FSP Report No. 24

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

R. Feichtinger, M. Fuchs, B. Jüttler,
O. Scherzer and H. Yang

October 2006

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 [7, 26], for the reconstruction of geometric objects from unorganized points, see [6, 15, 19, 35, 36] 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 [12]. 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 [3, 12, 23, 24, 25, 27, 28, 31, 32].

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 [24, 32]. This is similar the notion of ‘active (parametric) curves’ which are used for image segmentation in Computer Vision and image processing [4, 16]. In order to do segmentation, [16] introduced the idea ‘active curves’ which minimize an energy functional in a space of admissible curves. As shown in [8], his 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 evolution.

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 a particular type of 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.

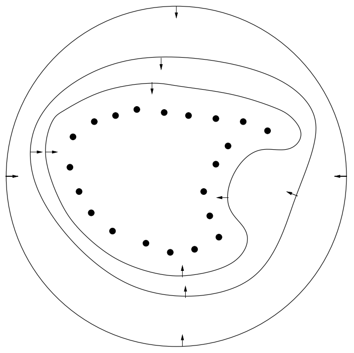


Figure 1: An "active" curve moving toward some data points (3 time steps).

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).

We will describe a technique for extracting the information 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.

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 $C = C_t$, 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. 2.1. The data is used to derive some information about the expected normal speed of the curve. This will be described in the next section.

Remark 1

We recommend to 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 of the initial curve lies within the target or if they intersect each other. This difference is caused by the different evolution speed functions which are described below.

We assume that the data does not specify nested loops. Techniques for handling this situation are described in [34].

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.

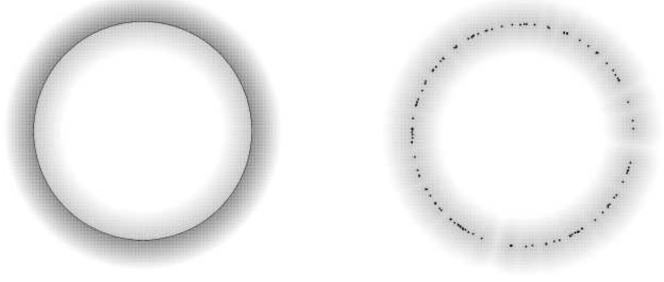


Figure 2: Precomputed distance fields.

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

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

which was proposed in [8], where e is the so–called edge detector function

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

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.

In the case of data points, we use

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

with the edge detector

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

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 [13]. 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.

2.3 Evolution of parametric curves

We consider a closed parametric spline curve

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

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 represents differentiation with respect to the time parameter τ,

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

If the control points vary in time, the points of the curve travel with the normal velocity

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

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

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

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

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

with

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

Finally, by using the B-spline representation for $\mathbf{f}$, this can be rewritten as

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

The solution $\dot{\mathbf{c}}(\tau)$ of this problem is obtained by solving the sparse linear system with a symmetric positive definite 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 [5].

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 of the geometry features in the target shape.

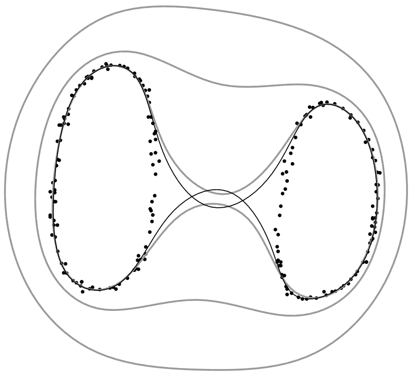


Figure 3: Evolution of a parametric curve towards a point cloud.

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

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 [10] for more information on this type of regularization.

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

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

e.g., by using a Gauss–Newton method, such as the method of normal distance minimization described in [4], see also [32]. 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 [1, 2], this can also be seen as an evolution of a curve, where the normal velocities of the closest points $\mathbf{Q}_j$ are equal to the oriented distances to the data.

