A FAST IMPLEMENTATION OF THE LEVEL SET METHOD WITHOUT SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Yonggang Shi, William Clem Karl

January, 2005

Boston University

Department of Electrical and Computer Engineering

Technical Report No. ECE-2005-02

BOSTON
UNIVERSITY

A FAST IMPLEMENTATION OF THE LEVEL SET METHOD WITHOUT SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Yonggang Shi, William Clem Karl

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

January, 2005

Technical Report No. ECE-2005-02

This work was partially supported by the Engineering Research Centers Program of the National Science Foundation under award number EEC-9986821, National Institutes of Health under Grant NINDS 1 R01 NS34189, Air Force under Award number F49620-03-1-0257.

Summary

The level set method is popular in the numerical implementation of curve evolution, but its high computational cost is a bottleneck for real-time applications. In this report, we propose a novel and fast level set implementation without the need of solving partial differential equations (PDEs). In our implementation, only simple operations such as insertion and deletion on two grid point lists are needed to evolve the curve. Two fast algorithms are developed: one for general evolution speeds and the other for a special, yet practically important, class of speed that is composed of a data dependent external speed and an internal speed for smoothness regularization. Compared with previous optimized narrow band algorithms, our first algorithm can achieve an order of magnitude speedup, and the second algorithm can achieve two orders of magnitude speedup in both 2D and 3D image segmentation. In our experiments, we also demonstrate a real-time video tracking system by combining our algorithm with the image acquisition toolbox of Matlab.

Contents

List of Figures

List of Tables

1 Introduction

The localization of object boundaries is an important and challenging task in many imaging problems, such as segmentation and tracking. In recent years there has been intensive interest in the use of the level set method [20, 21, 27] for boundary localization [4–8, 12, 15–17, 22, 29, 34]. The level set method is attractive for its ability to handle topological changes automatically. Its numerical implementation is also straightforward for any dimension. While the level set method has many advantages, its implementation, based on the solution of certain partial differential equations (PDEs), results in a significant computational burden, which limits its use in real-time applications. In this paper, we propose a novel and fast implementation of the level set method which avoids the solution of PDEs, reducing the computation time significantly and allowing real-time level-set-based curve evolution.

Considerable research has been done with the aim of reducing the computational cost of the level set method. When only the zero level set is of interest, narrow band techniques have been proposed to accelerate the evolution process. In [3], a tube is constructed in the neighborhood of the zero level set and the level set function is initialized as a signed distance function within this tube. The evolution PDE is solved only within this narrow band. When the zero level set becomes too close to the edge of the tube, both the tube and the level set function have to be reinitialized using the fast marching method [11, 30]. Improvements to the above narrow band algorithm were proposed in [23] and [32]. Both algorithms are similar in that they reinitialize the level set function to be a signed distance function at every iteration and the tube is only constructed once at the beginning and updated dynamically thereafter. In [23], the reinitialization of the level set function is achieved by solving a Hamilton-Jacobi PDE for a fixed number of steps in every iteration of evolving the level set function. For the sparse-field algorithm in [32], the reinitialization is achieved by computing the distance function approximately. For both methods, a bandwidth of at least five has to be selected for the evaluation of all the gradients. In [32], faster results than the narrow band method in [3] are reported. Recently, a hardware implementation [14] 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 [1]. In [22], the *Hermes* algorithm is proposed to speed up the level set method. 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 [32]. In [9], a fast algorithm for the geodesic active contour model [6, 12] is proposed using the additive operator splitting (AOS) scheme in [31]. 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

quite expensive. It is also worth to point out that real-time results have been reported in [25] for curve evolution based on a parametric representation, but parameterized representations have difficulties in handling complicated topological changes and 3D surface evolution. Our goal is to reduce the computation time of *level-set-based* curve evolution and achieve real-time results.

