The Visual Computer

Parameterization-free active contour models 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

Published online: 10 March 2004

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 subpixel 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

Correspondence to: S. Bischoff

1 Introduction

For 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 that they make it possible 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 that 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 allow the contour to change its topology in a natural manner (see the next section for a more detailed description).

Our new approach inherits parts of both the explicit and the implicit frameworks: 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 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 easily implemented. 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,

The Visual Computer (2004) 20:217–228
Digital Object Identifier (DOI) 10.1007/s00371-003-0228-9

we can use an explicit Euler integration scheme to trace the contour through the embedding force field.

Overview. In Sect. 2 we give a short overview of previous and related work. Section 3 briefly introduces active contour models. In Sect. 4 we describe our new scheme and give implementation hints. Results for synthetic as well as for real data are presented in Sect. 5. In Sect. 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, 25]. 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 was first 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 [2–4, 8, 12, 19]. Several authors have introduced different explicit representations, e.g., finite element models [2] and subdivision curves [12]. The explicit active contour model has also been generalized to higher dimensions in such a way as to segment volume data like MRI scans of human organs [2, 4, 22]. Early explicit active contour models could not handle topology changes like merging or splitting of contours. To overcome this limitation, several authors have proposed topologically adaptive contour models that 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 [23, 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 [5, 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 improving 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" that is provided by the user and that incorporates all the external and internal forces that account for the quality of the final segmentation.

The combination of implicit and explicit techniques provides us with 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 it is 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 [23, 27] or explicitly

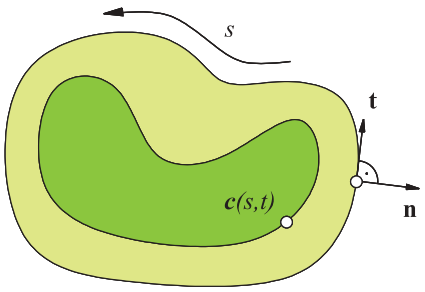


Fig. 1. Evolving contours: A contour $\boldsymbol{c}(s,t)$ is parameterized by arc length s and time t. The movement of each contour point $\boldsymbol{c}(s,t)$ can be decomposed into tangential and normal components. While the tangential component affects the parameterization of the contour, only the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

by a parametric representation [2, 12, 13]. Here we will focus on the latter case.
Consider a contour $\boldsymbol{c}(s,t) \in \mathbb{R}^2$ where $s \in [0,1]$ parameterizes the contour arc and $t \in \mathbb{R}_{\geq 0}$ designates time (Fig. 1).
The evolution of $\boldsymbol{c}$ can then be described by the following equation:

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

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

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

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.

$$\mathcal{E}(\boldsymbol{c}) = \mathcal{E}_{\text{internal}}(\boldsymbol{c}) + \mathcal{E}_{\text{external}}(\boldsymbol{c}), \tag{2}$$

where $\mathcal{E}_{\text{internal}}$ represents the internal energy of the contour and is in general a weighted combination of membrane and thin-plate energy that 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 that 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 (Sect. 4), Eq. 1 can be simplified to

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

i.e., the contour only evolves in the normal direction.
In practice, the contour $\boldsymbol{c}$ 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 $\boldsymbol{c}$ as a polygon in space and in time by a sequence of vertices, so-called *snaxels*,

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

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 Eq. 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., toward uniform or curvature-dependent vertex spacing. However, in general, some local or global vertex insertion/deletion strategies have to be implemented 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 the 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

