Published in final edited form as:
IEEE Trans Image Process. 2008 May ; 17(5): 645–656. doi:10.1109/TIP.2008.920737.

A Real-Time Algorithm for the Approximation of Level-Set-Based Curve Evolution

Yonggang Shi [IEEE Member] and
Electrical and Computer Engineering Department, Boston University, Boston, MA 02215 USA. He is now with the Laboratory of Neuro Imaging, Department of Neurology, School of Medicine, University of California, Los Angeles, CA 90095 USA

William Clem Karl [Senior Member, IEEE]
Electrical and Computer Engineering Department and the Biomedical Engineering Department, Boston University, Boston, MA 02215 USA

Yonggang Shi: yshi@loni.ucla.edu; William Clem Karl: wckarl@bu.edu

Abstract

In this paper, we present a complete and practical algorithm for the approximation of level-set-based curve evolution suitable for real-time implementation. In particular, we propose a two-cycle algorithm to approximate level-set-based curve evolution without the need of solving partial differential equations (PDEs). Our algorithm is applicable to a broad class of evolution speeds that can be viewed as composed of a data-dependent term and a curve smoothness regularization term. We achieve curve evolution corresponding to such evolution speeds by separating the evolution process into two different cycles: one cycle for the data-dependent term and a second cycle for the smoothness regularization. The smoothing term is derived from a Gaussian filtering process. In both cycles, the evolution is realized through a simple element switching mechanism between two linked lists, that implicitly represents the curve using an integer valued level-set function. By careful construction, all the key evolution steps require only integer operations. A consequence is that we obtain significant computation speedups compared to exact PDE-based approaches while obtaining excellent agreement with these methods for problems of practical engineering interest. In particular, the resulting algorithm is fast enough for use in real-time video processing applications, which we demonstrate through several image segmentation and video tracking experiments.

Index Terms

Curve evolution; image segmentation; integer operation; level set; real-time; video tracking

I. Introduction

Curve evolution is a powerful technique in many image and video analysis problems. In curve evolution applications, an initial curve is specified in the image domain and its location is updated iteratively according to a speed function, which is designed to localize the object boundary of interest. For the numerical implementation of curve evolution, the level-set method [1]–[3] has become especially popular [4]–[17] because of its many advantages, such as the automatic handling of topological changes, general numerical schemes for high dimensions, etc. Conventionally, the level-set implementation of the curve evolution process is based on the solution of certain partial differential equations (PDEs). This PDE-based solution results in a significant computational burden and has limited the use of level-set-based curve evolution methods in real-time applications, such as video-based object tracking. In this paper, we present a complete and practical algorithm for the approximation of level-set-based curve evolution corresponding to a broad class of evolution speeds which avoids the need to solve PDEs. With our fast algorithm, we can obtain real-time performance while retaining excellent agreement with PDE-based approaches on problems of practical engineering interest, such as the tracking of deformable objects in video sequences.

The basic idea of the level-set method is to represent the curve implicitly as the zero level set of a function defined over a regular grid, which is then evolved according to the specified speed function. Two different approaches to evolving the zero-level set have been taken. The first approach updates the level-set function globally over the entire grid. Perhaps the most well known example in this category is the Chan–Vese (CV) image segmentation model [8], which is associated with a particular choice of curve evolution speed function. The second approach to efficient level-set updates focuses on the local evolution of the curve provided at initialization, and only solves the level-set PDE around a narrow band in the neighborhood of the zero level set to evolve the curve according to its speed [18]. When the zero level set comes too close to the edge of the tube, both the tube and the level-set function generally have to be reinitialized, which is typically done using the fast marching method [19]–[21]. The fast algorithm that we propose in this paper falls into the category of narrow band methods.

Considerable research has been done with the aim of reducing the computational cost for both types of level-set methods. Techniques to approximate and speed up the CV model and its associated speed function include [22]–[26]. In the category of narrow-band level-set methods, many efficient algorithms have also been developed, which we focus on next. In [27], the level-set function is reinitialized by solving a Hamilton–Jacobi PDE for a fixed number of steps at every iteration of the evolving level-set function. In the sparse-field algorithm proposed in [28], the reinitialization is achieved by computing the signed distance function only approximately. Recently, a hardware implementation [29] of this algorithm using a graphics processor (GPU) is reported to achieve 10 to 15 times speed improvement over its software implementation in the Insight Toolkit (ITK) software package [30]. For both methods, a bandwidth of at least five pixels has to be selected to allow numerical evaluation of all the required gradients.

In [12], the *Hermes* algorithm is also proposed to speed up the evolution of the level-set function in a narrow band. In this algorithm, all points on the zero level set are sorted according to their speed and the point with the highest magnitude is picked. The fast marching method is then used to propagate the curve locally within a circular window centered at this point. This algorithm achieves speedups similar to the sparse-field algorithm in [28].

In [31], a fast algorithm for the geodesic active contour model [5], [6] is proposed using the additive operator splitting (AOS) scheme, which was first proposed in [32] and [33] and later discovered independently and popularized by [34] in the image processing community. This algorithm also restricts the computation in a narrow band and enables bigger time steps for the level-set implementation of the geodesic model. The limitation of this algorithm is that it is restricted to a specific model and it needs to maintain the level-set function as a signed distance function, which is computationally expensive.

Compared with the original level-set method for curve evolution [1], narrow banding techniques restrict the solution of the level-set PDE to a much smaller region than the whole grid, and significantly speed up the curve evolution process. However, to maintain numerical stability in the PDE solution process, small time steps must be chosen, the level-set function must be reinitialized from time to time, and the narrow band must also be updated as the curve evolves over time. In addition, many applications require the calculation of, e.g., level set curvature, which is costly. Such factors make the computational cost of these narrow banding algorithms still too high for real-time performance. To address these limitations, in this paper, we propose a novel segmentation algorithm that retains key advantages of the level-set approach in a practical and highly efficient method. We start by using a novel element switching mechanism between two linked lists to realize a general class of curve evolutions efficiently with only integer operations. The two linked lists represent a narrow band of width one that defines the evolving contour implicitly. Over this narrow band, we approximate the level-set function with a limited set of integer values. In addition, we separate the evolution process into two different cycles. In the first cycle we evolve the curve according to data-dependent terms, while in the second cycle we introduce smoothness regularization. Motivated by the diffusion generated curvature motion in [35]–[37], we implement the smoothness cycle by evolving the curve according to a smoothing speed derived from a Gaussian filtering process. Using our element-switching mechanism, we can replace real-valued and PDE-based calculations by inexpensive integer-based ones coupled with simple sign checks to realize efficient curve evolution in both of the cycles.

Thus, to achieve speed, we make certain approximations, reduce the domain of computation, and are careful to focus on integer operations, yet, as we show, the approach is both applicable to a broad class of problems of engineering interest and obtains results that are practically desirable and in excellent agreement with the corresponding exact PDE-based solutions. The integer-based nature of our algorithm is well matched for hardware implementation. As we will demonstrate in our experiments, real-time performance can be achieved easily with our algorithm. The approach is applicable to a broad range of evolution speeds that are composed of a data term and a smoothness regularization term. Among fast algorithms [22]–[26] that solve the CV model, our algorithm is most closely related to the work in [22] since both algorithms separate the evolution process into two different cycles. However, the element switching mechanism we use in both cycles to realize curve evolution is novel. Further, existing fast algorithms solving the CV model update the level-set function over the whole grid, while in our algorithm all computations are limited to a narrow band of width one. In addition, our algorithm is not limited to a single specific speed function, being applicable to a broader class of speeds that can be derived from both region-based and edge-based models.

Similar to the narrow band sparse field algorithm [28], we represent the evolving curve through a narrow-banded linked list structure. However, our narrow band is only of width one, our structure only requires a limited set of integer values, and we use the diffusion generated curvature motion ideas from [35] and [36] to incorporate curve smoothing. Similar to the work in [23], we will only use the sign of the level-set function to determine the location of the object boundary; thus, the accuracy of our method is also at the pixel level. To achieve sub-pixel accuracy, one can always use a solution with pixel level accuracy as initialization of a PDE-based algorithm. Before proceeding, it is worth pointing out that real-time curve evolution results have been reported in [38]–[41] for parametric contour representations, but these methods are not level-set based and have difficulties in handling complicated topological changes and 3-D surface evolution. Fast segmentation can also be achieved with efficient graph cut techniques [42], [43], but they are not as flexible as level-set methods in controlling geometric properties of the object boundary. Our goal in this paper is to reduce the computation time of *level-set-based* curve evolution to achieve real-time results.

II. Analysis of Implicit Curve Evolution

In this section, we review and analyze the motion of an implicitly represented curve on a discrete grid. Based on this analysis, we propose our strategy for the realization of level-set-based curve evolution. Our analysis also applies to the evolution of implicitly represented surfaces in $\mathbb{R}^K$ ($K \geq 2$), but we will use the 2-D terminology “curve” in our description for simplicity.

