The Visual Computer manuscript No.
(will be inserted by the editor)

Snakes with topology control

Stephan Bischoff, Leif P. Kobbelt

RWTH Aachen, Lehrstuhl für Informatik VIII, 52056 Aachen, Germany
e-mail: {bischoff|kobbelt}@informatik.rwth-aachen.de

Received: date / Revised version: date

Abstract We present a novel approach for representing and evolving deformable active contours by restricting the movement of the contour vertices to the grid-lines of a uniform lattice. This restriction implicitly controls the (re-) parameterization of the contour and hence makes it possible to employ parameterization independent evolution rules. Moreover, the underlying uniform grid makes self-collision detection very efficient. Our contour model is also able to perform topology changes but – more importantly – it can detect and handle self-collisions at sub-pixel precision. In applications where topology changes are not appropriate we generate contours that touch themselves without any gaps or self-intersections.

Key words active contour model – topology control – implicit parameterization

1 Introduction

For the segmentation and shape reconstruction from noisy image data, contour extraction schemes based on deformable models have become a standard technique. The major reason for their successful use in many applications is the possibility to integrate physical and topological knowledge into the segmentation process and thus "interpolate" the image information where it is destroyed by noise.

Various representations have been proposed which adapt to the extreme requirements in an active contour model. *Explicit* contour representations can be processed very efficiently and their physical properties can be controlled in a very intuitive manner. *Implicit* contour representations require more sophisticated implementations but they are free of parameterization artifacts and they allow the contour to change its topology in a natural manner (see the next section for a more detailed description).

Our new approach inherits from both the explicit and the implicit framework: The representation of the contour is basically explicit, its evolution however is governed by parameterization independent rules similar to those of the fast marching methods in the level set framework. Self-collisions of the contour can be detected easily and the algorithms can flexibly decide if the contour topology should change or should be preserved. The main contributions of our proposed scheme are:

- *Flexible topology control.* In contrast to previous work we are able to efficiently detect and resolve self-collisions without globally reparameterizing the contour. Depending on the user's preferences, our algorithm can be tuned to preserve the contour's topology as well as to merge the colliding contours.
- *Automatic resampling.* The resolution and parameterization of our contour is automatically determined by an underlying uniform lattice. As a consequence, there is no need for a complicated global resampling procedure when the contour is deformed.
- *Simplicity.* The basic operations used in our scheme are conceptually straightforward and can be implemented easily. All computations during the evolution are local and no handling of special cases is necessary. In particular, there is no need to maintain and update elaborate data structures, like narrow bands of voxels, or to approximate and discretize differential equations.
- *Speed.* Due to the robustness of the evolution procedure and the flexible control of the time steps, we can use an explicit Euler integration scheme to trace the contour through the embedding force field.

Overview In Section 2 we give a short overview of previous and related work. Section 3 briefly introduces active contour models. In Section 4 we describe our new scheme and give implementation hints. Results for synthetic as well as for real

Correspondence to:
Stephan Bischoff
Lehrstuhl für Informatik VIII
52056 Aachen
Germany
e-mail: bischoff@informatik.rwth-aachen.de
Phone: +49 241 8021817
Fax: +49 241 8022899

data are presented in Section 5. In Section 6 we give conclusions and propose future work.

2 Previous and related work

Previous work In recent years, image segmentation based on active contour models has become a powerful tool. Especially in medical imaging applications, like the segmentation of organic structures or the discrimination of brain tissues, these models are ubiquitous [1,6,10,16,23]. Depending on the representation of the contour shape as the range or the kernel of a function, active contour models can be classified as either *explicit* or *implicit.*

Image segmentation based on explicit active contour models has first been introduced in 1987 by Kass et. al [13]. In their work, a contour is represented by a parametric model (a so-called snake) and its evolution is governed by minimizing an energy-functional and applying a semi-implicit integration scheme. Since then numerous refinements and extensions to the original scheme have been proposed [3–5,8,12, 19]. Several authors have introduced different explicit representations e.g. finite element models [3] and subdivision curves [12]. The explicit active contour model has also been generalized to higher dimensions, such as to segment volume data like MRI scans of human organs [3,5,22]. Early explicit active contour models could not handle topology changes, like merging or splitting of contours. In order to overcome this limitation several authors have proposed topologically adaptive contour models which are e.g. based on repeated resampling of the contour on an affine grid [7,8,15,18,20].

Implicit models, on the other hand, represent the contour as the (zero-) level set of a scalar field and were first introduced by Sethian and Osher in 1988 [24]. Since then, level set methods have been applied in numerous applications, among them image segmentation, fluid dynamics and computer vision. We refer to the books [25,27] for a thorough overview. In order to overcome the computational complexity of level set methods, fast marching and narrow band methods have been introduced [2,26]. Implicit representations can easily handle topological changes of the contour, so in contrast to explicit representations special care has to be taken, if topological changes of the level set have to be *avoided* [11].

Contribution In this paper, we introduce an active contour model that combines properties of the implicit as well as the explicit frameworks. The evolution of the contour is driven by Huygen's principle and hence resembles that of the level set methods. The topology of the contour is represented explicitly by a control polygon and can be compared to the traditional snakes approach. Note, however, that we only address the evolution process and the topology control of the contour — we do *not* propose any new ways of defining image gradient forces or otherwise improve the quality of the segmentation. This is reflected by the fact that we will consider the forces that drive the contour as a "black box" which is provided by the user and which incorporates all the external and internal forces that account for the quality of the final segmentation.

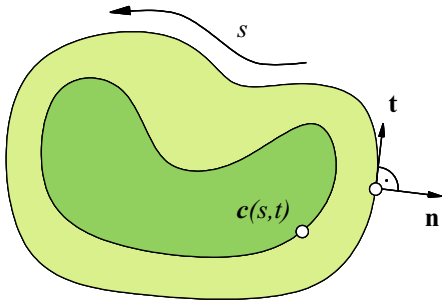

Fig. 1 Evolving contours: A contour $\mathbf{c}(s,t)$ is parameterized by arc length s and time t. The movement of each contour point $\mathbf{c}(s,t)$ can be decomposed into tangential and normal components. Whilst the tangential component affects the parameterization of the contour, only the normal component modifies the contour's shape.