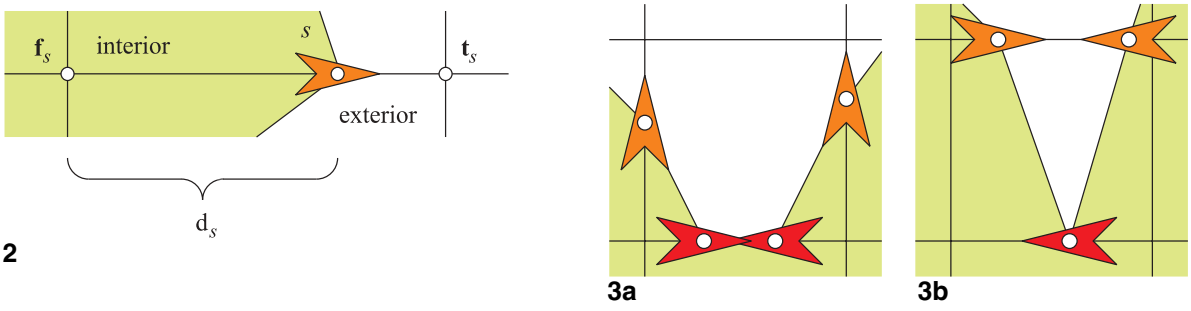


Fig. 2. Snaxel: In this and the other figures, snaxels are represented as arrowheads such as to indicate their "from" and "to" vertices
Fig. 3. Forbidden snaxel configurations: the configurations shown above are forbidden, as the darkened snaxels point from the interior to the interior of the contours

snaxels may only move along the lines of a given, fixed grid. Whenever a snaxel runs into a grid point, 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 discussion 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*, which 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 that we call the *supporting segment* of s. More precisely, for each snaxel s there are two grid points $\boldsymbol{f}_s \in \mathbb{Z} \times \mathbb{Z}$ ("from") and $\boldsymbol{t}_s \in \mathbb{Z} \times \mathbb{Z}$ ("to") such that

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

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

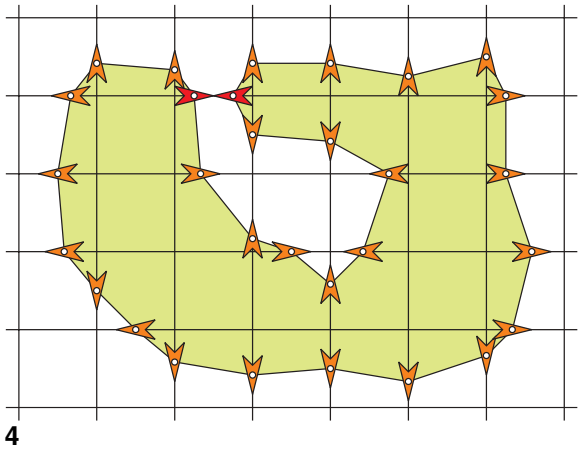
   $$\boldsymbol{p}_s = (1 - d_s)\, \boldsymbol{f}_s + d_s\, \boldsymbol{t}_s$$

   (Fig. 2).
2. *Orientation.* All snaxels are consistently oriented. By convention, each snaxel s points from the interior of $\mathcal{S}$ to the exterior of $\mathcal{S}$ (Fig. 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 Fig. 3 are forbidden, and notches that are thinner than one pixel cannot be represented.