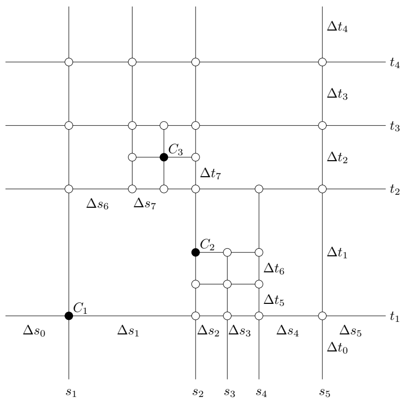


Figure 4: A T-spline grid. We use 4–fold knots at boundaries.

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 5: The knot vectors for some selected control points.

2.4 Evolution of implicitly defined curves

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

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

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

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

are defined with the help of (in our case) cubic B-splines over 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 and Fig. 2.4. See [29] for more information.

Since the T–spline grid allows T-junctions, T-splines can be refined locally. Clearly, this is not the case for tensor product B-splines. If the T–spline grid does not contain 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 $\mathbf{c} = (c_0,\dots,c_n)$. 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,

$$E(\dot{\mathbf{c}}) = \sum_{j=1}^{N_0} (\dot g(\mathbf{x}_j,\tau)+v(\mathbf{x}_j,\tau)\ |\nabla g(\mathbf{x}_j,\tau)|)^2 \to \min \tag{20}$$

with uniformly distributed sample points $\mathbf{x}_j$, $j=1,\dots,N$, on 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 similar to [9, 17, 33], we avoid the re-initialization by introducing a *distance field constraint*.

More precisely, we add the constraint term

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

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.

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

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

with a certain positive weight w. 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 w, the discretization of the signed distance constraint term, and the selection of the T–spline grid, have been presented in [33]. An example for the adaptive choice of the T-spline grid will be given later (Example 7).

2.5 Dual evolution

On the hand, a parametric spline representation of the curve is needed, e.g., in Computer Aided Design. On the other hand, the evolving parametric curves have some difficulties to deal with

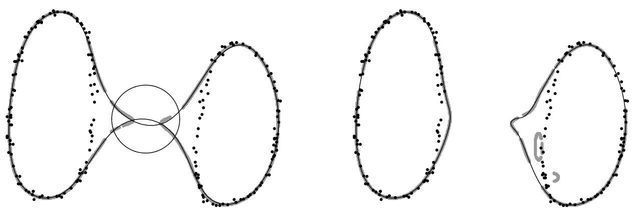


Figure 6: Adapting the topology by dual evolution of an implicitly defined (grey) and a parametric curve (black).

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 increased popularity of the level–set method.

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.
2. *Evolution:* Apply the evolution of the implicit and parametric curves for one time step.
3. *Synchronization:* Detect and deal with occurring problems, such as additional branches and topology 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. 6, 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.

3 The synchronization step

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

$$\vec{\mathbf{n}}_{\mathbf{f}}(u_j)^T \, \vec{\mathbf{n}}_g \, (\mathbf{f}_j)) \leq 0. \tag{23}$$

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. 7 for an illustration).

Lemma 1 *Consider a subdomain $S \subseteq \Omega$ with boundary ∂S, such that the function g has no extrema*[1] *in this domain. 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 the value of the inner product*

$$\nabla g^T \, \vec{\mathbf{n}}_{\mathbf{f}} \tag{24}$$

is not equal to zero everywhere on ∂S. Then the sign of this inner product (24) *changes on ∂S.*

[1]Here, a point $\mathbf{p}$ is said lo be a maximum (and similar for a minimum) of g on S if $g(\mathbf{p}) \geq g(\mathbf{x})$ and $g(\mathbf{p}) > g(\mathbf{x}')$ holds for all $\mathbf{x} \in S$ and $\mathbf{x}' \in \partial S$.