The combination of implicit and explicit techniques provides us with a greatly improved control over the topology of the contour. Collisions can accurately and robustly be detected and resolved without incurring a run-time overhead. Purely implicit models, in contrast, provide no collision detection at all. For purely explicit models collision detection is possible, but expensive and often inaccurate. In particular, in our model the user can choose whether two colliding contours should "clash", an operation that is not possible in implicit frameworks or whether they should "merge", an operation that is inefficient in explicit frameworks.

3 Active contour models

In this section we give a short introduction to active contour models. For simplicity we will restrict ourselves to the two-dimensional case although most of the following can readily be generalized to higher dimensions.

The idea of active contour models is to track the evolution of a simple, closed curve, the so-called *contour*. The contour can be represented either implicitly as the level set of a function [25,27] or explicitly by a parametric representation [3, 12,13]. Here we will focus on the latter case.

Consider a contour $\mathbf{c}(s,t) \in \mathbb{R}^2$ where $s \in [0,1]$ parameterizes the contour arc and $t \in \mathbb{R}_{\geq 0}$ designates time, see Figure 1.

The evolution of $\mathbf{c}$ can then be described by the following equation

$$\frac{\partial \mathbf{c}}{\partial t} = \alpha\,\mathbf{t} + \beta\,\mathbf{n} \tag{1}$$

where $\mathbf{t}$ is the tangent, $\mathbf{n}$ is the outward normal and α and β are arbitrary functions describing the tangential and the normal speeds of the contour respectively. Here and in the following we will assume that the contour is closed, i.e. that

$$\mathbf{c}(0,t) = \mathbf{c}(1,t)$$

and that it consists of only one component.

There are numerous ways to define the functions α and β. In the classical setup they are chosen such that the contour minimizes an energy functional

$$\mathcal{E}(\mathbf{c}) = \mathcal{E}_{\text{internal}}(\mathbf{c}) + \mathcal{E}_{\text{external}}(\mathbf{c}). \quad (2)$$

$\mathcal{E}_{\text{internal}}$ represents the internal energy of the contour, and is in general a weighted combination of membrane and thin-plate energy which penalize stretching and bending, resp. It is used to regularize, i.e. to smooth the contour, and hence to avoid artifacts like overshooting or ripples. $\mathcal{E}_{\text{external}}$ represents the external energy which is in general a potential field derived from the underlying segmentation problem, e.g. attraction to image features.

It can be shown, that for each choice of speed functions (α, β) there exist other speed functions $(0, \bar{\beta})$ such that the resulting contour shapes are equivalent [9, 14]. Hence, the tangential component α in general only affects the parameterization of the contour while β determines the contour's shape. For parameterization-less formulations, like the implicit level set formulation or our r-snake formulation (see Section 4), Equation 1 can be simplified to

$$\frac{\partial \mathbf{c}}{\partial t} = \bar{\beta}\mathbf{n}$$

i.e. the contour only evolves in normal direction.

In practice, the contour $\mathbf{c}$ is represented either *discretely* by the vertices of a polygon or *continuously* by e.g. B-splines, subdivision curves or other basis functions. For our purposes, we discretize the contour $\mathbf{c}$ as a polygon in space and in time by a sequence of vertices, so-called *snaxels*,

$$\mathcal{C}(t_i) = \mathbf{c}_1^i, \ldots, \mathbf{c}_{n_i}^i, \quad i \in \mathbb{N}$$

where n_i designates the number of vertices of the snake at timestep $t_i \in \mathbb{R}_{\geq 0}$. By approximating derivatives through finite differences, the continuous Equation 2 can then be transformed into a discrete update rule for the snaxel positions.

In this discrete setup, the tangential speed α can be thought of as regularizing the vertex distribution, e.g. towards uniform or curvature dependent vertex spacing. However, in general some local or global vertex insertion/deletion strategies have to be implemented in order to adapt the number of vertices to the contour's length.

4 Parameterization free active contour models

In this section we present a simplified type of snakes that we call *restricted snakes (r-snakes)*. Although r-snakes lack some of the original snakes flexibility, they can be used in a wide range of settings and allow for topology preserving, intersection-free evolution of a contour.

An r-snake is a special type of snake. Instead of letting the snaxels move freely, we impose certain restrictions on their movements. Most importantly the snaxels may only move along the lines of a given, fixed grid. Whenever a snaxel runs into a gridpoint, it is automatically split. Finally we assume that the snake moves only normal to itself and that it may not self-intersect.

The above restrictions allow us on the one hand to efficiently detect and avoid collisions. On the other hand, they automatically resample the snake according to the resolution of the underlying grid.

4.1 Definition

In the following we assume that the Euclidean plane is subdivided by a $\mathbb{Z} \times \mathbb{Z}$ integer grid into unit squares that we call *pixels*. The sides of the pixels are called *grid segments* in contrast to *snake segments* that join two consecutive snaxels.

Consider an intersection-free, closed snake

$$\mathcal{S} = s_1, \ldots, s_n,$$

such that $\mathcal{S}$ divides the Euclidean plane into an interior and an exterior part. We call $\mathcal{S}$ a *restricted snake*, *r-snake* for short, if the following three properties hold:

1. (Supporting segments) Each snaxel $s \in \mathcal{S}$ lies on a grid segment which we call the *supporting segment* of s. To be more precise, for each snaxel s there are two gridpoints $\mathbf{f}_s \in \mathbb{Z} \times \mathbb{Z}$ ("from") and $\mathbf{t}_s \in \mathbb{Z} \times \mathbb{Z}$ ("to") such that

   $$||\mathbf{f}_s - \mathbf{t}_s|| = 1$$

   and an affine parameter $0 \leq d_s < 1$ ("distance") such that s's position $\mathbf{p}_s$ on the Euclidean plane is given as

   $$\mathbf{p}_s = (1 - d_s)\,\mathbf{f}_s + d_s\,\mathbf{t}_s$$

   (see Figure 2).