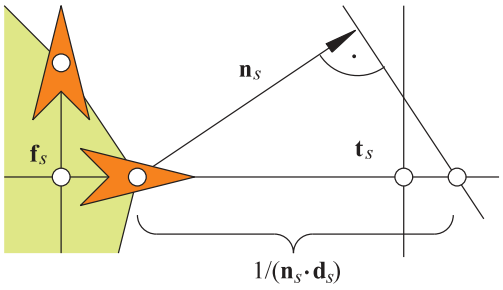
4.2 Implementation

For representing an r-snake, we use a simple data structure. Each snaxel object has eight 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  */
}.
```



4



5

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

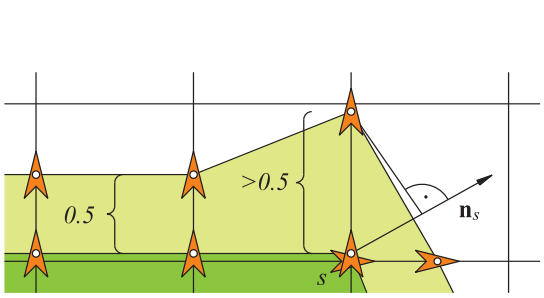
The `next` and `prev` pointers are used to arrange all snaxels of an r-snake in a counterclockwise direction in a doubly-linked list.
An r-snake can be initialized by resampling an arbitrary closed, self-intersection-free curve on the $\mathbb{Z} \times \mathbb{Z}$ grid. If the curve is, e.g., given 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 Fig. 4.

4.3 Evolving an r-snake

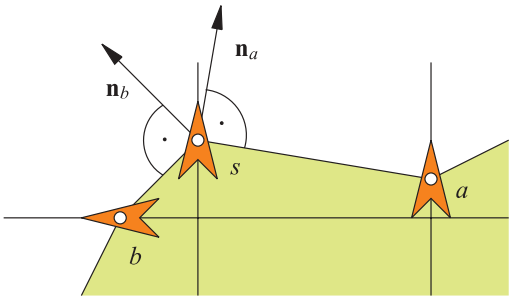
In this section we describe how an r-snake evolves according to the impact of external and internal forces. In the following discussion we will always assume that the r-snake moves (1) in the normal direction and (2) only outward.
In general, the tangential component of a force affects a contour's parameterization 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 the 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 $\boldsymbol{n}_s$ is given (this will be explained in more detail below). In general, the normal $\boldsymbol{n}_s$ 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 Fig. 5, the projected snaxel speed can be easily computed as

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

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



6



7

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

snaxel positions:

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

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

There are numerous ways to approximate a normal $\boldsymbol{n}_s$ 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 that evolves with constant (unit) speed $v \equiv 1$ and let $\Delta t = 0.5$. After updating all vertices we expect that the r-snake will have moved outwards by 0.5 units. Actually, however, because we use only approximated normals, cusps and creases may appear, as shown in Fig. 6. In this case, the speed $\tilde{v}_s$ resulting from the projection of $\boldsymbol{n}_s$ is too high and results in an "overshooting" effect. Analogous effects can be observed in the case of a concave corner.

To avoid the "overshooting" and to avoid the necessity of approximating 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 (Fig. 7). Then we set

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

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