In the level-set method, a curve *C* is represented implicitly as the zero level set of a function φ defined over a regular grid *D* of size $M_1 \times M_2 \times \cdots \times M_K$. Without loss of generality, we assume the grid is sampled uniformly and the sampling interval is one. The coordinates of a point in the grid are given as $\mathbf{x} = (x_1, x_2, \ldots, x_K)$. We denote the set of grid points enclosed by *C* as the object region Ω and the set of points $D\backslash\Omega$ as the background region. For many image analysis problems, the goal in using curve evolution is to classify each point in the grid into either the object or the background region. With the level-set representation, this classification is determined by the sign of φ.

Given the implicit representation, we can define the list of inside neighboring grid points L_{in} and outside neighboring grid points L_{out} for the object region Ω as follows:

$$\begin{aligned} L_{\text{in}} &= \{\mathbf{x} | \mathbf{x} \in \Omega \text{ and } \exists \mathbf{y} \in N(\mathbf{x}) \text{ such that } \mathbf{y} \in D\backslash\Omega\} \\ L_{\text{out}} &= \{\mathbf{x} | \mathbf{x} \in D\backslash\Omega \text{ and } \exists \mathbf{y} \in N(\mathbf{x}) \text{ such that } \mathbf{y} \in \Omega\} \end{aligned} \tag{1}$$

where $N(\mathbf{x}) = \left\{ \mathbf{y} \in D \middle| \sum_{k=1}^{K} |y_k - x_k| = 1 \right\}$ is a discrete neighborhood of **x**. Various choices of discrete neighborhoods exist from the theory of digital topology [44], and the fast algorithm we propose in this paper can be easily generalized to other discrete neighborhoods.

We illustrate in Fig. 1(a) an example of an implicitly represented curve and the two lists of neighboring grid points for the region enclosed by the curve. For a fixed curve *C*, the definition of φ can be arbitrary, but the two lists of neighboring grid points L_{in} and L_{out} are fixed once we choose a definition of neighborhoods for the object and background region. Conversely, given the two lists L_{in} and L_{out}, the location of the curve can be determined up to the accuracy of the sampling grid as the boundary between the sets of pixels defined by L_{in} and L_{out}. This motivates us to analyze the relation between the motion of this implicitly represented curve and the two lists L_{in} and L_{out}.

Consider the following curve evolution equation:

$$\frac{dC}{dt} = F\vec{N} \tag{2}$$

which evolves the curve *C* according to the speed field *F* in the normal direction $\vec{N}$. In the classical level-set method [1] this evolution is achieved by numerically solving the following PDE on the regular grid

$$\frac{d\varphi}{dt} + F|\nabla\varphi| = 0 \tag{3}$$

where φ is the level-set function. As the function φ evolves continuously, so does the implicitly represented curve *C*. We illustrate in Fig. 1(b) the result of an evolution process of the curve *C* shown in Fig. 1(a). At the location of grid point *A*, the curve moves outward as the value of φ at point *A* changes from positive to negative. At the location of grid point *B*, the curve moves inward and splits into two curves as the value of φ at point *B* changes from negative to positive. This all happens nicely in the level-set method except that it is computationally intensive to solve the PDE in (3) and difficult to meet real-time requirements. However, if we only care about the object and background region on the discrete grid determined by the final location of the curve *C*, the same result can be achieved easily if we use the relation between *C* and

L_{in} and L_{out}. To move the curve outward at the grid point *A*, we just need to switch the grid point *A* from L_{out} to L_{in}. Similarly, we only need to switch the grid point *B* from L_{in} to L_{out} to move the curve inward. By applying such procedures to all points in L_{in} and L_{out}, we can move *C* inward or outward one grid point everywhere along the curve with minimal computation. By repeating the switching operations, we can achieve arbitrary object boundaries defined on the sample grid. Note that this switching process always moves the curve at least one pixel, while the exact level-set method generates sub-pixel movement. The pixel-based switching process, while approximating the sub-pixel level-set-based curve evolution process, results in the same classification for the object and background region. This observation forms the foundation of our fast implementation, which combines grid-level switch-based evolution with integer computation and level-set representation.

III. Fast Algorithm for Geometry Independent Speeds

In this section, we present a basic version of our fast algorithm for level-set-based curve evolution to illustrate the main points. We assume the evolution speed is independent of the geometric properties of the curve (i.e., depends only on the data term). Convergence results of the algorithm for this simpler special case are presented.

A. Basic Algorithm

The basic elements of our representation and curve evolution scheme are presented here. The data structure for our fast algorithm is quite simple and consists of:

- an integer array $\hat{\varphi}$ for the level-set function;
- an integer array $\hat{F}$ for the speed function;
- two lists of grid points adjacent to the evolving curve *C:* L_{in} and L_{out}.

We call those grid points inside *C* but not in L_{in} *interior points* and those points outside *C* but not in L_{out} *exterior points*. For faster computation, we choose the value of the level-set function from a limited set of integers {−3, −1, 1, 3}. This function locally approximates the signed distance function and is defined as follows:

$$\widehat{\varphi}(\mathbf{x})=\begin{cases}3, & \text{if } \mathbf{x} \text{ is an exterior point}\\ 1, & \text{if } \mathbf{x}\in L_{\text{out}}\\ -1, & \text{if } \mathbf{x}\in L_{\text{in}}\\ -3 & \text{if } \mathbf{x} \text{ is an interior point.}\end{cases} \tag{4}$$

Only the sign of the evolution speed *F* is used in our algorithm and we represent it as an integer-valued array $\hat{F}$ with values +1, 0 or −1. Since, in this section, we assume the speed is independent of the geometric properties of the curve, our selection of an integer-valued level-set function will not affect the accuracy in the evaluation of *F*. The two lists L_{in} and L_{out} are bidirectionally linked such that insertion and deletion can be done easily.

Before we present the details of the basic algorithm, let us first define two basic procedures on our data structure. The procedure switch_in() effectively moves the boundary outward by one pixel. For a point $\mathbf{x} \in L_{\text{out}}$, it is defined as follows.

The first step in the switch_in() procedure switches **x** from L_{out} to L_{in}. With $\mathbf{x} \in L_{\text{in}}$ now, all its neighbors that were exterior points become neighboring grid points and are added to L_{out} in the second step. By applying a switch_in() procedure to any point in L_{out}, the boundary is moved outward by one grid point at that location.

Similarly, the procedure switch_out() effectively moves the boundary inward by one pixel. For a point $\mathbf{x} \in L_{\text{in}}$, it is defined as follows.

By applying a switch_out() procedure to an inside neighboring grid point, we move the boundary inward by one grid point at that location.

With the basic steps of moving neighboring grid points defined, we propose the basic version of our fast algorithm. At every iteration, we first compute the speed at each point in L_{out} and L_{in} and store the sign of the speed in the array $\hat{F}$. After that, we scan through the two lists sequentially to evolve the curve. More specifically, we first scan through the list L_{out} and apply a switch_in() procedure at a point if $\hat{F} > 0$. This scan takes care of those parts of the curve with positive speed and moves them outward by one grid point. After this scan, some of the points in L_{in} become interior points due to the newly added inside neighboring grid points, so they are deleted from L_{in}. We then scan through the list L_{in} and apply a switch_out() procedure for a point with $\hat{F} < 0$. This scan moves those parts of the curve with negative speed inward by one grid point. Similarly, points that have become exterior points are deleted from L_{out} after this scan. After a scan through both lists, a stopping condition is checked. If it is satisfied, we stop the evolution; otherwise, we continue this iterative process. In our implementation, the following stopping condition is used.

The Stopping Condition—The curve evolution algorithm stops if either of the following conditions is satisfied.

a. The speed at all the neighboring grid points satisfies

$$\begin{aligned} \widehat{F}(\mathbf{x}) \le 0 \quad & \forall\, \mathbf{x} \in L_{\text{out}} \\ \widehat{F}(\mathbf{x}) \ge 0 \quad & \forall\, \mathbf{x} \in L_{\text{in}}. \end{aligned} \tag{5}$$

b. A prespecified maximum number of iterations is reached.

The first condition is from the convergence result of Theorem 1 which we present next. We use the second condition as a simple way to avoid limiting pixel-level oscillation that is possible in engineering practice due to the pixel-oriented nature of our evolution. This kind of limiting oscillation is generally not important in most applications with a real-time requirement. The final object boundary is always localized very accurately in our experience.

As a summary, the complete algorithm for our implementation is listed in Table I.

B. Convergence Analysis

Since our goal is to localize the object region in imaging applications, it is interesting to analyze under what condition the above algorithm will converge to the true object region. We present a theorem that ensures that the algorithm in Table I will localize the true object region on the fixed grid with no error as long as certain conditions on the speed $\hat{F}$ and the initial level-set function $\hat{\varphi}$ are satisfied.