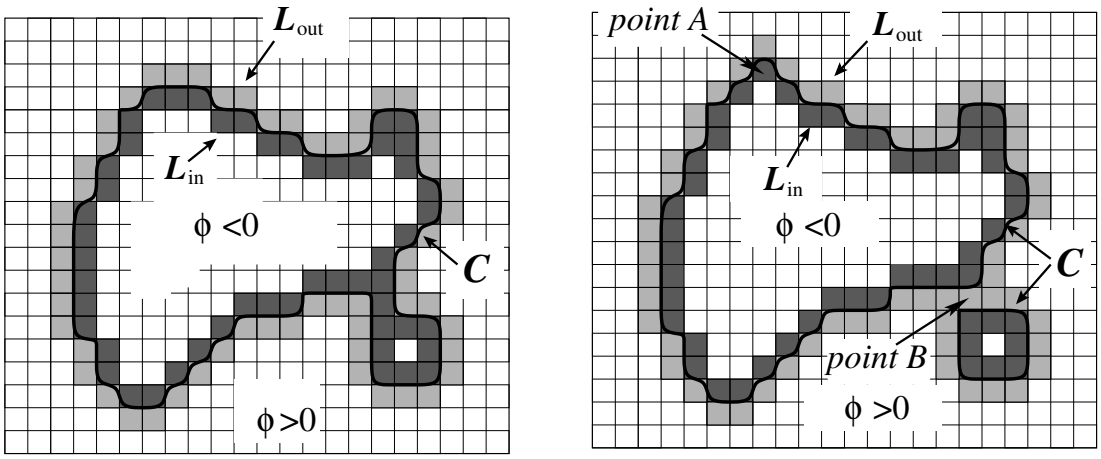
Despite their differences, previous narrow band algorithms all try to track the evolution of the zero level set accurately by solving the associated evolution PDEs locally. However, this accuracy is not necessary for many imaging problems, such as segmentation and tracking, where the goal is to extract the final object boundary. In such cases, the evolution process of the level set function itself is of less interest than the final result. In this paper, we propose a new and fast implementation of the level set method which improves the speed dramatically for this type of problem. Compared with previous approaches, our implementation is much faster for the following reasons. First only simple operations such as insertion and deletion on two point lists are needed to evolve the curve in our implementation, thus no complicated gradient evaluation is involved, as in solving the evolution PDE. In the solution of PDEs, numerical stability concerns usually put stringent step size restrictions on the curve evolution process and frequent reinitialization of the level set function is necessary. While in our implementation, we can move the curve one grid point inward or outward at all locations of the curve in each iteration. Our implementation also completely eliminates reinitialization since we define the value of the level set function from a limited set of integers and it is updated dynamically as the boundary propagates. For image segmentation problems, our method can achieve two orders of magnitude speedup over previous optimized narrow band algorithms.

The rest of the paper is organized as follows. In section II, we analyze the evolution of implicitly represented curves on a discrete grid in the level set method. A key observation between the motion of the curve and two lists of its neighboring grid points is made as the result of this analysis. Based on this observation, we propose a fast implementation for general evolution speeds in section III. In section IV, we propose a more efficient algorithm for a special class of speed function by incorporating smoothness regularization using Gaussian filtering. Experimental results on time critical applications, including 3D segmentation and real-time video tracking, are presented in section V. Finally, conclusions are made in section VI.

2 Analysis of Implicit Curve Evolution

In this section, we analyze the motion of an implicitly represented surface in general $K(K \geq 2)$ dimensional Euclidean space $\mathbf{R}^K$. Based on this analysis, we propose a new strategy for the implementation of the level set method. For convenience, we use the 2D terminology "curve" to denote this surface and the method of evolving the surface as the curve evolution method.