- Both normals lie on the same side of the supporting segment of s (Fig. 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 (Fig. 8b). In this case the final projected speed should be set to

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

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 $\boldsymbol{n}_a$, $\boldsymbol{n}_b$ and the supporting segment (Fig. 8c and d). Both cases lead to the same formula, namely,

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

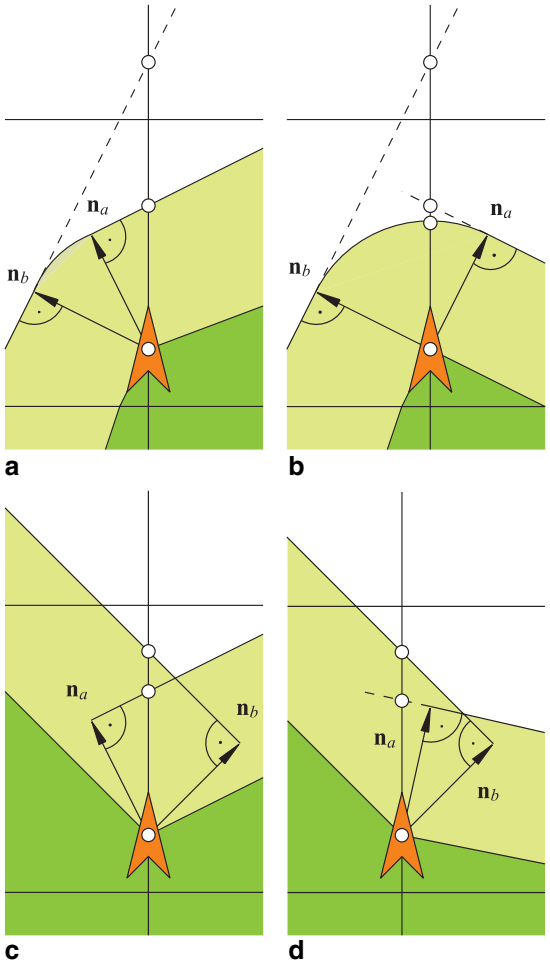


Fig. 8. Snaxel speeds are computed by applying Huygen's principle on continuous contours: the original contour (*dark*) is locally propagated with constant speed. The intersection points of the resulting continuous offset contour (*light*) with the grid are then used to determine the new snaxel positions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

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 will not cross a grid point "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 grid point $\boldsymbol{x} = \boldsymbol{t}_s$, i.e., whenever

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

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

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

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

After a snaxel split, condition 3 of Sect. 4.1 might be violated, as is depicted in Fig. 10. To reestablish the r-snake property, we perform a cleaning conquest: all snaxels that violate condition 3 are simply 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 Fig. 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 $\tilde{v}_s = v_s$.

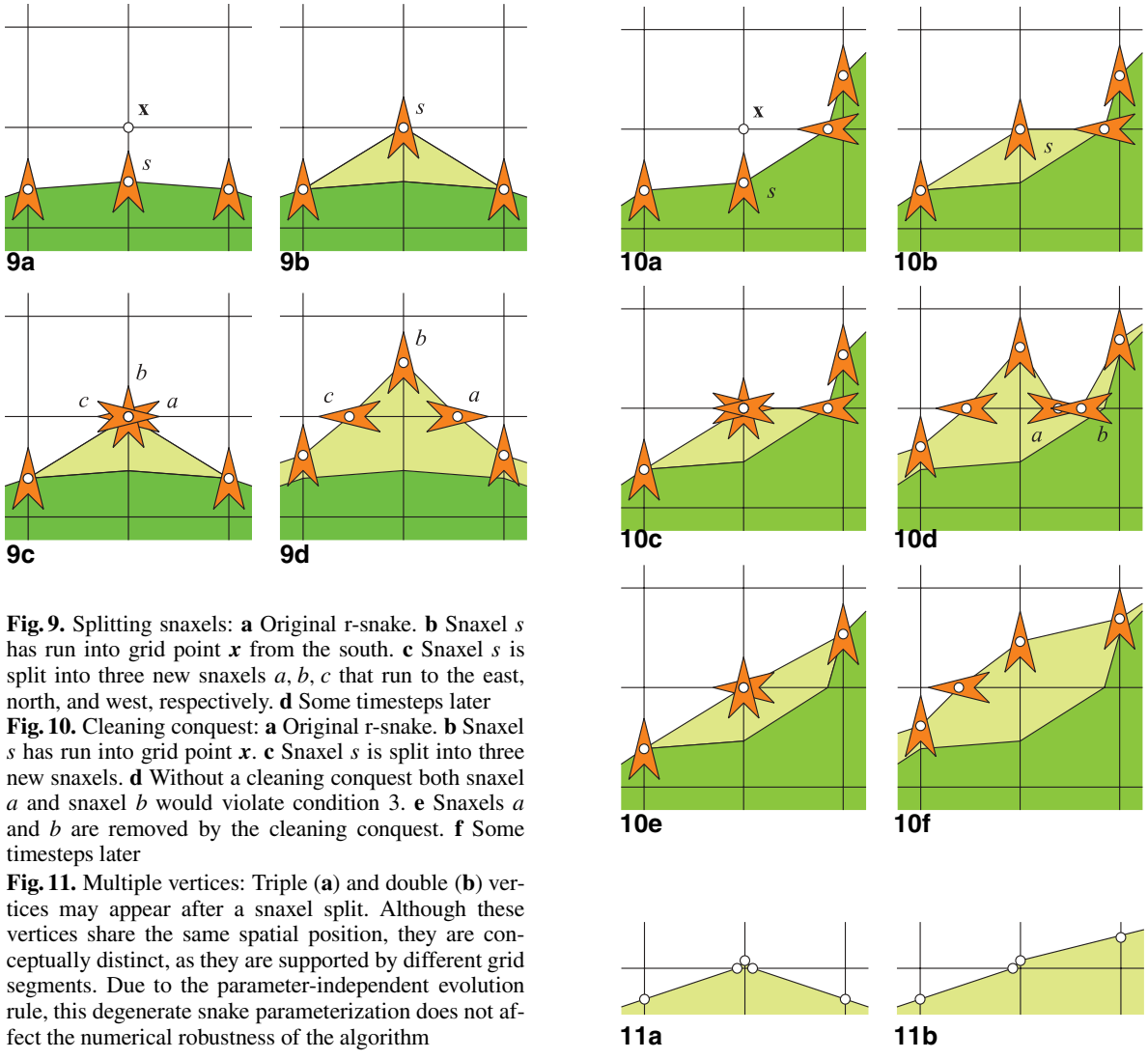


Fig. 9. Splitting snaxels: **a** Original r-snake. **b** Snaxel s has run into grid point $\boldsymbol{x}$ from the south. **c** Snaxel s is split into three new snaxels a, b, c that run to the east, north, and west, respectively. **d** Some timesteps later

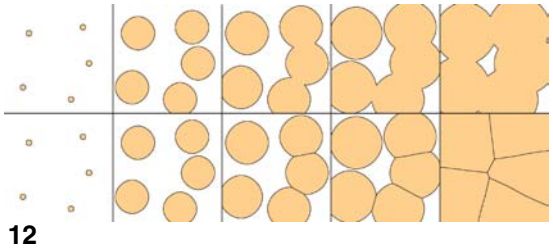
Fig. 10. Cleaning conquest: **a** Original r-snake. **b** Snaxel s has run into grid point $\boldsymbol{x}$. **c** Snaxel s is split into three new snaxels. **d** Without a cleaning conquest both snaxel a and snaxel b would violate condition 3. **e** Snaxels a and b are removed by the cleaning conquest.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

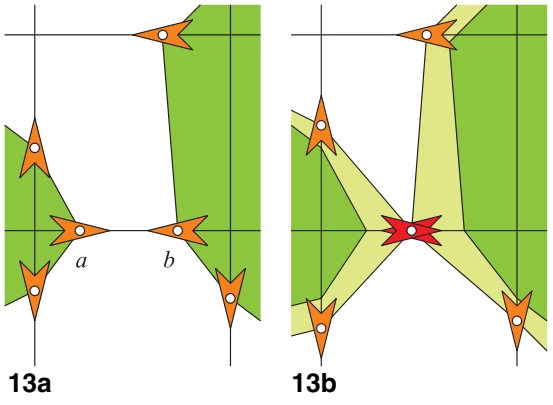
Fig. 11. Multiple vertices: Triple (**a**) and double (**b**) vertices may appear after a snaxel split. Although these vertices share the same spatial position, they are conceptually distinct, as they are supported by different grid segments. Due to the parameter-independent evolution rule, this degenerate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