Let $\mathcal{A}$ denote a set of points in the grid D. Using the concepts from digital topology, we call $\mathcal{A}$ a *connected* region if for any two points $\mathbf{x}$ and $\mathbf{y}$ in $\mathcal{A}$, there exist a sequence of points $\mathbf{x}_m(m = 0,1,\ldots, M)$ in $\mathcal{A}$ such that $\mathbf{x} = \mathbf{x}_0$, $\mathbf{y} = \mathbf{x}_M$ and $\mathbf{x}_m \in N(\mathbf{x}_{m+1})$, where $N(\cdot)$ is the discrete neighborhood defined in Section II. Let the true object region Ω^* in D be composed of P connected regions $\Omega^*_p(p=1,\ldots,P)$, i.e., $\Omega^* = \cup_{p=1}^{P} \Omega^*_p$. The two lists of boundary grid points for Ω^* according to the definition in (1) are denoted as L^*_{in} and L^*_{out}. We then define two neighborhoods for Ω^*. The interior neighborhood Ω^*_{in} is a set of points inside the object region that satisfies $L^*_{\text{in}} \subseteq \Omega^*_{\text{in}} \subseteq \Omega^*$. Similarly, the exterior neighborhood Ω^*_{out} is a set of grid points outside the object region that satisfies $L^*_{\text{out}} \subseteq \Omega^*_{\text{out}} \subseteq D\backslash\Omega^*$. For a level-set function $\hat{\varphi}$ defined over D, the set of grid points inside its zero level set, i.e., with $\hat{\varphi} < 0$, is denoted as Ω. The symmetric difference between Ω and Ω* is denoted as $\pi(\Omega, \Omega^*)$. With this notation, we have the following convergence result, whose proof is in the Appendix.

Theorem 1—Let the object region Ω* be composed of P connected regions $\Omega^*_p(p=1, \ldots, P)$ and the background region $D\backslash\Omega^*$ be composed of Q connected regions $\Gamma^*_q(q=1, \ldots, Q)$. Assume that the evolution speed satisfies: $\hat{F} > 0$ in Ω^*_{in} and $\hat{F} < 0$ in Ω^*_{out}. At initialization, the level-set function $\hat{\varphi}$ is chosen such that the region Ω inside its zero level set satisfies the following two conditions: 1) $\Omega^*_p \cap \Omega \neq \varnothing$ for all $1 \leq p \leq P$, and $(D\backslash\Omega) \cap \Gamma^*_q = \varnothing$ for all $1 \leq q \leq Q$; 2) $L_{\text{in}} \cup L_{\text{out}} \cup \pi(\Omega, \Omega^*) \subseteq (\Omega^*_{\text{in}} \cup \Omega^*_{\text{out}})$, where L_{in} and L_{out} are the two boundary point lists of Ω. Then the algorithm in Table I will converge to the true object region Ω* in a finite number of iterations.

This theorem ensures that the algorithm in Table I will localize the true object region on the fixed grid with no error as long as conditions on the speed $\hat{F}$ and the initial level-set function $\hat{\varphi}$ are satisfied. The condition on the speed $\hat{F}$ in the interior and exterior region makes the union of these two regions form an *attraction* region for the initial level-set function. There are two conditions on the initial level-set function to guarantee convergence. The first condition intuitively requires the region enclosed by the initial curve C to "touch" every object and background region. The second condition essentially requires the initial curve to be in the *attraction* region formed by the union of Ω^*_{in} and Ω^*_{out}, so that it will be driven by the evolution speed to localize the true object region. Note that there is no topological constraints on the initial curve in Theorem 1 and topological changes are automatically taken care of in our algorithm.

Theorem 1 also provides us with an estimate of the computational complexity of the algorithm in Table I. If the speed field and the initial curve satisfy the condition in Theorem 1, we know that either a switch_in or switch_out procedure will be applied to each point in the region $\pi(\Omega^*, \Omega)$ once and only once as the initial curve converges to the true object boundary. Let $\mathcal{N}_A$ denote the number of points in $\pi(\Omega, \Omega^*)$, and $\mathcal{P}$ denote the maximum number of operations for a switch_in or switch_out procedure, the overall computational cost of the algorithm is then $O(\mathcal{N}_A \mathcal{P})$.

To illustrate the improvement of our algorithm to previous PDE-based narrow band algorithms, we compare results with the sparse-field algorithm proposed in [28], a very efficient narrow band algorithm. An open source implementation of the sparse-field algorithm for imaging applications is available in ITK [30], which facilitates comparison. Our aim is to compare to a readily available standard that others can also reference to. Specialized implementations of this or any algorithm can perform better than general purpose versions.

In [28], a simple example that shrinks a circle with constant speed was implemented with both the global level-set method [1] and the sparse-field algorithm. Using the running times in [28] for different domain sizes, we compute the speedup factor of the sparse-field algorithm over the global level-set method and plot it as a function of the domain size in Fig. 2. We also perform this calculation for our algorithm in Table I for a similar example. In particular, for a domain of size $M \times M$, we shrink a circle of radius $M/3$ with constant speed toward a ball of radius $M/12$. The speed is chosen as 1 inside the small ball and −1 everywhere else. For the global level-set method, we start with the real valued level-set function φ as the signed distance function of the initial circle. Driven by the same speed field, both our fast algorithm and the global level-set method successfully locate the small ball of interest. Since the speed field is very simple here, no reinitialization is necessary for the global level-set method. We have plotted the speedup factor of our algorithm over the global level-set method in Fig. 2. We can see that our algorithm has achieved significantly better speedups than the sparse-field algorithm and the advantage of our algorithm grows as the domain size increases.

Compared with previous narrow band methods, there are several reasons that our algorithm is computationally more efficient. First, our algorithm evolves the curve with no need of solving PDEs, while the major advantages of the level-set method are kept, such as the automatic handling of topological changes, and the generality of the numerical scheme for arbitrary dimensions. Second, there is no step-size control or reinitialization of the level-set function in our algorithm. This leads to faster convergence towards object boundaries with far fewer iterations needed. Third, the size of our computational domain is smaller since we only perform calculations on two lists of neighboring grid points. In addition, all our computations are on integers and this further increases the efficiency of our algorithm.

IV. Fast Two-Cycle Algorithm With Smoothness Regularization

In this section, we propose a fast two-cycle (FTC) algorithm to extend the fast algorithm in Table I to include curvature-dependent smoothness regularization. Here we assume the evolution speed F can be viewed as composed of a data-dependent speed F_d, which is a function of the image data and depends on up to the first order geometric properties of the curve (for example, the normal vector) and a smoothness regularization speed F_{int}, which is typically chosen proportional to the mean curvature, so that $F = F_d + F_{\text{int}}$. This assumption covers a quite general class of variational curve evolution models of engineering interest. The addition of curvature-based speeds poses some challenges. For efficiency, our proposed approach approximates the level-set function in a narrow band using an integer valued array. Curvature calculations performed directly on such discrete valued level-set functions are not good reflections of the true curvature of the underlying curve and can lead to spurious results if used directly [45]. Curvature calculations are possible through multistep approaches which basically interpolate or smooth the approximate level-set function and subsequently calculate the curvature [45]–[47]. However, such calculations remove us from the realm of integer calculations and are computationally expensive, so we want to avoid them.

Instead, we propose a two-cycle algorithm to approximate the curve evolution process according to the general speed $F = F_d + F_{\text{int}}$. In the first cycle, we evolve the curve using the sign of the data-dependent speed F_d, which we denote as $\hat{F}_d$, using the algorithm in Table I. In the second cycle, we introduce smoothness regularization by evolving the curve according to a smoothing speed $\hat{F}_{\text{int}}$ derived from a simple Gaussian filtering process. This second cycles approximates curvature-based evolution, but avoids the need for expensive computations. One can view this as applying an approximation $\hat{F}_{\text{int}}$ to F_{int}. These two cycles are then repeated to evolve the curve towards the object boundary.

The idea of using convolution operations to generate mean curvature motion was originally proposed by Merriman, Bence, and Osher (MBO) in their work on diffusion generated mean curvature motion [35], [36]. Let the indication function of the region Ω inside the curve C be χ_Ω, then the MBO algorithm for mean curvature motion is as follows.

1. Initialize $\chi(\mathbf{x}, 0) = \chi_\Omega(\mathbf{x})$.
2. Apply heat diffusion to χ for some time T by solving

$$\frac{\partial \chi}{\partial t} = \Delta \chi.$$

3. “Sharpen” the diffused function χ by setting

$$\chi=\begin{cases} 1, & \text{if } \chi>\frac{1}{2} \\ 0, & \text{otherwise.} \end{cases}$$

The above steps are then repeated to continue the curvature evolution process. The heat diffusion process in step 2 of the MBO algorithm is equivalent to convolving the region indication function χ with a Gaussian filter. In 2-D, the Gaussian filter is of the form

$$G(\mathbf{x})=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2\sigma^2}|\mathbf{x}|^2\right) \tag{6}$$

where the variance $\sigma^2 = 2T$. As the diffusion time T goes to zero the motion generated by the above “diffusion + thresholding” process converges to the true motion by mean curvature [48]–[51].