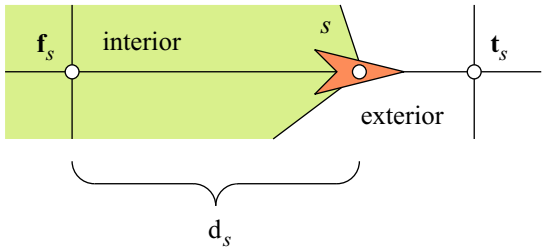

Fig. 2 Snaxel: In this and the other figures snaxels are represented as arrowheads such as to indicate their "from" and "to" vertices.

2. (Orientation) All snaxels are consistently oriented. By convention, each snaxel s points from the interior of $\mathcal{S}$ to the exterior of $\mathcal{S}$ (see Figure 2).
3. (Uniqueness) No two consecutive snake segments of an r-snake $\mathcal{S}$ may lie in the same pixel. Note that this condition follows readily from condition 2, and is merely stated for convenience. Hence snaxel configurations as shown in Figure 3 are forbidden and notches that are thinner than one pixel cannot be represented.

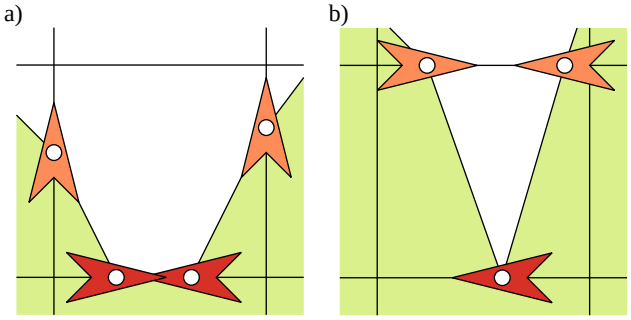

Fig. 3 Forbidden snaxel configurations: The configurations shown above are forbidden, as the darkened snaxels point from the interior to the interior of the contours.

4.2 Implementation

For representing an r-snake, we use a simple data structure. Each snaxel object has 8 members, namely

```
struct Snaxel {
  float d;      /* affine parameter */
  float v;      /* speed            */
  int   fx, fy; /* "from" vertex    */
  int   tx, ty; /* "to" vertex      */
  Snaxel *next, *prev;
                /* connectivity     */
}
```

The `next` and `prev` pointers are used to arrange all snaxels of an r-snake in counter-clockwise direction in a doubly-linked list.

An r-snake can be initialized by resampling an arbitrary closed, self-intersection free curve on the $\mathbb{Z} \times \mathbb{Z}$ grid. If the curve is e.g. given by a signed distance function, the resampling can easily be performed by a Marching Cubes like algorithm [17]. A typical r-snake is shown in Figure 4.

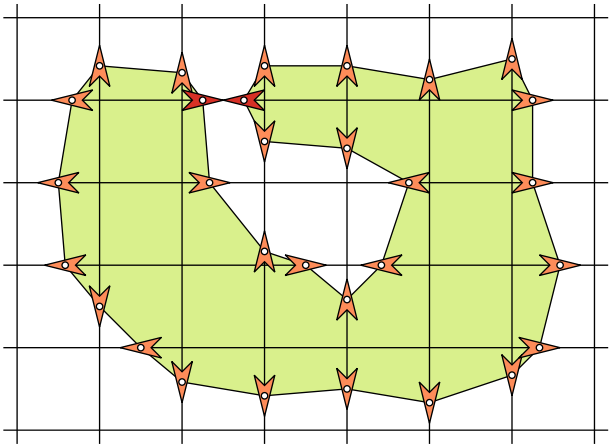

Fig. 4 A typical r-snake: Note that the two darkened snaxels share the same supporting segment. Such a sub-pixel configuration could not be modeled with snakes in level set formulations [11].

4.3 Evolving an r-snake

In this section we describe how an r-snake evolves according to the impact of external and internal forces. In the following we will always assume

1. that the r-snake moves in normal direction and
2. that the r-snake moves outward only.

In general the tangential component of a force affects only the contours' parameterization but not its geometry [14]. Restriction 1 is hence very natural in the parameterization-less level set framework. As the parameterization of an r-snake is automatically adapted according to the underlying grid and hence does not need to be adjusted by tangential forces, we also apply restriction 1 for our setup.

Restriction 2 is basically for convenience only. The following exposition could also be formulated without this restriction but then it would be more elaborate. Note that restriction 2 can also be circumvented by alternately reversing the orientation of the r-snake after each update step, hence exchanging the "inside" and the "outside", see also [21].

In general, the evolution of a snake is determined by various factors and parameters, like external and internal forces. For the sake of generality and simplicity, however, we assume the existence of a "black box" v, which, given an arbitrary snaxel s computes the (scalar) speed v_s of s in direction normal to the r-snake. This black box speed function is assumed to take the application dependent internal and external energies into account.

Suppose that for each snaxel s a normal $\mathbf{n}_s$ is given (this will be explained in more detail below). In general the normal $\mathbf{n}_s$ of the snaxel s will not coincide with the direction of the supporting segment of s. As the snaxel can only move along its supporting segment, we have to project the normal onto the segment and compute the "projected" speed $\tilde{v}_s$ of the snaxel. As one can see from Figure 5 the projected snaxel speed can be easily computed as

$$\tilde{v}_s = \frac{v_s}{\mathbf{n}_s \cdot \mathbf{d}_s}$$

where $\mathbf{d}_s = \mathbf{t}_s - \mathbf{f}_s$ is the unit vector pointing in direction of the supporting segment of s. This formula results in the following update rule for the snaxel positions

$$d_s \leftarrow d_s + \Delta t\, \tilde{v}_s$$

where Δt is the timestep. (The computation of Δt is described in Section 4.4.) Note that the projected speed $\tilde{v}_s$ is in general larger in magnitude than the original speed v_s, as $\mathbf{d}_s$ is in general not parallel to $\mathbf{n}_s$.