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 decide whether the two colliding snaxels will *clash* and come to a halt (topology preservation) or whether they will *merge* (topology change) (Fig. 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 (Fig. 13). We can further decide whether or not frozen snaxels will be affected by the cleaning conquest. This will result in different behavior, as is demonstrated in Fig. 14.





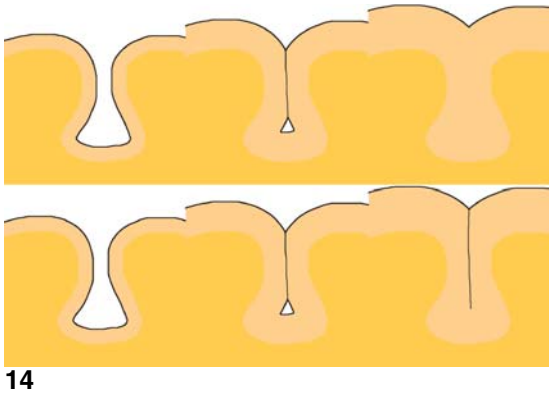


Fig. 12. Collision detection and handling. The above sequences show five snakes evolving and colliding. In the *top row*, the colliding snakes change topology and merge. In the *bottom row*, the colliding snakes keep their topology and come to a halt

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/subsequent snaxels are connected such that the two parts of the snakes merge

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*)

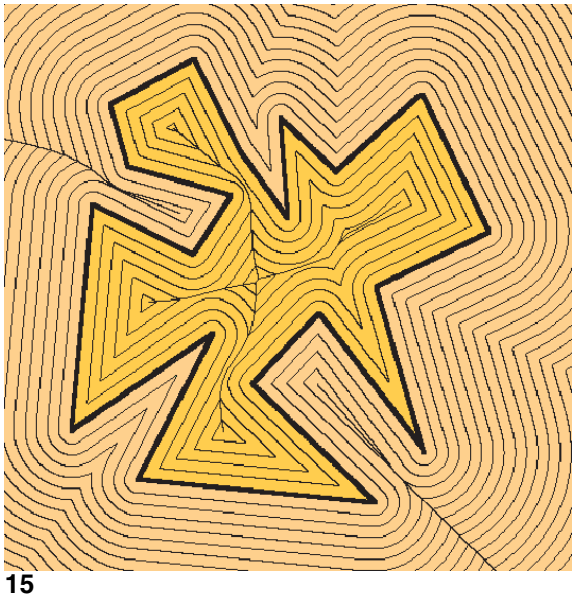
Merging. If topology changes of the snakes are permitted, the two colliding snaxels a and b can be merged by simply relinking their neighboring snaxels. For this we just set

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

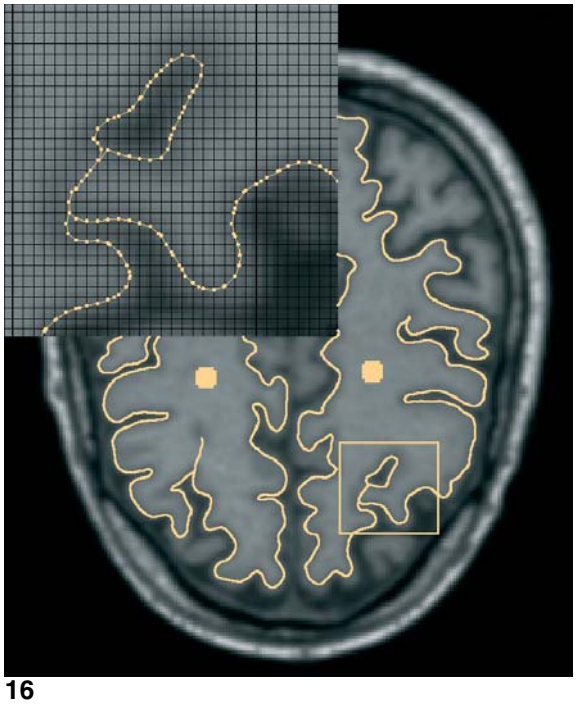
and remove a and b (Fig. 13). After that we perform a cleaning conquest to remove spurious bad snaxels. Hence, in contrast to [8] and [18], this operation does not require any resampling. Notice that the topology changes are very simple operations even in this explicit representation setting. This is 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 and then propagated with unit speed $v \equiv 1$. The figure shows snap-



15



16

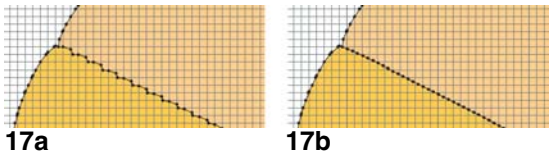
Fig. 15. Contour evolution: The image above shows the evolution of two contours that were initialized as the inner and outer boundary of the polygon. Both contours propagate with unit speed $v \equiv 1$ and are shown at equidistant time intervals. The resolution of the underlying grid is 512×512

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 gapless seam that reconstructs the "intensity valleys" as shown in the closeup

shots 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 Fig. 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 preprocessed 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 3 s. 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 a diagonal direction, "ripples" in the order of one pixel's magnitude may appear, as is depicted in Fig. 17. These ripples in particular occur in synthetic datasets where there is no underlying external force that provides a meaningful gradient that guides the snaxels to their final destinations. Since the ripples represent features at subpixel precision that can be considered as sampling artifacts of the discretized underlying scalar field, we smooth the frozen snaxels of the r-snake in a postprocessing step to remove these artifacts.