Motivated by the MBO algorithm, we propose to evolve the curve according to a *smoothing speed* $\hat{F}_{\text{int}}$ to introduce curvature-dependent smoothing regularization. This smoothing speed is defined as follows for all the boundary points in L_{out}:

$$\widehat{F}_{\text{int}}(\mathbf{x})=\begin{cases} 1, & \text{if } G\otimes H(-\widehat{\varphi})(\mathbf{x})>\frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where H is the Heaviside function and $H(-\hat{\varphi}) = \chi$, which is the indication function of the object region enclosed by the zero level set of $\hat{\varphi}$, and $\otimes$ is the convolution operation. Since $\chi(\mathbf{x}) = 0$ for a point $\mathbf{x} \in L_{\text{out}}$, it will be a member of the object region if the result of the convolution is greater than 1/2. Thus, we define its speed as +1 so that a *switch_in* procedure will be applied to it as we evolve the curve with $\hat{F}_{\text{int}}$. Otherwise, if the convolution result is less than or equal to 1/2, the point $\mathbf{x}$ will remain outside the object region; thus, we define the speed $\hat{F}_{\text{int}} = 0$. Following the same principle, the speed $\hat{F}_{\text{int}}$ at all the boundary points in L_{in} is defined as follows:

$$\widehat{F}_{\text{int}}(\mathbf{x})=\begin{cases} -1, & \text{if } G\otimes H(-\widehat{\varphi})(\mathbf{x})<\frac{1}{2} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Once the smoothing speed $\hat{F}_{\text{int}}$ is computed, we can then apply the same algorithm in Table I to realize the evolution of the curve according to the smoothness regularization. Note that there is a difference between our smoothing cycle and the MBO algorithm. In the MBO algorithm, the diffusion process is solved over the whole grid and the distance each point on the curve moves after one iteration is dependent on the curvature at that point, so it may move across multiple grid points. In our smoothing cycle, all operations are performed only on the two boundary point lists and the curve can at most move by one grid point. Even though the distance the curve moves in our algorithm may differ from the MBO algorithm, the direction of the motion is the same, so our motion still introduces smoothness regularization.

The Gaussian filter G makes the smoothness regularization process dependent on the curvature of the underlying curve C. To resolve the mean curvature motion on a discretized grid with diffusion, it was pointed out in [36] that we need to choose the grid interval $\Delta\mathbf{x} \ll |\kappa| T$, where κ is the curvature, such that the curve will move at least one grid point after one iteration of the MBO algorithm. However, we usually cannot afford to choose an infinitesimal grid interval due to computational resource constraints. For a fixed grid, then, the diffusion time (i.e., the width of the Gaussian kernel) will decide to what extent we can resolve the curvature motion. If both the Gaussian kernel and grid interval are fixed, only those parts of the curve with curvature $|\kappa| > \Delta\mathbf{x}/T$ will be moved after one iteration of the MBO algorithm. Once we fix the grid resolution in our two-cycle algorithm, the width of the Gaussian filter G will then control the smoothness by smoothing out corners or concavities on the object boundary with their curvature above a certain magnitude. As we increase the width of G, more corners or concavities will be eliminated and the final shape becomes smoother. As an extreme, the whole shape will disappear as the width of the Gaussian filter tends to infinity. This is because in this case the smoothing speed $\hat{F}_{\text{int}}(\mathbf{x}) = 0$ for all $\mathbf{x} \in L_{\text{out}}$ and $\hat{F}_{\text{int}}(\mathbf{x}) = -1$ for all $\mathbf{x} \in L_{\text{in}}$ according to the definition in (7) and (8). Thus, the width of the Gaussian filter should be proportional to the scale of the structure we want to keep on the final object boundary.

To maintain efficiency in our numerical implementation of the smoothing cycle, we approximate the Gaussian filter G of variance σ^2 with an array of size $N_g \times \cdots \times N_g$. This will not change the curvature-dependent property of the evolution in our smoothing cycle since, as pointed out in [35] and proved in [52], any positive, and radially symmetric kernel can be used to replace the Gaussian filter to convolve with the indicator function and generate mean curvature motion. Further, because the result of the filtering process is used only for a thresholding process, we can implement it with integer operations after scaling both the filter coefficients and the threshold. As a consequence, the entire smoothing cycle can be performed with integer operations.

As a demonstration of the smoothing cycle, we show in Fig. 3 the evolution of a synthetic shape with the speed $\hat{F}_{\text{int}}$ using only the smoothing cycle of our two-cycle algorithm. This shape is formed by adding a small circle protrusion to a big circle. The Gaussian filter is of size 9×9 with $\sigma = 3$. As we can see from Fig. 3, only those parts close to the small circle move during the evolution process, while the rest of the big circle stays static. This type of curvature thresholded smoothing evolution is consistent with our previous analysis.

Note that this smoothing behavior, arising from approximations performed for computational efficiency, is different than what one obtains from standard level-set-based curvature motion (which, for example, would eventually eliminate both circles). However, we may argue that this thresholded type of smoothing regularization is actually more desirable in many practical image processing applications because it only eliminates small structures with high curvature while leaving coarse scale structures intact.

Finally, we can obtain our overall algorithm by combining the curve evolution driven by the two speeds $\hat{F}_d$ and $\hat{F}_{\text{int}}$. Our overall fast two-cycle (FTC) algorithm is given in Table II. In the first cycle, we evolve the curve with $\hat{F}_d$ for N_a iterations, and then in the second cycle, we introduce smoothness regularization by evolving the curve according to $\hat{F}_{\text{int}}$ for N_s iterations. The strength of smoothness regularization is controlled by the Gaussian filter G and the number of iterations N_a and N_s for each cycle. The relation between N_a and N_s is easy to understand. The bigger N_s is relative to N_a, the stronger the regularization. Overall, since each cycle of the two-cycle algorithm is simply an application of the algorithm in Table I to the speeds $\hat{F}_d$ and $\hat{F}_{\text{int}}$, the two-cycle algorithm maintains the same level of computational efficiency as the algorithm in Table I. In particular, all computations in our two-cycle algorithm are on integers and evolution involves simple list updates. As already pointed out, the price for this efficiency is pixel-level accuracy and approximate curvature-based motion. Even so, excellent agreement can be obtained with PDE-based level-set algorithms for problems of practical interest.

V. Experimental Results

In this section, we present experimental results from image segmentation and real-time video tracking problems to demonstrate the practical utility and performance of our FTC algorithm. For the problem of image segmentation, we first compare the FTC algorithm with the PDE-based narrow band algorithm in ITK [30] in Section V-A. We consider both edge-based and region-based level set-based segmentation approaches to demonstrate the flexibility of our approach. Robustness of the FTC algorithm with respect to perturbations of parameters is then discussed in Section V-B. After that, we demonstrate in Section V-C the application of the FTC algorithm in real-time video tracking. All experiments in this section were run on a 1.6-GHz Intel PC.

A. Image Segmentation

In our image segmentation experiments, we compare the FTC algorithm with the PDE-based narrow band algorithm in ITK [30], described earlier. We have chosen evolution speeds in this section that are implemented in ITK because they are openly available and, thus, can be used as references by other researchers. Even though these speeds are fairly simple in terms of segmentation, they serve well in measuring the computational efficiency of level-set-based curve evolution algorithms. Table III collects the parameters of the FTC algorithm for all the experimental results shown in this section. Quantitatively, we use a popular measure called the *Dice coefficient* [53]-[56] to compare the final segmentation results obtained by the two methods. Given two segmented object regions Ω_1 and Ω_2 from two different algorithms, the Dice coefficient is defined as

$$d_v(\Omega_1, \Omega_2)=\frac{2Area(\Omega_1 \cap \Omega_2)}{Area(\Omega_1)+Area(\Omega_2)}. \tag{9}$$

The Dice coefficient varies from 0 to 1 and it measures the degree of agreement between the two segmented regions. It is 1 when the two regions are identical and 0 when they are completely different.

First, we consider an edge-based segmentation problem based on the geodesic active contour model. The curve evolution equation of the geodesic active contour model is

$$\frac{dC}{dt}=g(f)(c+\kappa)\overrightarrow{N} - (\nabla g \cdot \overrightarrow{N})\overrightarrow{N} \tag{10}$$

where f is the image to be segmented, g is an edge potential function, and c is a balloon force [5]. To approximate the geodesic model using our FTC approach, we define $F_d = g(f)c - (\nabla g \cdot \overrightarrow{N})$ in our two-cycle algorithm. We use the second, smoothing cycle of our algorithm to capture the remaining, curvature-based term $\kappa\overrightarrow{N}$. Specific parameters are given in Table III. The same edge potential function as described in the manual of ITK [30] is used for both our algorithm and ITK.

In Fig. 4, we present the segmentation results of a plane image with our algorithm and ITK. We can see that the final segmentation results with our FTC method and the ITK method show excellent qualitative agreement. The numerical measure d_v in Table IV confirms this. With our FTC method, a speedup factor of approximately 100 is obtained, as shown in Table IV. For a more detailed analysis of the various factors contributing to the efficiency of the FTC algorithm, we compute at each iteration of the FTC algorithm and ITK the Dice coefficient between the current segmentation and the final segmentation result of the ITK algorithm in Fig. 4(c). As shown in Fig. 5, the FTC algorithm converges in less than 200 iterations, while ITK needs almost 800 iterations to reach the final segmentation. This demonstrates our algorithm is able to achieve faster convergence than the PDE-based method in ITK. Because overall a speedup factor of 106 is achieved, it also shows in this case the computational cost per iteration in the FTC algorithm is around 25 times lower than ITK.