There are numerous ways to approximate a normal $\mathbf{n}_s$ in a snaxel s. Although often sufficient, these schemes tend to exhibit some artifacts as is demonstrated in the following example. Consider an r-snake which evolves with constant (unit) speed $v \equiv 1$ and let $\Delta t = 0.5$. After updating all vertices we expect that the r-snake has moved outwards by 0.5 units. Actually, however, because we use only approximated normals, cusps and creases may appear as is shown in Figure 6.

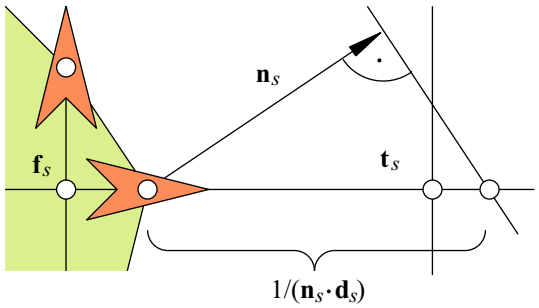

Fig. 5 Projecting the snaxel speed: As snaxels can only move along their supporting segments, the normal speed has to be projected onto the segment.

In this case, the speed $\tilde{v}_s$ resulting from the projection of $\mathbf{n}_s$ is too high and results in an "overshooting" effect. Analogous effects can be observed in case of a concave corner.

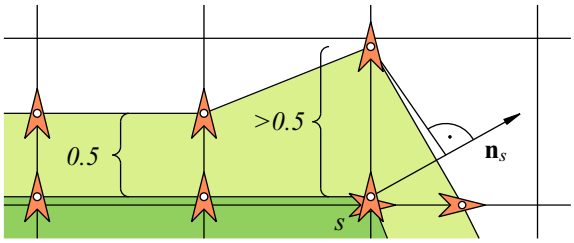

Fig. 6 A contour evolving with unit speed $v \equiv 0.5$ may nevertheless develop cusps because of the arbitrary approximation of the normal $\mathbf{n}_s$ of the snaxel s.

To avoid the "overshooting" and to avoid the necessity to approximate vertex normals altogether, we employ a construction following Huygens' principle [27]. For this we imagine for a moment, that the r-snake is not discrete but continuous and that it locally evolves with constant speed. The intersections of the resulting continuous contour with the grid will then determine the new snaxel positions of the discrete contour.

Consider the case of a snaxel s on a convex corner. We approximate normals both "from the left" and "from the right" by taking the normals of the two snake segments adjacent to s. To be more precise, let s be the snaxel under consideration, a its predecessor and b its successor (see Figure 7). Then we set

$$\mathbf{n}_a = (\mathbf{p}_s - \mathbf{p}_a)^\perp$$
$$\mathbf{n}_b = (\mathbf{p}_b - \mathbf{p}_s)^\perp$$

These two normals give rise to two projected snaxel speeds $\tilde{v}_a$ and $\tilde{v}_b$. Two cases have to be distinguished, see Figure 8.

– Both normals lie on the same side of the supporting segment of s (Figure 8a). The final projected speed should be taken as

$$\tilde{v} = \min\{\tilde{v}_a, \tilde{v}_b\}$$

– The normals lie on different sides of the supporting segment of s (Figure 8b). In this case the final projected speed should be set to

$$\tilde{v}_s = v_s$$

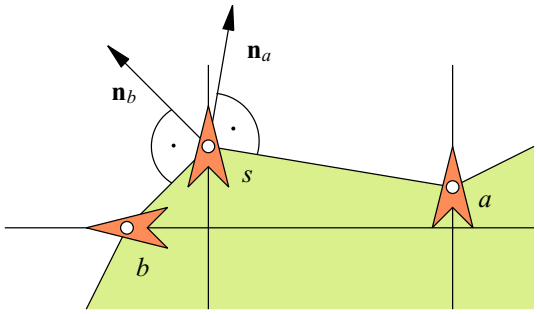

Fig. 7 Normal approximation: In each snaxel s we approximate normals from the "left" and from the "right" by taking the normals of the two snake segments adjacent to s.

Let us now consider the case of a snaxel s on a concave corner. Here again there are two possibilities for the relative orientation of the normals $\mathbf{n}_a$, $\mathbf{n}_b$ and the supporting segment, see Figures 8c and 8d. Both cases lead to the same formula, namely

$$\tilde{v}_s = \max\{\tilde{v}_a, \tilde{v}_b\}$$

4.4 Determining the timestep

To compute the optimal timestep Δt, we proceed as follows. First we note that the number of vertices of an r-snake $\mathcal{C}$ does not change, as long as the r-snake does not cross a grid vertex. Hence a natural upper bound for the timestep Δt can be determined as follows: We first compute for each snaxel s its speed v_s, and then set

$$\Delta t = \min_{s \in \mathcal{C}} \{(1 - d_s)/\tilde{v}_s\}$$

to update all snaxels simultaneously with this timestep. In this way we can be sure that the r-snake does not cross a gridpoint "during" the snaxel update, i.e. it is always guaranteed, that

$$d_s + \Delta t\, \tilde{v}_s \leq 1.$$

Note that as this is a global bound on the timestep, the timestep is expected to decrease when the number of snaxels increases. Hence, for a larger number of snaxels, the algorithm has to perform more update cycles.