17a 17b

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

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, and
- 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 timesteps by taking a *global* minimum. This could easily be improved by adjusting the timesteps locally and integrating the cleaning conquest into the evolution procedure to avoid inconsistent snake configurations. The "ripple" problem is of minor importance 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 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. Berger MO (1990) Snake growing. In: Proceedings of the 1st European conference on computer vision. Lecture notes in computer science. Springer, Berlin Heidelberg New York, pp 570–572
2. Cohen I, Cohen L, Ayache N (1992) Using deformable surfaces to segment 3-d images and infer differential structures. Comput Vision Graph Image Process Image Understand 56(2):242–263
3. Cohen L (1991) On active contour models and balloons. Vision Graph Image Process Image Understand 53(2):211–218
4. Cohen LD, Cohen I (1993) Finite element methods for active contour models and balloons for 2d and 3d images. IEEE Trans Patt Anal Mach Intell 15(11):1131–1147
5. Chopp DL (1993) Computing minimal surfaces via level set curvature flow. J Comput Phys 106:77–91
6. Davatzikos CA, Prince JL (1995) An active contour model for mapping the cortex. IEEE Trans Med Imag 14(1):112–115
7. DeCarlo D, Metaxas D (1994) Blended deformable models. In: Proceedings of CVPR '94, pp 566–572
8. Delingette H, Montagnat J (2001) Shape and topology constraints on parametric active contours. Comput Vision Image Understand 83:140–171
9. Gage M (1986) On an area-preserving evolution equation for plane curves. Contemp Math 51:51–62
10. Gupta A, O'Donnell T, Singh A (1994) Segmentation and tracking of cine cardiac mr and ct images using a 3d deformable model. In: Proceedings of the IEEE conference on computers and cardiology, pp 661–664
11. Han X, Xu C, Prince JL (2001) A topology preserving deformable model using level sets. In: Proceedings of the conference on computer vision and pattern recognition, pp 765–770
12. Hug J, Brechbühler C, Szekely G (1999) Tamed snake: a particle system for robust semi-automatic segmentation. In: Proceedings of the conference on medical image computing and computer-assisted intervention. Lecture notes in computer science, vol 1679. Springer, Berlin Heidelberg New York, pp 106–115