In the level set method, a curve C is represented implicitly as the zero level set of a function ϕ defined over a fixed grid as shown in Fig. 1(a). Here we choose ϕ to be



(a) (b)

Figure 1: (a) The 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) The motion of the implicitly represented curve can be achieved by switching points between L_{in} and L_{out}.

negative inside the curve C and positive outside C. We assume that ϕ is defined over a domain $D \subset \mathbf{R}^K$ and it is discretized into a grid 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. For a point $\mathbf{x}$ in the grid, we denote its coordinate as $\mathbf{x} = (x_1, x_2, \cdots, x_K)$.

Given this implicit representation, we can define two lists of neighboring grid points L_{in} and L_{out} for the curve C as follows:

$$L_{in} = \{\mathbf{x} | \phi(\mathbf{x}) < 0 \ and \ \exists \mathbf{y} \in N(\mathbf{x}) \text{ such that } \phi(\mathbf{y}) > 0\},$$
$$L_{out} = \{\mathbf{x} | \phi(\mathbf{x}) > 0 \ and \ \exists \mathbf{y} \in N(\mathbf{x}) \text{ such that } \phi(\mathbf{y}) < 0\}$$

where $N(\mathbf{x})$ is a discrete neighborhood of $\mathbf{x}$ defined as follows:

$$N(\mathbf{x}) = \{\mathbf{y} \in D | \sum_{k=1}^{K} |y_k - x_k| = 1\} \quad \forall \mathbf{x} \in D. \tag{1}$$

As we can see from Fig. 1(a), L_{in} is the list of neighboring grid points that is inside C and L_{out} is the list of neighboring grid points that is outside C. 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 *unique*. Vice versa, the location of the curve can be determined up to the accuracy of the grid sampling interval if the two lists are given. This motivates us to analyze the relation between the motion of this implicitly represented curve and the two lists L_{in} and L_{out}.

In the classical level set method [21], the following PDE is solved numerically on

the fixed grid to evolve the curve C under a speed field F:

$$\frac{d\phi}{dt} + F\,|\nabla\phi| = 0. \tag{2}$$

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 (2) and difficult to meet real-time requirements. But if we only care about 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 outside the grid point A, we just need to switch it 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. This observation forms the foundation of our fast implementation.

To avoid confusion between our approach and Lagrangian type methods for curve evolution, we note that the points in L_{in} and L_{out} are *not* on C. They are used to define the neighborhood of the curve on the discrete grid and the curve is still represented implicitly by ϕ.

3 Fast Implementation For General Speeds

In this section, we present our fast implementation of the level set method for a general speed field F. A new way of incorporating smoothness regularization will be proposed in the next section to further reduce the computation.

The data structure in our implementation is quite simple and it is listed as follows:

- An array for the level set function ϕ;
- An array for the speed F;
- Two lists of neighboring grid points: L_{in} and L_{out}.

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

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

With this definition, we can easily tell the relative location of a point with respect to the zero level set from the value of ϕ. For the evolution speed, only the sign is used in our algorithm, thus F is also an integer array and equals 1, 0 or -1. The two lists L_{in} and L_{out} are bi-directionally linked such that insertion and deletion can be done easily.

Before we present the details of the algorithm, let us first define two basic procedures on our data structure.

The procedure $switch\_in()$ for a point $\mathbf{x} \in L_{out}$ is defined as follows:

$switch\_in(\mathbf{x})$:

- Step 1: Delete $\mathbf{x}$ from L_{out} and add it to L_{in}. Set $\phi(\mathbf{x}) = -1$.
- Step 2: $\forall \mathbf{y} \in N(\mathbf{x})$ satisfying $\phi(\mathbf{y}) = 3$, add $\mathbf{y}$ to L_{out}, and set $\phi(\mathbf{y}) = 1$.

The first step in the $switch\_in()$ procedure switches $\mathbf{x}$ from L_{out} to L_{in}. With $\mathbf{x} \in L_{in}$ now, all its neighbors that were exterior points before 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 $switch\_out()$ for a point $\mathbf{x} \in L_{in}$ is defined as follows:

$switch\_out(\mathbf{x})$:

- Step 1: Delete $\mathbf{x}$ from L_{in} and add it to L_{out}. Set $\phi(\mathbf{x}) = 1$.
- Step 2: $\forall \mathbf{y} \in N(\mathbf{x})$ satisfying $\phi(\mathbf{y}) = -3$, add $\mathbf{y}$ to L_{in}, and set $\phi(\mathbf{y}) = -1$.