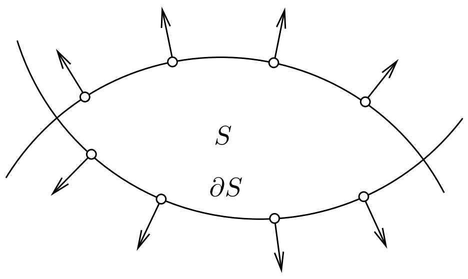


Figure 7: The assumptions of Lemma 1.

Proof: If there were no such sign changes, then value of g would always increase (or decrease) when one crosses the boundary ∂S. This is in contradiction to the assumption that g has no local extrema in S. $\square$

Remark 5 The assumption concerning the non–vanishing inner product is needed, e.g., in order to exclude the case of a constant function g.

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 6 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,

$$g(\mathbf{f}(u) + \lambda(u)\,\vec{\mathbf{n}}_f(u)) = 0. \tag{25}$$

By differentiation we obtain a differential equation for λ,

$$\lambda' = -\frac{\nabla g^T\,(\mathbf{f}' + \lambda\,\vec{\mathbf{n}}')}{\nabla g^T\,\vec{\mathbf{n}}} \tag{26}$$

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 λ 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.

Remark 7 In order to speed up the computation one may instead simply check whether the the sign of g changes in a tubular ϵ–neighborhood around the parametric curve. If the zero level is close to the parametric curve, then the sign changes in this neighborhood.

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 structure. 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 very well.

Note that it is not a very serious problem if the methods are too sensitive, i.e., if they report topological changes if no such events have actually taken place. As we will see later, the method for adapting 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 duly reported by the first method (bottom right).

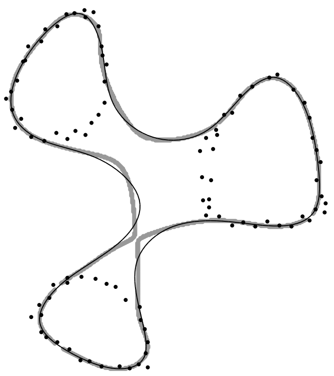
3.2 Synchronization without topological changes

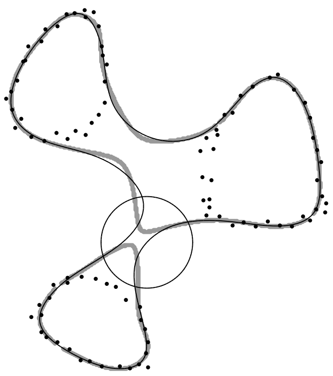
If no change of the topology has been reported, then we try to make sure that the two representations of the curve stay close to each other. Two possibilities exist:

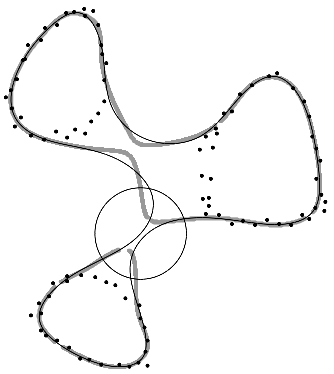
Fitting the implicitly defined curve to the parametric one

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

$$\sum_{j=1}^{N} e^{-\lambda\phi(\mathbf{y}_j)}\,(g(\mathbf{y}_j) - \phi(\mathbf{y}_j))^2 \to \min\,. \tag{27}$$


Initial position


Step 1: Method 3 detects the change


Step 4: Method 2 detects the change.

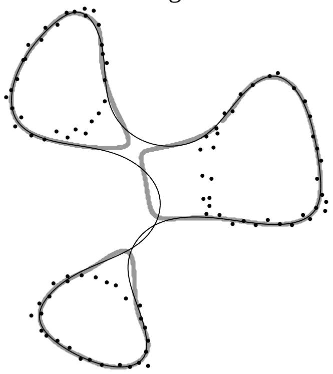

Step 7; Method 1 detects the change.