Next, we consider a region-based segmentation problem. We use an intensity-threshold-based data term for simplicity. The curve evolution equation for the threshold-based speed is as follows:

$$\frac{dC}{dt}=F_d\overrightarrow{N}+\lambda\kappa\overrightarrow{N} \tag{11}$$

where is F_d the data-dependent speed defined as

$$F_d=\begin{cases}+1, & \text{if } f\in[I_1,I_2]\\ -1, & \text{otherwise.}\end{cases} \tag{12}$$

Here, f denotes the image to be segmented, and $[I_1, I_2]$ is the range of intensities for the region to be segmented. To approximate this evolution using our FTC approach we use F_d in (12) in the first cycle and use the second smoothing cycle to capture the curvature-based component of the speed $\kappa\overrightarrow{N}$.

The image to be segmented is a simulated 3-D MRI brain image from the BrainWeb [57] as shown in Fig. 6(a). The size of the image is 181 × 217 × 181. In Fig. 6(b) and (c), we show the segmentation results from our FTC algorithm and ITK. The intensity range used is $[I_1, I_2]$ = [120, 160] and the regularization parameter for ITK is $\lambda = 0.75$. Parameters for our FTC method are given in Table III.

In this example, we again obtain excellent qualitative agreement between the results obtained with our FTC method and the PDE-based ITK method. The numerical measure d_v in Table IV confirms this agreement. A speedup of around two orders of magnitude is also obtained as illustrated in Table IV.

B. Robustness of the FTC Algorithm

In this section, we demonstrate the robustness of the FTC algorithm with respect to regularization parameters. Overall, there are four parameters controlling the result: N_g, σ, N_a, and N_s. The parameters N_g and σ decide the approximated Gaussian filter G and control how much smoothness regularization is imposed in each iteration. The parameters N_a and N_s control the relative weight between evolution using the data-dependent speed $\hat{F_d}$ and the smoothing speed $\hat{F}_{\text{int}}$. In the following experiments, we test the sensitivity of the FTC algorithm by applying different sets of parameters to the segmentation of the plane image in Fig. 4.

In our experiments, we first perturb the parameter N_a while fixing the other three parameters N_s, N_g, and σ. Three different values of N_a are used as listed in the first three rows of Table V. For each set of parameters, the time to perform the segmentation and the Dice coefficient between the FTC result and the ITK segmentation shown in Fig. 4(c) are listed in Table V. The results show that less time is used with the increase of N_a as a reduced number of smoothing iterations are applied. We can also see from the Dice coefficients that the segmentation results do not have significant changes and are very robust to perturbations of N_a. Similar to the results in Fig. 5, we also compute the Dice coefficient at each iteration with respect to the segmentation result of ITK. The maximal difference of the Dice coefficient at each iteration is less than 0.001 for the three different values of N_a used, which shows that the convergence process of the FTC algorithm is also fairly robust to perturbations of N_a.

We next fix the parameter N_a as in Table III and apply the FTC algorithm with three sets of parameters for N_s, N_g and σ with increasing smoothness regularization as listed in the last three rows of Table V. Not surprisingly, with the increase of N_s, N_g and σ, the computational cost in evaluating the smoothing speed $\hat{F}_{\text{int}}$ will increase and this affects the overall computation time of the FTC algorithm as listed in Table V. As more smoothness regularization is incorporated, more details are removed from the final segmentation and this is reflected in the corresponding Dice coefficients in Table V.

In summary, the above experiments demonstrate the impact of regularization parameters on the performance of the FTC algorithm in terms of its execution time and segmentation result. These results suggest that the FTC algorithm is quite stable with respect to the perturbation of parameters.

C. Real-Time Video Tracking

The application of the level-set method for video tracking has attracted significant interest and much work has been done in this direction, such as [7] and [12]–[17], but it is difficult to achieve real-time video-based performance with PDE-based approaches. In this section, we will demonstrate how our FTC method can perform such tasks by implementing a region-based video tracking system.

For simplicity, we assume there is an object region Ω_{obj} and a background region Ω_{bg} in the scene, but extension to the tracking of multiple objects is straightforward [58]. The data-dependent speed F_d used in our FTC tracking system is based on the region competition model [59] and is defined as follows:

$$F_d = \log \frac{p(\underline{v}\,\Omega_{\text{obj}})}{p(\underline{v}\,\Omega_{\text{bg}})} \tag{13}$$

where $\underline{v}$ is a feature vector used for tracking (for example the intensity difference with respect to a reference frame or the color vector [60]–[63]), and $p(\underline{v}|\Omega_{\text{obj}})$and $p(\underline{v}|\Omega_{\text{bg}})$ are the probability distribution of this feature vector for the object and background region. Since the purpose here is to demonstrate real-time performance for level-set-based video tracking, we use a simple tracking strategy with the tracking results from the previous frame providing the initial curve for the current frame. The parameters of our FTC algorithm used in the tracking examples below are listed in Table VI. Again, our aim here is not to design or demonstrate the best tracking strategy, but to demonstrate that level-set-based tracking strategies of engineering interest can be efficiently implemented with our FTC method.

First, we will perform some offline analysis to demonstrate how the computational requirements of typical tracking problems relate to the constraints of typical video sensors. To start, in Fig. 7, we show tracking results for the *Hall monitor* test sequence. The underlying video is in CIF format. The absolute intensity difference with respect to a reference frame is used as the feature and a Gaussian distribution is assumed for the feature of both the object and background region. For the tracking of the object region, the mean and standard deviation used in the Gaussian distribution are 50 and 25, and for the background region, the Gaussian distribution has mean 6 and standard deviation 3. The person is successfully tracked from frame

41 to 65. The tracking time for each frame may vary due to the difference of interframe motions. For this sequence, the maximum time used by the FTC algorithm for a frame is 0.0049 s. Overall, the average time needed by the FTC algorithm to perform tracking is 0.0044s/frame, which corresponds to a tracking rate of 227 frames/s.

For a second offline example, we show the tracking results of 65 frames from the SIF format *Tennis* test sequence. The feature vector used here consists of the Y and V components of the YUV color at each pixel. A 32 × 32 histogram is estimated at initialization to approximate the feature distribution used in (13). This is done for both the object and background region separately. As shown in Fig. 8, the player is successfully tracked as he moves into the scene. Note that the background also changes over time. For this sequence, the maximum time used by the FTC algorithm for the tracking of a frame is 0.0048 s and the average tracking time is 0.0022 s/frame, corresponding to a tracking rate of 464 frames/s.

From the offline examples shown above, we can see that our FTC algorithm is able to track objects in video sequences at a rate much higher than real-time on a regular PC. As we have shown, PDE-based level-set algorithms are typically two orders of magnitude slower than our FTC algorithm, making them unsuitable for real-time video use. For practical tracking systems, the FTC algorithm might be even more desirable since only a small faction of CPU time can be allocated for tracking. As a demonstration, we have implemented a real-time tracking system using Matlab on our PC. We use the image acquisition toolbox in Matlab to capture the current frame of video in CIF format from an inexpensive USB-based camera, then this data is sent to our tracking algorithm implemented in C++. Once the tracking result is available, it is sent back to Matlab for display along with the video frame. Then the next frame from the video camera is captured to continue the tracking process. In our nonoptimized implementation most of the time is spent in getting frames in and out of Matlab, yet even here the system runs comfortably at more than 20 frames per second. In Fig. 9, we show an example of the tracking results from our system where two moving hands are tracked based on color. Various topological changes are handled automatically during the tracking process, demonstrating the advantages of the level-set framework on which our FTC algorithm is based.

VI. Conclusion

In this paper, we have proposed a fast two-cycle algorithm for the approximation of level-set-based curve evolution. The computational core of our algorithm is a novel element switching mechanism between two linked lists that realizes implicit curve evolution using only integer operations. This results in computational efficiency, yet excellent agreement with PDE-based implementations. By design, the approach is applicable to a very broad class of speeds, including speeds derived from both region-based and edge-based models, that can be viewed as composed of a data-dependent term and a smoothness regularization term. Considerable speedups have been demonstrated as compared to PDE-based narrow band level-set implementations. With our algorithm, real-time level-set-based video tracking can be realized.

Acknowledgments

This work was supported in part by the Engineering Research Centers Program of the National Science Foundation under award number EEC-9986821, in part by the National Institutes of Health under Grant NINDS 1 R01 NS34189, and in part by the Air Force under Award number FA9550-07-1-0295. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Peyman Milanfar.

References

1. Osher S, Sethian J. Fronts propagation with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations. J Comput Phys 1988;79:12–49.