4.5 Splitting snaxels

Whenever a snaxel s runs into a gridpoint $\mathbf{x} = \mathbf{t}_s$, i.e. whenever

$$d_s + \Delta t\, \tilde{v}_s = 1$$

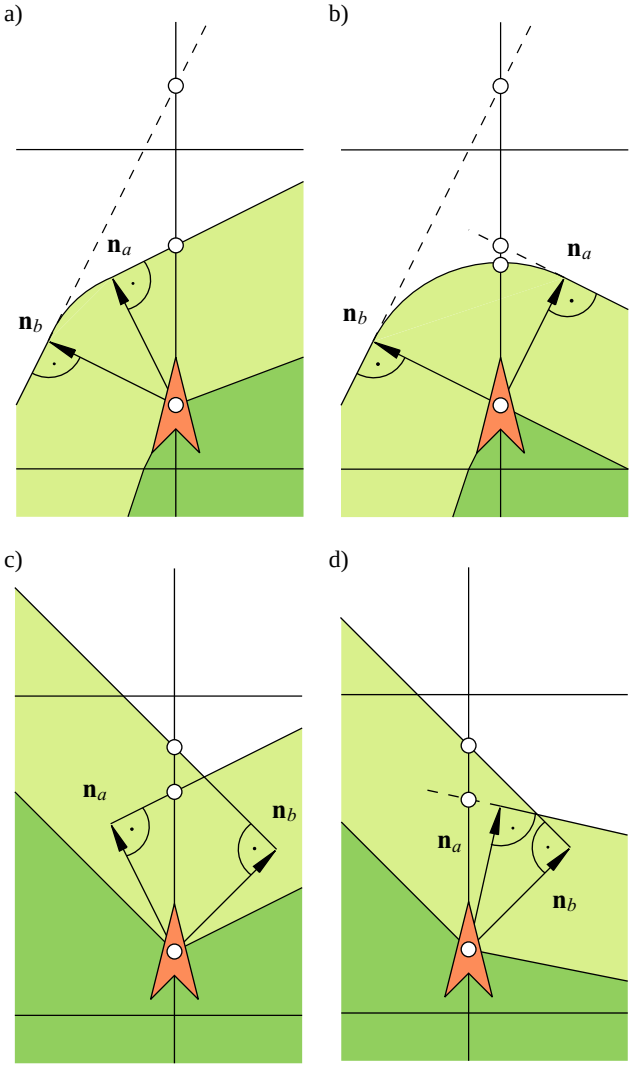

Fig. 8 Snaxel speeds are computed by applying Huygen's principle on continuous contours: The original contour (dark) is locally propagated with constant speed. The intersection points of the resulting continuous offset contour (light) with the grid are then used to determine the new snaxel positions of the original contour.

we split it into three new snaxels a, b, c. The three new snaxels emanate from $\mathbf{x}$ in the other directions but have the same position as s. To be more precise, we set

$$s_a = s_b = s_c = 0$$
$$\mathbf{f}_a = \mathbf{f}_b = \mathbf{f}_c = \mathbf{x}$$

and $\mathbf{t}_a$, $\mathbf{t}_b$, $\mathbf{t}_c$ accordingly. Figure 9 depicts this operation.

After a snaxel split, condition 3 of Section 4.1 might be violated, as is depicted in Figure 10. To reestablish the r-snake property we perform a cleaning conquest: All snaxels that violate condition 3 simply are removed. The cleaning conquest has to be applied recursively to the neighborhood of the split snaxel and to the neighborhood of each removed snaxel.

Performing a snaxel split and the following cleaning conquest reestablishes the r-snake property. Nonetheless there often remain double and triple vertices as depicted in Figure 11. These vertices are conceptually distinct, as they have different supporting segments. However, they share the same spatial position such that the "left" and/or "right" normals are not well-defined. In such a case, we only use the well-defined normals to compute the projected speed. If there is none, as in the case of a center triple vertex s, we set $\tilde{v}_s = v_s$.

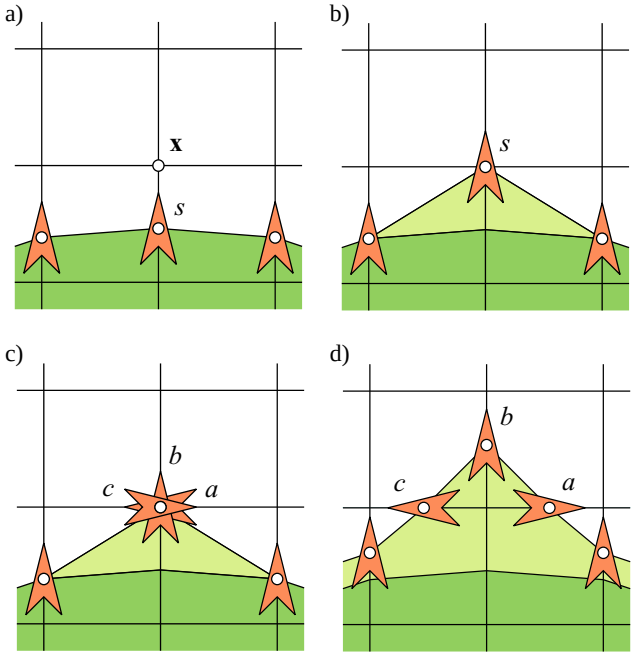

Fig. 9 Splitting snaxels: a) Original r-snake. b) Snaxel s has run into gridpoint $\mathbf{x}$ from the south. c) Snaxel s is split into three new snaxels a, b, c that run to the east, north and west respectively. d) Some timesteps later.

4.6 Collision detection and avoidance

In our setup it is easy to detect and avoid (self-) intersections of r-snakes. For each grid segment, we store the snaxels that are supported by this segment (at most two). If memory requirements are an issue, this can be accomplished by a hashtable, which is indexed by the snaxels "from" and "to" coordinates, see e.g. [8]. Hence it is easy to detect potential collision partners: They are supported by the same grid segment and hence have the same hash-key. Whenever a potential collision is detected, we adapt the timestep such that the two corresponding snaxels will not cross, but just touch each other. Depending on the application, we may choose whether the two colliding snaxels will *clash* and come to a halt (topology preservation) or whether they will *merge* (topology change), see Figure 12.