Figure 8: Detection of topological changes.

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 λ is a user–defined parameter.

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.

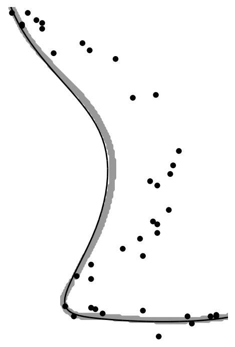
Example 5 Fig. 9 shows an example.

Fitting the parametric curve to the implicitly defined one

In some cases, e.g., if additional constrains 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.

3.3 Synchronization with topological changes

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.


before synchronization

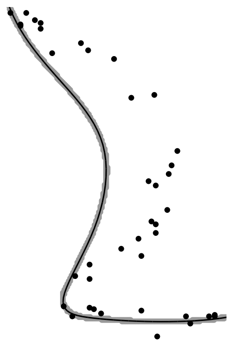

after synchronization v

Figure 9: Synchronization without topological changes.

Algorithm 2

1. Create a circle C with a predefined radius (which should be chosen as the expected feature size) around $\mathbf{p}_i$.

2. Compute the set P of intersections between C and the parametric curve $\mathbf{f}$. Compute the set I set 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.

Remark 8 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 loose 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 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).

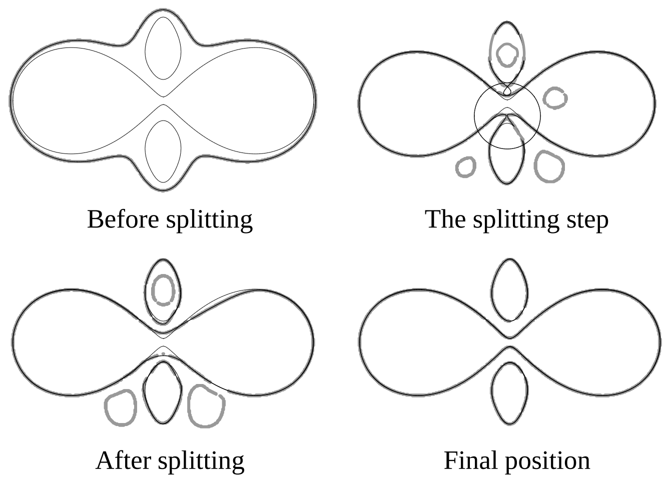


Figure 10: Synchronization with topological changes. The target is defined by a smooth curve.

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.

Remark 9 With this method we can theoretically deal with an 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 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. 11 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.

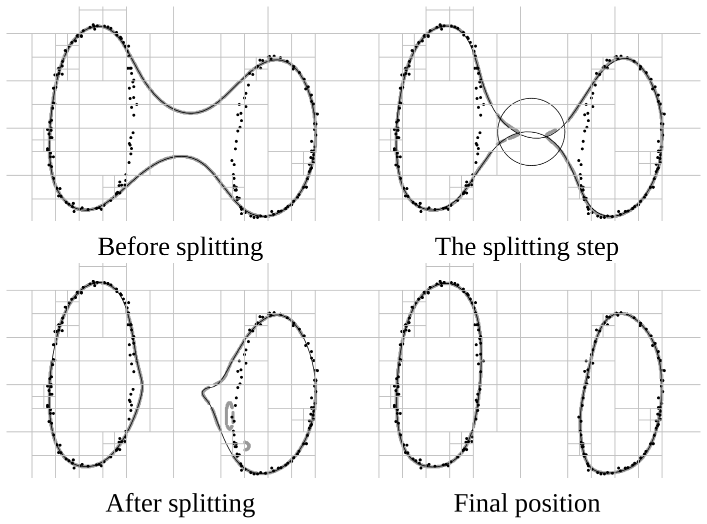


Figure 11: Synchronization with topological changes. The target is defined by a point cloud.

