, Geometric Modeling and Industrial Geometry Group, Vienna University of Technology.
[40] Yang H, Jüttler B. Meshing Non-uniformly Sampled and Incomplete Data Based on Displaced T-spline Level Sets. In Proc. Shape Modeling and Applications 2007, IEEE Computer Society; 2007, p. 251-60. Available as FSP report no. 39 at www.ig.jku.at.