Clashing Because the contour propagates only outward and may not self-intersect, the two colliding snaxels will stay in their position forever. Hence, we flag them as *frozen* and exclude them from the remaining update steps (Figure 13). We can further decide, whether frozen snaxels are affected by the

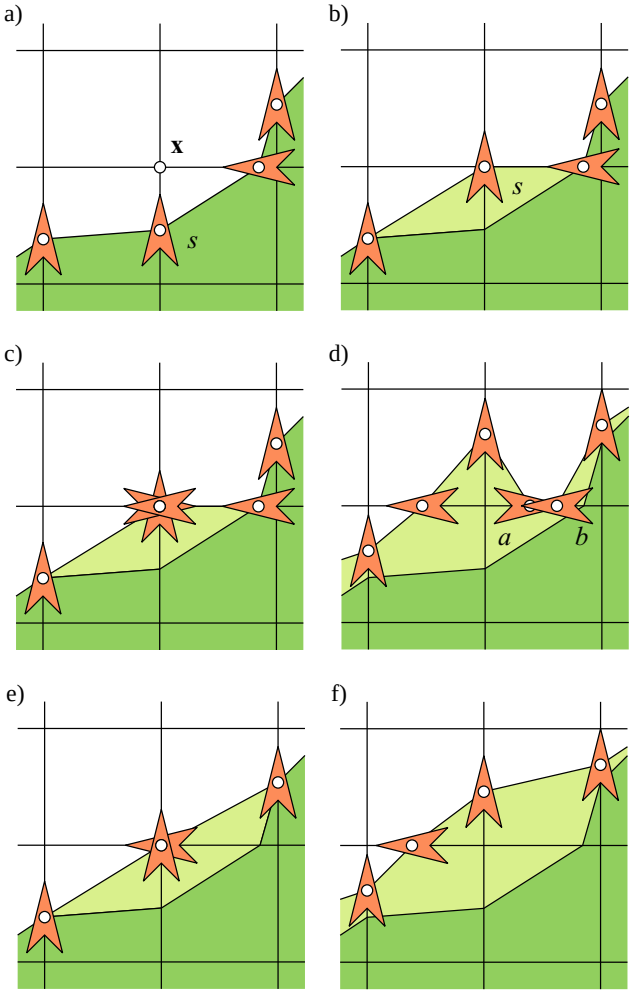

Fig. 10 Cleaning conquest: a) Original r-snake. b) Snaxel s has run into gridpoint $\mathbf{x}$. c) Snaxel s is split into three new snaxels. d) Without a cleaning conquest both snaxel a and snaxel b would violate condition 3. e) Snaxels a and b are removed by the cleaning conquest. f) Some timesteps later.

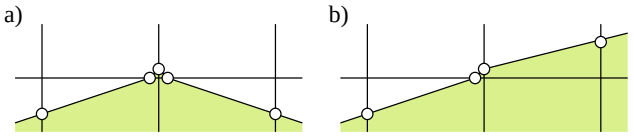

Fig. 11 Multiple vertices: Triple (a) and double (b) vertices may appear after a snaxel split. Although these vertices share the same spatial position, they are conceptually distinct, as they are supported by different grid segments. Due to the parameter independent evolution rule this degenerate snake parameterization does not affect the numerical robustness of the algorithm.

cleaning conquest or not. This will result in different behavior as is demonstrated in Figure 14.

Merging In case topology changes of the snakes are permitted, the two colliding snaxels a and b can be merged by simply relinking their neighboring snaxels. For this we just set

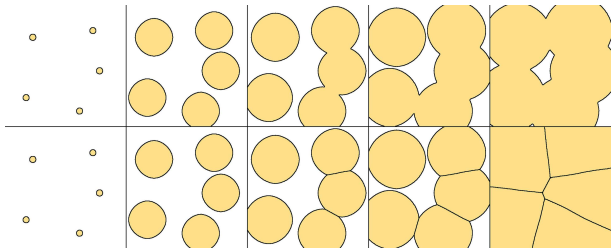

Fig. 12 Collision detection and handling. The above sequences show 5 snakes evolving and colliding. In the top row, the colliding snakes change topology and merge. In the bottom row, the colliding snakes keep their topology and come to a halt.

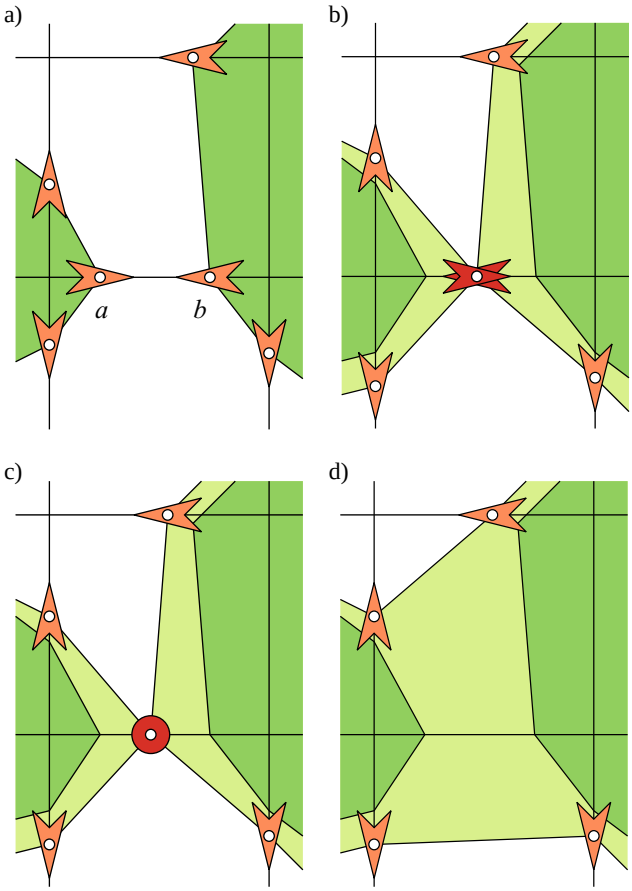

Fig. 13 Collision detection and handling: a) Original r-snake, a potential collision is detected as snaxels a and b are supported by the same grid segment. b) The timestep is adapted such that snaxels a and b do not cross but just touch each other. c) If the topology of the snakes must not change, the snaxels that have collided are frozen and excluded from further updating. d) If topology changes are wanted, snaxels a and b are removed and their previous/following snaxels are connected such that the two parts of the snakes merge.

```
a->next->prev = b->prev
b->prev->next = a->next
b->next->prev = a->prev
a->prev->next = b->next
```