4 Range constraints

The implicitly defined curve decomposes the domain into an *inner* region, where $g(\mathbf{x}) \leq 0$, and an *exterior* region, where $g(\mathbf{x}) \geq 0$. 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

$$g(\mathbf{x}_i) \leq 0, \quad \text{and} \quad g(\mathbf{y}_j) \geq 0, \tag{28}$$

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

$$C(\dot{\mathbf{c}}) = \sum_{j=1}^{N_0} \left(\dot{g}(\mathbf{x}_j, \tau) + g(\mathbf{x}_j, \tau) + \delta\right)^2 \alpha_\varepsilon(g(\mathbf{x}_j, \tau)) \tag{29}$$

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

$$\alpha_\varepsilon(g) = \begin{cases} 1 & g > -\varepsilon \\ 0 & g < -2\varepsilon \\ C^2 - \text{blend} & \text{in between} \end{cases} \tag{30}$$

where ε is a used–defined positive constant. Again, the optimization problem

$$\widehat{F}(\dot{\mathbf{c}}) = E(\dot{\mathbf{c}}) + w_1\, S(\dot{\mathbf{c}}) + w_2\, C(\dot{\mathbf{c}}) \rightarrow \min. \tag{31}$$

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

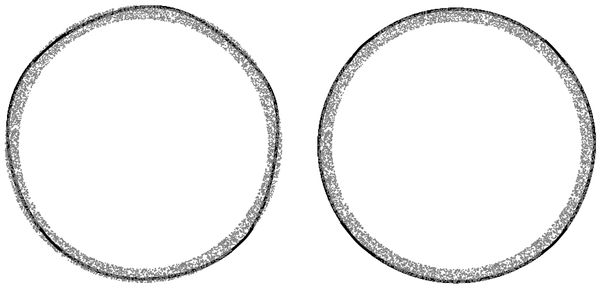


Figure 12: Approximation with (right) and without (left) range constraints.

‘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. 12, 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 (29) with $\delta = -0.5$.

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

$$((x - \frac{3}{4})^2 + y^2)\,((x + \frac{3}{4})^2 + y^2) = 0.316. \tag{32}$$

Fig. 13 shows the dual evolution for a target defined by this curve. Depending on the choice of δ, we obtain offsets of the algebraic curve.

Example 10 In this final example (see Fig. 14) we consider a target which consists of three parts. Evolution without constraits 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 [18], the use of tight piecewise linear enclosures have been proposed in [22], and certain optimization techniques are explored in [11].

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. 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.

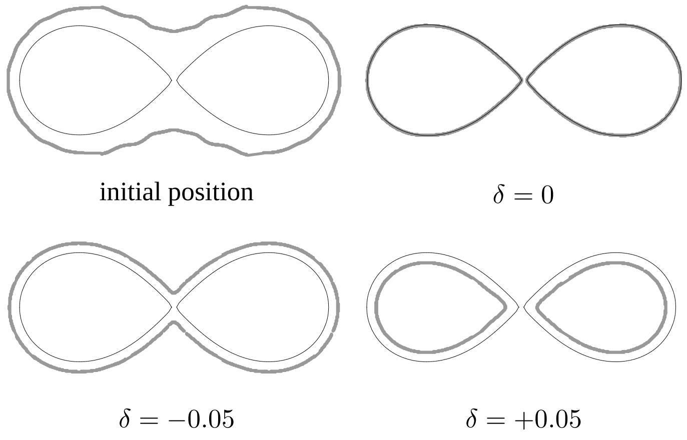


Figure 13: Dual evolution with range constraints for a target defined by an algebraic curve.

As a matter of future research, we plan to investigate other constraints, such as convexity. The convexity of implicitly defined curves can be guaranteed via convexity of the underlying scalar field, and computationally efficient criteria for the convexity of spline functions exist [14]. Their application within the framework of dual evolution seems to be promising.