2. Sethian, J. Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science. Cambridge, U.K: Cambridge Univ. Press; 1999.
3. Osher, S.; Fedkiw, R. Level Set Methods and Dynamic Implicit Surfaces. New York: Springer Verlag; 2002.
4. Malladi R, Sethian J, Vemuri B. Shape modeling with front propagation: A level set approach. IEEE Trans Pattern Anal Mach Intell Feb;1995 17(2):158–175.
5. Caselles V, Kimmel R, Sapiro G. Geodesic active contours. Int J Comput Vis 1997;22(1):61–79.
6. Kichenassamy, S.; Kumar, A.; Olver, P.; Tannenbaum, A.; Yezzi, A. Gradient flows and geometric active contour models. Proc. ICCV; Boston, MA. 1995. p. 810-815.
7. Bertalmio M, Sapiro G, Randall G. Morphing active contours: A geometric approach to topology-independent image segmentation and tracking. Proc Int Conf Image Processing 1998;III:318–322.
8. Chan T, Vese L. Active contours without edges. IEEE Trans Image Process Feb;2001 10(2):266–277. [PubMed: 18249617]
9. Tsai A, Yezzi A, Willsky A. Curve evolution implementation of the Mumford–Shah functional for image segmentation, denoising, interpolation, and magnification. IEEE Trans Image Process Aug; 2001 10(8):1169–1186. [PubMed: 18255534]
10. Feng H, Karl W, Castañòn D. A curve evolution approach to object-based tomographic reconstruction. IEEE Trans Image Process Jan;2003 12(1):44–57. [PubMed: 18237878]
11. Zeng X, Staib L, Schultz R, Duncan J. Segmentation and measurement of the cortex from 3D MR images using coupled surfaces propagation. IEEE Trans Med Imag Oct;1999 18(10):927–937.
12. Paragios N, Deriche R. Geodesic active contours and level sets for the detection and tracking of moving objects. IEEE Trans Pattern Anal Mach Intell Mar;2000 22(3):266–280.
13. Besson S, Barlaud M, Aubert G. Detection and tracking of moving objects using a new level set based method. Proc ICPR Sep;2000 3:1100–1105.
14. Mansouri A. Region tracking via level set PDEs without motion computation. IEEE Trans Pattern Anal Mach Intell Jul;2002 24(7):947–961.
15. Freedman D, Zhang T. Acitve contours for tracking distributions. IEEE Trans Image Process Apr; 2004 13(4):518–526. [PubMed: 15376586]
16. Mukherjee D, Ray N, Acton S. Level set analysis for leukocyte detection and tracking. IEEE Trans Image Process Apr;2004 13(4):562–572. [PubMed: 15376590]
17. Yilmaz A, Li X, Shah M. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. IEEE Trans Pattern Anal Mach Intell Nov;2004 26(11):1531–1536. [PubMed: 15521500]
18. Chopp D. Computing minimal surfaces via level set curvature flow. J Comput Phys 1993;106:77–91.
19. Tsitsiklis JN. Efficient algorithms for globally optimal trajectories. IEEE Trans Autom Contr Sep; 1995 40(9):1528–1538.
20. Sethian J. A fast marching level set method for monotonically advancing fronts. Proc Nat Acad Sci 1996;93(4):1591–1595. [PubMed: 11607632]
21. Adalsteinsson D, Sethian J. The fast construction of extension velocities in level set methods. J Comput Phys 1999;148:2–22.
22. Gibou, F.; Fedkiw, R. A fast hybrid k-means level set algorithm for segmentation. Proc. 4th Annu. Hawaii Int. Conf. Statistics and Mathematics; Jan. 2005; p. 281-291.
23. Song B, Chan T. A fast algorithm for level set based optimization. UCLA CAM Rep. 02-68. 2002
24. Esedoglu S, Tsai Y. Threshold dynamics for the piecewise constant Mumford–Shah functional. UCLA CAM Rep. 04-63. 2004
25. Tai XC, Christiansen O, Lin P. A remark on the MBO scheme and some piecewise constant level set methods. UCLA CAM Rep. 05-24. 2005
26. Lie J, Lysaker M, Tai X. A binary level set model and some applications to Mumford–Shah image segmentation. IEEE Trans Image Process May;2006 15(5):1171–1181. [PubMed: 16671298]
27. Peng D, Merriman B, Osher S, Zhao H, Kang M. A PDE-based fast local level set method. J Comput Phys 1999;155:410–438.

28. Whitaker R. A level-set approach to 3D reconstruction from range data. Int J Comput Vis Oct;1998 29(3):203–231.
29. Lefohn A, Kniss J, Hansen C, Whitaker R. A streaming narrow-band algorithm: Interactive computation and visualization of level sets. IEEE Trans Vis Comput Graph Jul./Aug;2004 10(4): 422–433. [PubMed: 18579970]
30. The Insight Toolkit [Online]. Available: http://www.itk.org
31. Goldenberg R, Kimmel R, Rivlin E, Rudzsky M. Fast geodesic active contours. IEEE Trans Image Process Oct;2001 10(10):1467–1475. [PubMed: 18255491]
32. Lu T, Neittaanmaki P, Tai XC. A parallel splitting up method and its application to Navier–Stokes equations. Appl Math Lett 1991;4(2):25–29.
33. Lu T, Neittaanmaki P, Tai XC. A parallel splitting up method for partial differential equations and its application to Navier–Stokes equations. RAIRO Math Model and Numer Anal 1992;26(6):673–708.
34. Weickert J, Romeny B, Viergever M. Efficient and reliable scheme for nonlinear diffusion filtering. IEEE Trans Image Process Mar;1998 7(3):398–410. [PubMed: 18276260]
35. Merriman, B.; Bence, J.; Osher, S.; Taylor, J., editors. Diffusion generated motion by mean curvature; Proc. Computational Crystal Growers Workshop; Providence, RI. 1992. p. 73-83.
36. Merriman B, Bence J, Osher S. Motion of multiple junctions: A level set approach. J Comput Phys 1994;112(2):334–363.
37. Jawerth B, Lin P. Shape recovery by diffusion generated motion. J Vis Commun Image Represent May/Jun;2002 13(1/2)
38. Isard M, Blake A. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. Proc ECCV 1998:767–781.
39. Chen Y, Rui Y, Huang TS. JPDAF based HMM for real-time contour tracking. Proc CVPR 2001;1:543–550.
40. Comaniciu D, Ramesh V, Meer P. Kernel-based object tracking. IEEE Trans Pattern Anal Mach Intell May;2003 25(5):564–577.
41. Precioso, F.; Barlaud, M.; Blu, T.; Unser, M. Smoothing B-spline active contour for fast and robust image and video segmentation. presented at the Int. Conf. Image Processing; Sep 2003;
42. Boykov, Y.; Jolly, MP. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. Proc. ICCV; 2001. p. 105-112.
43. Boykov Y, Kolmogorov V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans Pattern Anal Mach Intell Sep;2004 26(9):1124–1137. [PubMed: 15742889]
44. Kong T, Rosenfeld A. Digital topology: Introduction and survey. Comput Vis Graph Image Process Dec;1989 48(3):357–393.
45. Thirion J, Gourdon A. Computing the differential characteristics of isointensity surfaces. Comput Vis Image Understand 1995;61(2):190–202.
46. Monga O, Benayoun S. Using partial dirivatives of 3D images to extract typical surface features. Comput Vis Image Understand 1995;61(2):171–189.
47. Rieger B, Timmermans F, van Vliet LJ, Verbeek P. On curvature estimation of iso surfaces in 3D gray-value images and the computation of shape descritors. IEEE Trans Pattern Anal Mach Intell Aug;2004 26(8):1088–1094. [PubMed: 15641738]
48. Mascarenhas P. Diffusion generated motion by mean curvature. UCLA CAM Rep. 92-33. 1992
49. Evans L. Convergence of an algorithm for mean curvature motion. Indiana Univ Math J 1993;42:553–557.
50. Barles G, Georgelin C. A simple proof of convergence for an approximation scheme for computing motions by mean curvature. SIAM J Numer Anal 1995;32(2):484–500.
51. Ishii H, Pires G, Souganidis P. Threshold dynamics type approximation schemes for propagating fronts. J Math Soc Jpn 1999;51:267–308.
52. Ishii, H.; Damlamian, A.; Spruck, J.; Visintin, A., editors. A generalization of the Bence, Merriman and Osher algorithm for motion by mean curvature; Proc. Curvature Flows and Related Topics; Tokyo, Japan. 1995. p. 111-127.

53. Dice L. Measures of the amount of ecologic association between species. Ecology 1945;26:297–302.
54. Zijdenbos AP, Dawant BM, Margolin RA, Palmer AC. Morphometric analysis of white matter lesions in MR images. IEEE Trans Med Imag Dec;1994 13(4):716–724.
55. Shattuck DW, SandorLeahy SR, Schaper KA, Rottenberg DA, Leahy RM. Magnetic response image tissue classification using a partial volume model. Neuro Image 2001;13:858–876.
56. Boesen K, Rehm K, Schaper K, Stoltzner S, Woods R, Luders E, Rottenberg D. Quantitative comparison of four brain extraction algorithms. Neuro Image 2004;22:1255–1261. [PubMed: 15219597]
57. Brainweb: Simulated Brain Database [Online]. Available: http://www.bic.mni.mcgill.ca/brainweb
58. Shi Y, Karl W. Real-time tracking using level sets. Proc CVPR Jun;2005 2:34–41.
59. Zhu S, Yuille A. Region competition: Unifying snake/balloon, region growing, and Bayes/MDL/energy for multi-band image segmentation. IEEE Trans Pattern Anal Mach Intell Sep;1996 18(9):884–900.
60. Oliver N, Pentland A, Berarad F. Lafter: Lips and face real time tracker. Proc CVPR 1997:123–129.
61. Yang, J.; Weier, L.; Waibel, A. Skin-color modeling and adaptation. Proc. Asian Conf. Computer Vision; 1998. p. 687-694.
62. Raja, Y.; McKenna, S.; Gong, S. Tracking and segmenting people in varying lighting conditions using colour. Proc. Int. Conf. Automatic Face and Gesture Recognition; 1998. p. 228-233.
63. Sigal L, Sclaroff S, Athitsos V. Skin color-based vidoe segmentation under time varying illumination. IEEE Trans Pattern Anal Mach Intell Jul;2004 26(7):862–877. [PubMed: 18579945]

Appendix

The proof of Theorem 1 is as follows.