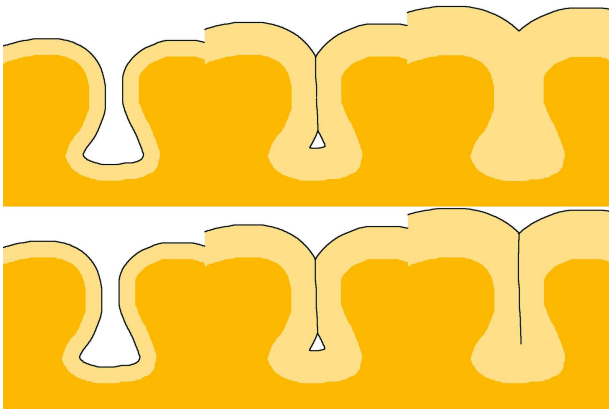

Fig. 14 Depending on whether frozen snaxels are removed by the cleaning conquest, recesses will dissappear as soon as they completely touch each other (top) or are conserved (bottom).

and remove a and b (Figure 13). After that we perform a cleaning conquest to remove spurious bad snaxels. Hence, in contrast to [8] and [18] this operation does not require any resampling. Notice that the topology changes are very simple operations even in this explicit representation setting. This is due to the fact that the restrictions for the snaxel movement guarantee that collisions always happen at the contour vertices (and not at the segments).

5 Results

Synthetic Data Figure 15 shows the evolution of two r--snakes. They are initialized at the outside and at the inside of a polygon, then they are propagated with unit speed $v \equiv 1$. The figure shows snapshots of the contour at equidistant time intervals. As can be seen, the contours obey Huygens' principle and nicely handle concave as well as convex corners.

Real Data In Figure 16 we applied our algorithm to the problem of reconstructing the brain cortex from an MRI image. In this case the grid resolution has been set to the resolution of the image (256×256), but higher resolutions would also be possible. First, the MRI image has been pre-processed by a 3×3 Gaussian filter. Then we initialized two circular r-snakes for each of the two hemispheres. The snaxels' speeds are set proportional to the underlying image intensities. The segmentation process took less than three seconds. In practice, the results could be enhanced by additionally applying well-known standard segmentation techniques, like using internal forces on the r-snake or applying scale-space techniques on the image [13].

Drawbacks Because the snaxels of an r-snake are restricted to move along grid segments, the algorithm sometimes exhibits preferences for certain directions. In particular, when two r-snakes collide in diagonal direction, "ripples" in the order of one pixel's magnitude may appear, as is depicted in Figure 17. These ripples in particular occur in synthetic

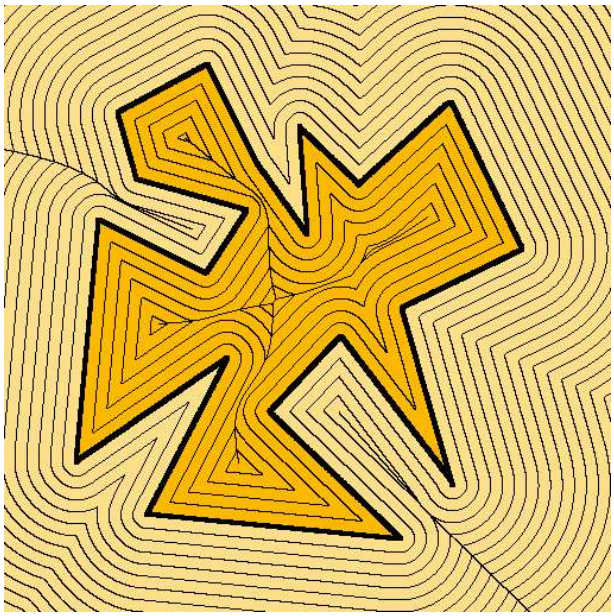

Fig. 15 Contour evolution: The image above shows the evolution of two contours that were initialized as the inner and outer boundary of the polygon. Both contours propagate with unit speed $v \equiv 1$ and are shown at equidistant time intervals. The resolution of the underlying grid is 512×512.

datasets where there is no underlying external force that provides a meaningful gradient which guides the snaxels to their final destinations. Since the ripples represent features at subpixel precision which can be considered as sampling artifacts of the discretized underlying scalar field, we smooth the frozen snaxels of the r-snake in a post-processing step to remove these artifacts.

6 Conclusions and future work

We have presented a novel approach for the representation and evolution of active contours. Its major advantages are

- Ease of implementation
- Automatic adaption of the contour parameterization
- Efficient collision detection and avoidance
- Topology control

We have demonstrated its applicability on synthetic as well as on real image data. Our approach proves to be especially well suited for the reconstruction of convoluted organic structures, like the cortex of the brain.

The major drawback in our current implementation is that we set the time steps by taking a *global* minimum. This could easily be improved by adjusting the time steps locally and integrating the cleaning conquest into the evolution procedure to avoid inconsistent snake configurations. The "ripple"-problem is of minor relevance in real applications since here the speed function is usually dominated by the external energy forces. In the future, we plan to generalize our algorithm

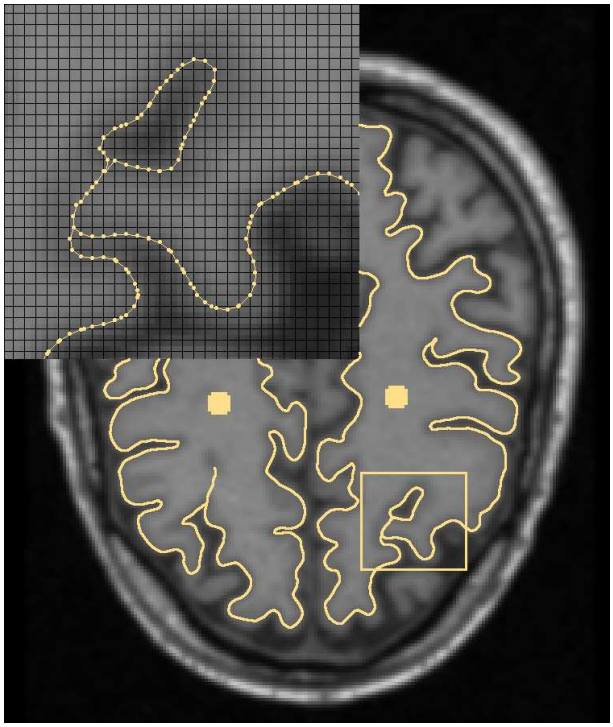

Fig. 16 Contour evolution: The image above shows the evolution of two r-snakes in order to segment the brain cortex in an MRI image. The speed of a snaxel is proportional to the image intensity at the position of the snaxel. Note in particular the gap-less seam that reconstructs the "intensity valleys" as shown in the closeup.