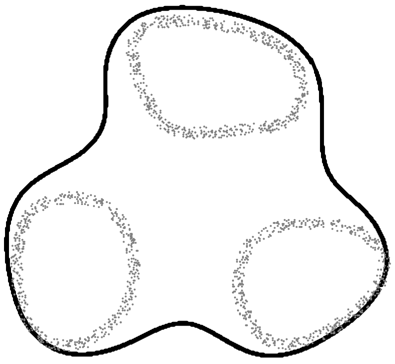
Currently we study the extension of the results to surfaces, where we need to define evolution processes and to discuss the interaction of the two representations. The evolution of T–spline Level sets in 3D has already been discussed in [33]. However, the choice of a suitable surface representation is still open, and we are considering triangular meshes, manifold splines, or geometrically continuous spline surfaces.

Acknowledgments. The financial support of the Austrian Science fund through the Joint Research Programme S92 ‘Industrial Geoemtry’, subproject 2, and by the European Union through the Marie Curie IIF Fellowship for Huaiping Yang (project 22073 ISIS) is gratefully acknowledged.

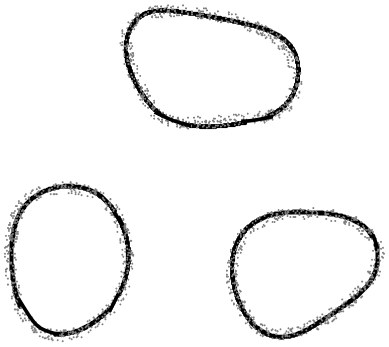
References

[1] Aigner, M., Šír, Z., and Jüttler, B., Least–Squares Approximation by Pythagorean Hodograph Spline curves via an Evolution Process, Geometric Modelling and Processing, M.-S. Kim and K. Shimada (eds.), Springer LNCS 4077, 2006, 45–58.

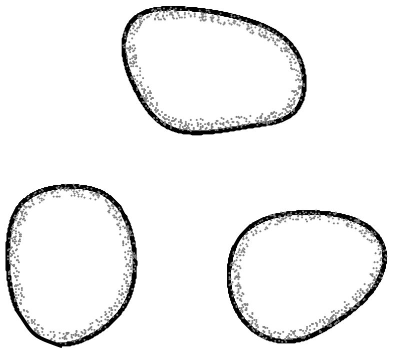
[2] Aigner, M., and Jüttler, B., Hybrid curve fi tting, Computing, to appear, available as an FSP report at `http://www.ig.jku.at`.



initial position



result without constraints



result with constraint, $\delta = -0.5$

Figure 14: Dual evolution with range constraints for a target defined by three point clouds.

[3] Alhanaty, M., and Bercovier, M., Curve and surface fitting and design by optimal control methods, Computer–Aided Design 33 (2001), 167–182

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

[5] Botsch, M., Bommes, D. and Kobbelt, L., Efficient linear system solvers for mesh processing. In Martin, R., et al., eds., *The Mathematics of Surfaces XI*, volume 3604 of *LNCS*, Springer, 2005, 62–83

[6] Carr, J.C., et al., Reconstruction and representation of 3D objects with radial basis functions. In *Proc. SIGGRAPH'01,* pages 67–76, 2001.

[7] Cartwright, R., et al. Web-based shape modeling with HyperFun. *IEEE Computer Graphics and Applications*, 25:60–69, 2005.

[8] Caselles, V., Kimmel, R., and Sapiro, G., Geodesic active contours, *Int. J. of computer vision*, **22(1)** 1997, 61–79

[9] van den Doel, K., and Ascher, U., On level set regularization for highly ill-posed distributed parameter estimation problems. manuscript available at `www.cs.ubc.ca/~kvdoel/pubs.html`.

[10] Engl, H., Hanke, M., and Neubauer, A., Regularization of inverse problems, Kluwer, Dodrecht 1996.