Proof

At initialization, let $\Omega^a = \Omega^* \cap \Omega = \cup_{p=1}^{P} \Omega_p^a$ with $\Omega_p^a = \Omega_p^* \cap \Omega$. These sets are fixed once defined and will not change as Ω evolves over time. Let L_{in}^a be the inside neighboring grid point list of Ω^a. Because at initialization $L_{\text{in}} \subseteq (\Omega_{\text{in}}^* \cup \Omega_{\text{out}}^*)$, we have $L_{\text{in}}^a \subseteq \Omega_{\text{in}}^*$. Since $\pi(\Omega, \Omega^*) \subseteq \Omega_{\text{in}}^* \cup \Omega_{\text{out}}^*$, we also have $\Omega^* \cap (D \backslash \Omega^a) \subseteq \Omega_{\text{in}}^*$. During the evolution process, no *switch_out* operation can be applied to $\forall\, \mathbf{x} \in L_{\text{in}}^a$ because its speed $\hat{F}(\mathbf{x}_k) > 0$, which ensures that $\Omega^a \subseteq \Omega$ as Ω is updated with our alogrithm and, thus, $\Omega^* \cap (D \backslash \Omega) \subseteq \Omega^* \cap (D \backslash \Omega^a) \subseteq \Omega_{\text{in}}^*$.

Assume $\Omega^* \cap (D \backslash \Omega) \neq \varnothing$ at initialization, then $\exists \mathbf{x} \in \Omega_p^* \cap (D \backslash \Omega)$. Because Ω_p^* is a connected region and $\Omega_p^a \neq \varnothing$, $\exists \mathbf{y} \in \Omega_p^a$ that is connected to $\mathbf{x}$, so we can find a sequence of points $\mathbf{x}_1 = \mathbf{x}, \mathbf{x}_2, \ldots, \mathbf{x}_K = \mathbf{y}$ in Ω_p^* such that $\mathbf{x}_k \in N(\mathbf{x}_{k-1})$. Since $\mathbf{x}_1 \in (D \backslash \Omega)$ and $\mathbf{x}_K \in \Omega$, there must exist a point $\mathbf{x}_k \in L_{\text{out}}$ in this sequence. Since its speed $\hat{F}(\mathbf{x}_k) > 0$, a *switch_in* operation will be applied to it, so $\mathbf{x}_k$ will become inside Ω after one iteration. This shows the set $\Omega^* \cap (D \backslash \Omega)$will lose at least one point at each iteration if the set is nonempty. Because this is a finite set, it will converge to an empty set after a finite number of iterations.

Following similar arguments, we can also show $\Omega \cap (D \backslash \Omega)$will converge to an empty set after a finite number of steps. This completes the proof of Theorem 1.

Biographies

Yonggang Shi (M'01) received the B.S. and M.S. degrees in electrical engineering from Southeast University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree in electrical engineering from Boston University, Boston, MA, in 2005.

He is currently a Postdoctorate Fellow at the Laboratory of Neuro Imaging (LONI), School of Medicine, University of California, Los Angeles. His research interests include level-set and PDE-based methods for image processing, 3-D shape analysis with applications in brain mapping, and statistical signal processing.

Dr. Shi was a winner of the student paper contest at ICASSP 2005, Philadelphia, PA.

William Clem Karl (M'91–SM'00) received the S.M., E.E., S.B., and Ph.D. degrees degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge.

He held the position of Staff Research Scientist with the Brown-Harvard-MIT Center for Intelligent Control Systems and the MIT Laboratory for Information and Decision Systems from 1992 to 1994. He joined the faculty of Boston University (BU), Boston, MA, in 1995, where he is currently a Professor in the Electrical and Computer Engineering Department. Since January 1996, he has also held a joint appointment in the Department of Biomedical

Engineering, BU. His research interests are in the areas of multidimensional signal and image processing, geometric estimation, detection, and medical signal and image processing.

Dr. Karl has been an Associate Editor of the IEEE Transactions on Image Processing. He has also served in various organizational capacities, including as Session Organizer and Chair for the Asilomar Conference on Signals, Systems, and Computers, Special Session on Inverse Problems in Imaging; Session Organizer and Chair for the Conference in Information Sciences and Systems, Special Session on Medical Imaging; and as part of the organizing committee for the First SIAM Conference on the Life Sciences. He will serve as the General Chair of the 2009 IEEE International Symposium on Biomedical Imaging.

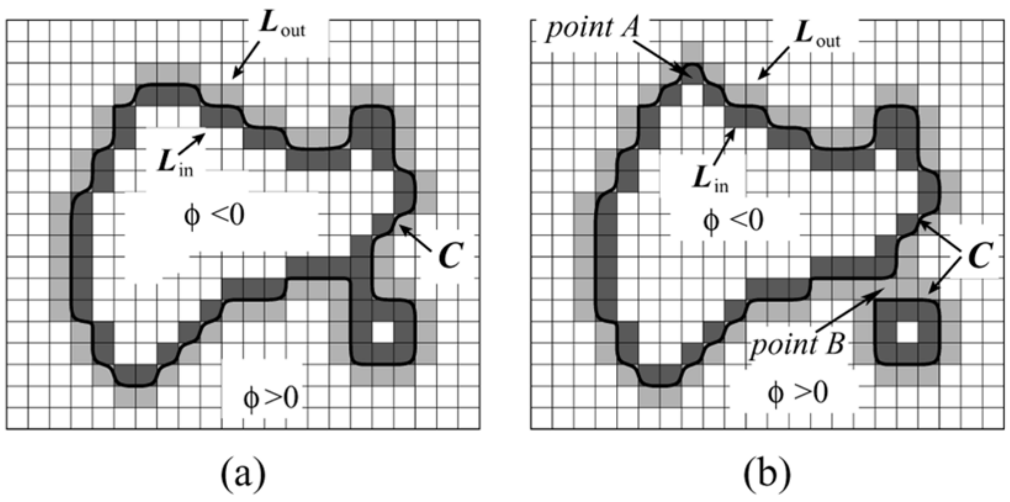

Fig. 1.
(a) Implicit representation of a curve C in the level-set method. Two lists of neighboring grid points L_{in} and L_{out} can be defined uniquely on this grid. (b) Motion of the implicitly represented curve can be achieved by switching points between L_{in} and L_{out}.

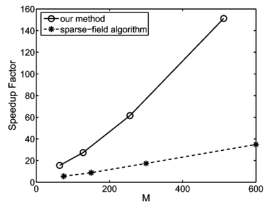

Fig. 2.
Comparison of the speedup factor to the global level-set method between our algorithm in Table I and the sparse-field algorithm versus problem size.

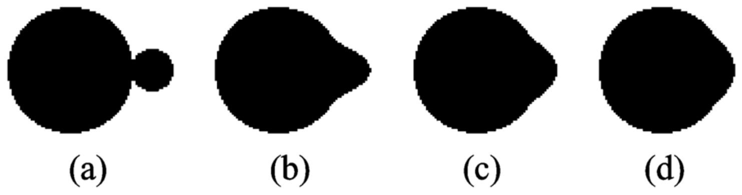

Fig. 3.
Evolution process of a shape with the smoothing cycle of our two-cycle algorithm. The Gaussian filter is of size 9×9 with $\sigma = 3$. (a) Initial object region; (b)–(d) shows the object region at iteration 18, 36, and 54.

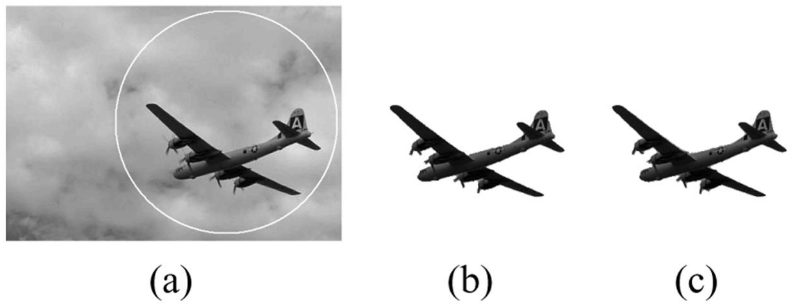

Fig. 4.

Edge-based segmentation of a plane image. (a) Initial curve as a white circle plotted over the original image. (b) Segmentation results from our FTC algorithm. (c) Segmentation results from ITK.