13. Kass M, Witkin A, Terzopoulos D (1988) Snakes: active contour models. Int J Comput Vision 1:321–331
14. Kimia B, Tannenbaum A, Zucker S (1992) On the evolution of curves via a function of curvature i: the classical case. J Math Anal Appl 163:438–458
15. Lachaud J-O, Montanvert A (1999) Deformable meshes with automatic topology changes for coarse-to-fine three-dimensional surface extraction. Med Image Anal 3(2):187–207
16. Lobregt S, Viergever M (1995) A discrete dynamic contour model. IEEE Trans Med Imag 14(1):12–23
17. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3d surface reconstruction algorithm. In: Proceedings of SIGGRAPH '87, pp 163–169
18. McInerney T, Terzopoulos D (1995) Topologically adaptable snakes. In: Proceedings of the international conference on computer vision, pp 840–845
19. McInerney T, Terzopoulos D (1996) Deformable models in medical image analysis: a survey. Med Image Anal 1(2):91–108
20. McInerney T, Terzopoulos D (1999) Topology adaptive deformable surfaces for medical image volume segmentation. IEEE Trans Med Imag 18(10):840–850
21. McInerney T, Terzopoulos D (2000) T-snakes: topology adaptive snakes. Med Image Anal 4(2):73–91
22. Miller JV, Breen DE, Lorensen WE, O'Bara RM, Wozny MJ (1991) Geometrically deformed models: a method for extracting closed geometric models from volume data. In: Proceedings of SIGGRAPH '91, pp 217–226

23. Osher SJ, Fedkiw RP (2002) Level set methods and dynamic implicit surfaces. Springer, Berlin Heidelberg New York
24. Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. J Comput Phys 79(1):12–49
25. Prêteux F, Rougon N (1998) Directional adaptive deformable models for segmentation. J Electron Imag 7(1):231–256
26. Sethian JA (1996) A fast marching level set method for monotonically advancing fronts. Proc Natl Acad Sci 93(4):1591–1595
27. Sethian JA (1999) Level set methods and fast marching methods. Cambridge University Press, Cambridge, UK

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 Saarbrücken, Germany from 1999 to 2000 and received his Habilitation degree from the University of Erlangen, Germany, where he worked from 1996 to 1999. In 1995/1996 he spent a postdoc year at the University of Wisconsin, Madison. He received his Master's (1992) and Ph.D. (1994) degrees from the University of Karlsruhe, Germany. In recent 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 Saarbrücken, Germany. In 2001 he joined the Computer Graphics Group at the Aachen University of Technology, Germany, where he is currently pursuing his Ph.D. His research interests focus on free-form shape representations for efficient geometry processing and on topology control techniques for level set surfaces.