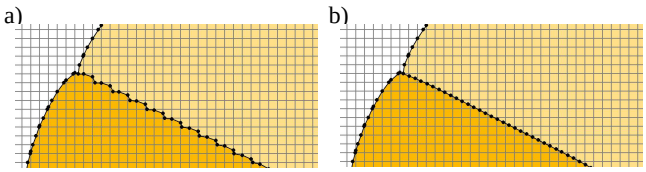

Fig. 17 Ripples: When two r-snakes clash together, there may appear ripples in the order of one pixel's magnitude (a). These ripples can be removed in a post-processing step by smoothing the r-snake (b).

to higher dimensions and to develop a method such that positive as well as negative speeds can be applied in one update step.

References

1. M. O. Berger. Snake growing. In *Proc. First European Conf. on Computer Vision*, pages 570–572. Springer LNCS, 1990.
2. D. L. Chopp. Computing minimal surfaces via level set curvature flow. *Jour. of Comp. Phys.*, 106:77–91, 1993.
3. I. Cohen, L. Cohen, and N. Ayache. Using deformable surfaces to segment 3-d images and infer differential structures. *Computer Vision, Graphics and Image Processing: Image Understanding*, 56(2):242–263, 1992.
4. L. Cohen. On active contour models and balloons. *Computer Vision, Graphics and Image Processing: Image Understanding*, 53(2):211–218, 1991.
5. L. D. Cohen and I. Cohen. Finite element methods for active contour models and balloons for 2d and 3d images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, 1993.
6. C. A. Davatzikos and J. L. Prince. An active contour model for mapping the cortex. *IEEE Trans. on Medical Imaging*, 14(1):112–115, 1995.
7. D. DeCarlo and D. Metaxas. Blended deformable models. In *Proceedings CVPR '94*, pages 566–572, 1994.
8. H. Delingette and J. Montagnat. Shape and topology constraints on parametric active contours. *Computer Vision and Image Understanding*, 83:140–171, 2001.
9. M. Gage. On an area-preserving evolution equation for plane curves. *Contemp. Math.*, 51:51–62, 1986.
10. A. Gupta, T. O'Donnell, and A. Singh. Segmentation and tracking of cine cardiac mr and ct images using a 3d deformable model. In *IEEE Conf. on Computers and Cardiology*, pages 661–664, 1994.
11. X. Han, C. Xu, and J. L. Prince. A topology preserving deformable model using level sets. In *Computer Vision and Pattern Recognition Proceedings*, pages 765–770, 2001.
12. J. Hug, C. Brechbühler, and G. Szekely. Tamed snake: A particle system for robust semi-automatic segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, number 1679 in LNCS, pages 106–115, 1999.
13. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Internation Journal of Computer Vision*, 1:321–331, 1988.
14. B. Kimia, A. Tannenbaum, and S. Zucker. On the evolution of curves via a function of curvature i. the classical case. *Journal of Mathematical Analysis and Applications*, 163:438–458, 1992.
15. J.-O. Lachaud and A. Montanvert. Deformable meshes with automatic topology changes for coarse-to-fi ne three-dimensional surface extraction. *Medical Image Analysis*, 3(2):187–207, 1999.
16. S. Lobregt and M. Viergever. A discrete dynamic contour model. *IEEE Trans. on Medical Imaging*, 14(1):12–23, 1995.
17. W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface reconstruction algorithm. In *SIGGRAPH 87 proceedings*, pages 163–169, 1987.
18. T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *International Conference on Computer Vision*, pages 840–845, 1995.
19. T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: A survey. *Medical Image Analysis*, 1(2):91–108, 1996.
20. T. McInerney and D. Terzopoulos. Topology adaptive deformable surfaces for medical image volume segmentation. *IEEE Transactions on Medical Imaging*, 18(10):840–850, 1999.
21. T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.
22. J. V. Miller, D. E. Breen, W. E. Lorensen, R. M. O'Bara, and M. J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. In *SIGGRAPH '91 Proceedings*, pages 217–226, 1991.
23. F. Prêteux N. Rougon. Directional adaptive deformable models for segmentation. *Journal of Electronic Imaging*, 7(1):231–256, 1998.
24. S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

25. S. J. Osher and R. P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.
26. J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci.*, 93(4):1591–1595, 1996.
27. J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

Leif P. Kobbelt is a full professor and the head of the Computer Graphics Group at the Aachen University of Technology, Germany. His research interests include all areas of Computer Graphics and Geometry Processing with a focus on multiresolution and free-form modeling as well as the efficient handling of polygonal mesh data. He was a senior researcher at the Max-Planck-Institute for Computer Sciences in Saarbr¨ucken, Germany from 1999 to 2000 and received his Habilitation degree from the University of Erlangen, Germany where he worked from 1996 to 1999. In 1995/96 he spent a post-doc year at the University of Wisconsin, Madison. He received his Master's (1992) and Ph.D. (1994) degrees from the University of Karlsruhe, Germany. Over the last years he has authored many research papers in top journals and conferences and served on several program committees.

Stephan Bischoff graduated in 1999 with a master's in computer science from the University of Karlsruhe, Germany. He then worked at the graphics group of the Max-Planck-Institute for Computer Science in Saarbr¨ucken, Germany. In 2001 he joined the Computer Graphics Group at the Aachen University of Technology, Germany, where he is currently pursuing his PhD. His research interests focus on freeform shape representations for efficient geometry processing and on topology control techniques for level-set surfaces.