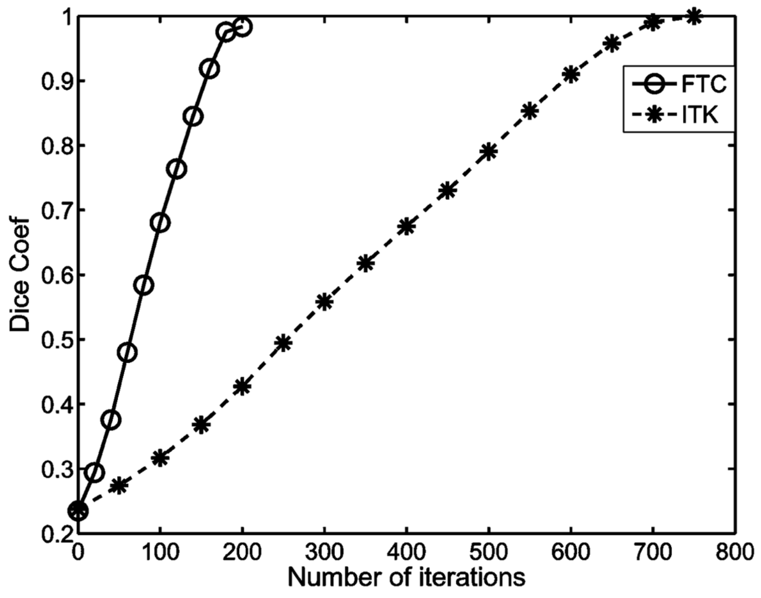

Fig. 5.
Comparison of the convergence speed between the FTC algorithm and ITK.

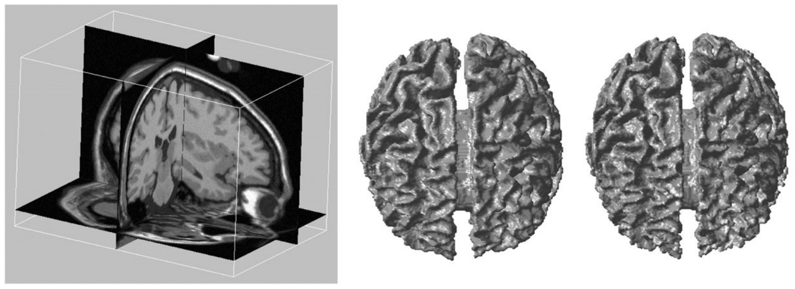

Fig. 6.
Segmentation of a 3-D MRI brain image. (a) Snapshot of the 3-D image on three orthogonal slices. (b) Segmented surface from our FTC algorithm. (c) Segmented surface from ITK.

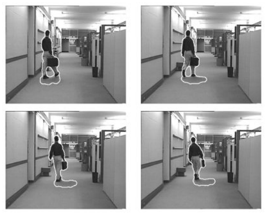

Fig. 7.

Hall sequence tracking results. Frame 41, 49, 57, and 65 are shown.

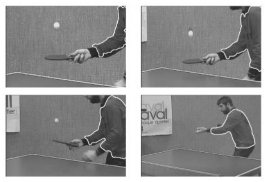

Fig. 8.
Tennis sequence tracking results. Frame 1, 38, 45, and 65 are shown.

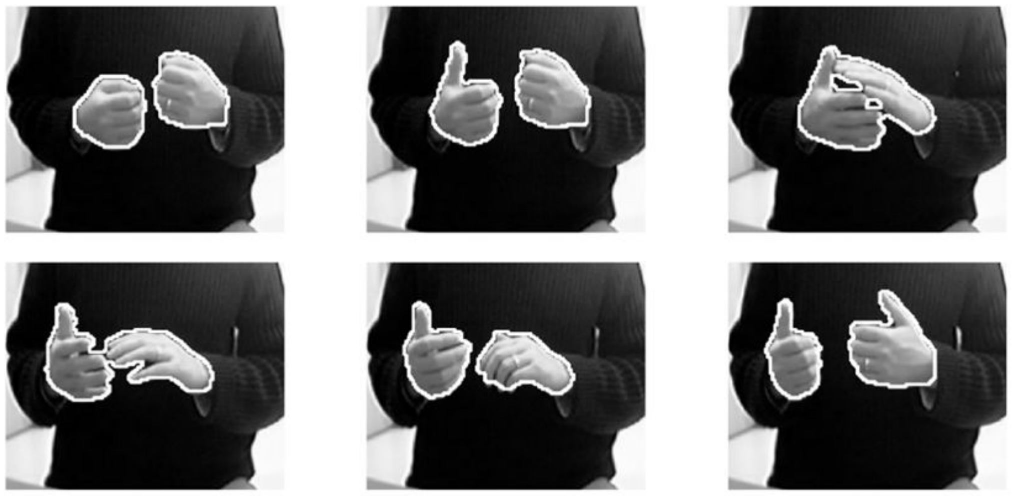

Fig. 9.
Hand tracking results. Frames 1, 40, 81, 130, 150, and 200 of a sequence are shown.

TABLE I

Basic Fast Algorithm for Geometry Independent Speeds

- Step 1: Initialize arrays $\hat{\varphi}$ and $\hat{F}$, and the lists L_{out} and $L_{in.}$
- Step 2: Compute speed F for all points in L_{out} and L_{in} and store sign in $\hat{F}$.
- Step 3: Scan through the two lists L_{out} and L_{in} and update:
  - **Outward evolution.** Scan through L_{out}. For each point $\mathbf{x} \in L_{out}$, *switch_in*($\mathbf{x}$) if $\hat{F}(\mathbf{x}) > 0$.
  - **Eliminate redundant points in L$_{in}$.** Scan through L_{in}. For each point $\mathbf{x} \in L_{in}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\hat{\varphi}(\mathbf{x}) = -3$.
  - **Inward evolution.** Scan through L_{in}. For each point $\mathbf{x} \in L_{in}$, *switch_out*($\mathbf{x}$) if $\hat{F}(\mathbf{x}) < 0$.
  - **Eliminate redundant points in L$_{out}$.** Scan through L_{out}. For each point $\mathbf{x} \in L_{out}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\hat{\varphi}(\mathbf{x}) = 3$.
- Step 4: If the stopping condition is not satisfied, go to Step 2.

TABLE II

Fast Two-Cycle (FTC) Algorithm

- Step 1: Initialize arrays $\hat{\varphi}$, $\hat{F}_d$, $\hat{F}_{int}$, and lists L_{out} and L_{in}.
- Step 2 (**cycle one: data dependent evolution**):

  For i=1:N_a do

  - Compute the data dependent speed F_d for each point in L_{out} and L_{in} and store its sign in $\hat{F}_d$.
  - **Outward evolution.** For each point $\mathbf{x} \in L_{\text{out}}$, *switch_in*($\mathbf{x}$) if $\hat{F}_d(\mathbf{x}) > 0$.
  - **Eliminate redundant points in L**$_{\text{in}}$. For each point $\mathbf{x} \in L_{\text{in}}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\hat{\varphi}(\mathbf{x}) = -3$.
  - **Inward evolution.** For each point $\mathbf{x} \in L_{\text{in}}$, *switch_ out*($\mathbf{x}$) if $\hat{F}_d(\mathbf{x}) < 0$.
  - **Eliminate redundant points in L**$_{\text{out}}$. For each point $\mathbf{x} \in L_{\text{out}}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\hat{\varphi}(\mathbf{x}) = 3$.
  - Check the stopping condition. If it is satisfied, go to Step 3; otherwise continue this cycle.
- Step 3 (**cycle two: smoothing via Gaussian filtering**):

  For i=1:N_s do

  - Compute the smoothing speed $\hat{F}_{int}$ for each point in L_{out} and L_{in}.
  - **Outward evolution.** For each point $\mathbf{x} \in L_{\text{out}}$, *switch_in*($\mathbf{x}$) if $\hat{F}_{\text{int}}(\mathbf{x}) > 0$.
  - **Eliminate redundant points in L**$_{\text{in}}$. For each point $\mathbf{x} \in L_{\text{in}}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\hat{\varphi}(\mathbf{x}) = -3$.
  - **Inward evolution.** For each point $\mathbf{x} \in L_{\text{in}}$, *switch_out*($\mathbf{x}$) if $\hat{F}_{int}(\mathbf{x}) < 0$.
  - **Eliminate redundant points in L**$_{\text{out}}$. For each point $\mathbf{x} \in L_{\text{out}}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\hat{\varphi}(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\hat{\varphi}(\mathbf{x}) = 3$.
- Step 4: If stopping condition not satisfied in cycle one, go to Step 2.

TABLE III

Parameters of Our FTC Algorithm Used in Image Segmentation

Image	Na	Ns	Ng	σ (pixels)	max iter
Plane	30	3	3	1	200
MRI brain	30	3	3	1	200

TABLE IV

Comparison Of Segmentation Results

Image	FTC Time	ITK Time	SF	d_v
Plane	0.0249s	2.629s	106	0.983
MRI brain	4.407s	723.641s	164	0.956

SF: speed up factor of our FTC algorithm with respect to ITK.

TABLE V

Impact of Parameters in the FTC Algorithm on the Segmentation of the Plane Image

N_a	N_s	N_g	σ	Time	d_v
20	3	3	1.0	0.0275s	0.983
30	3	3	1.0	0.0249s	0.983
40	3	3	1.0	0.0234s	0.983
30	6	3	1.0	0.0299s	0.983
30	5	5	2.0	0.0351s	0.968
30	10	5	2.0	0.0515s	0.956

TABLE VI

Parameters of Our FTC Algorithm Used in Video Tracking

Video	**Na**	**Ns**	**Ng**	σ	**max iter**
Hall monitor	12	3	7	2	20
Tennis	12	3	5	2	40
Hands	10	5	5	2	50