[11] Flöry, S., Fitting curves and surfaces to point clouds in the presence of obstacles, Technical Report no. 160, Geometric Modeling and Industrial Geometry Group, Vienna University of Technology, 2006.

[12] Hoschek, J., and Lasser, D., Fundamentals of Computer Aided Geometric Design, AK Peters, Wellesley, 1996.

[13] Hoff, K.E., Culver, T., Keyser, J., Lin, Ming and Manocha, D., Fast computation of generalized Voronoi diagrams using graphics hardware. *SIGGRAPH'99*, 1999, 277–286

[14] Jüttler, B., Surface fitting using convex tensor–product splines, J. Comp. Appl. Math. 84 (1997), 23–44.

[15] Jüttler, B., and Felis, A., Least–squares fitting of algebraic spline surfaces. *Adv. Comput. Math.*, 17:135–152, 2002.

[16] Kass, M., Witkin, A. and Terzopoulus, D., Snakes: active contour models, *Int. J. of computer vision*, **1(4)** 1988, 231–233

[17] Li, C., Xu, C., Gui, C., and Fox, M.. Level set evolution without re-initialization: a new variational formulation, *Proc. Comp. Vision and Pattern Recognition*, vol. 1, 430–436. IEEE, 2005.

[18] Meek, D. S., Ong, B. H., Walton, D. J., Constrained interpolation with rational cubics. Computer-Aided Design 20 (2003), 253–275.

[19] Ohtake, Y., et al., Multi-level partition of unity implicits. ACM Transactions on Graphics 22(3), (2003) (Proc. SIGGRAPH)

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

[21] Osher, S., and Fedkiw, R.P., Level set methods and dynamic implicit surfaces, Springer, 2003

[22] Peters, J., Wu, X., SLEVEs for planar spline curves, Comput. Aided Geom. Design 21 (2004), 615–635.

[23] Pottmann, H., and Leopoldseder, S., A concept for parametric surface fitting which avoids the parametrization problem. Comp. Aided Geom. Design 20 (2003), 343-362.

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

[25] Pottmann, H., et al., Industrial geometry: recent advances and applications in CAD, Computer-Aided Design **37** (2005), 751–766.

[26] Raviv, A, and Elber, G., Three dimensional freeform sculpting via zero sets of scalar trivariate functions. In *Proc. 5th ACM Symposium on Solid Modeling and Applications*, pages 246–257, 1999.

[27] Rogers, D.F., and Fog, N.G., Constrained B-spline curve and surface fitting, Computer Aided Design 21 (1989), 641–648.

[28] Sarkar, B., and Menq, C.H., Parameter optimization in approximating curves and surfaces to measurement data, Comp. Aided Geom. Design 8 (1991), 267–280

[29] Sederberg, T.W., Zheng, J., Bakenov, A. and Nasri, A., T-splines and T-NURCCs, *ACM Transactions on Graphics*, **22(3)** 2003, 477–484

[30] Sethian, J., A fast marching level set method for monotonically advancing fronts, *Proceedings of the National Academy of Sciences*, volume 93, 1996, 1591–1595

[31] Speer, T., Kuppe, M., and Hoschek, J., Global reparametrization for curve approximation, Comput. Aided Geom. Design **15** (1998), 869–877.

[32] Wang, W., Pottmann, H., and Liu, Y., Fitting B-spline curves to point clouds by squared distance minimization. ACM Transactions on Graphics **25.2** (2006).

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

[34] Yang, H., Jüttler, B., and Gonzalez-Vega, L., Approximate parameterization of algebraic curves by an evolution process, talk at AGGM 2006, extended abstract available at `www.imub.ub.es/aggm06`, manuscript in preparation.

[35] Yang, Z.W., Deng, J.S., and Chen, F.L., Fitting unorganized point clouds with active implicit B-spline curves. *The Visual Computer*, 21(8-10):831–839, 2005.

[36] Zhao, H.K., Osher, S., and Fedkiw, R. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 194–201, Vancouver, 2001.