By applying a $switch\_out()$ procedure to an inside neighboring grid point, we move the boundary inward by one grid point.

With the basic steps of moving neighboring grid points defined, we propose our fast implementation for general evolution speeds. At every iteration, we first compute the speed at all points in L_{out} and L_{in} and store its sign in the array 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 $F > 0$. 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, and they are deleted. We then scan through the list L_{in} and apply a $switch\_out()$ procedure for a point with $F < 0$. This scan moves those parts of the curve with negative speed inward by one grid point. Similarly, 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

Table 1: Fast Algorithm For General Evolution Speeds

- Step 1: Initialize the array ϕ, F, and the two lists L_{out} and L_{in}.
- Step 2: Compute the speed F for all the points in L_{out} and L_{in}.
- Step 3: Scan through the two lists to update ϕ, L_{out} and L_{in}.
  - **Outward evolution**. Scan through L_{out}. For each point $\mathbf{x} \in L_{out}$, $switch\_in(\mathbf{x})$ if $F(\mathbf{x}) > 0$.
  - **Eliminate redundant points in $\mathbf{L_{in}}$**. Scan through L_{in}. For each point $\mathbf{x} \in L_{in}$, if $\forall \mathbf{y} \in N(\mathbf{x}), \phi(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\phi(\mathbf{x}) = -3$.
  - **Inward evolution**. Scan through L_{in}. For each point $\mathbf{x} \in L_{in}$, $switch\_out(\mathbf{x})$ if $F(\mathbf{x}) < 0$.
  - **Eliminate redundant points in $\mathbf{L_{out}}$**. Scan through L_{out}. For each point $\mathbf{x} \in L_{out}$, if $\forall \mathbf{y} \in N(\mathbf{x}), \phi(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\phi(\mathbf{x}) = 3$.
- Step 4: If the stopping condition is satisfied, terminate the algorithm; otherwise, go back to Step 2.

following conditions is satisfied:

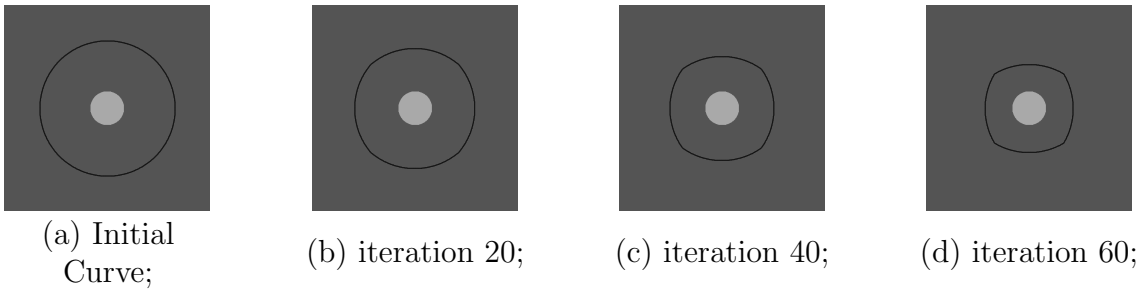
(a) The speed at all the neighboring grid points satisfies:

$$\begin{aligned} F(\mathbf{x}) &\leq 0 \quad \forall \mathbf{x} \in L_{out}; \\ F(\mathbf{x}) &\geq 0 \quad \forall \mathbf{x} \in L_{in}. \end{aligned} \tag{4}$$

(b) A pre-specified maximum number of iterations is reached.

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

The complexity of our algorithm is linear with respect to the number of neighboring grid points. Let the maximum number of operations needed for a $switch\_in()$ or $switch\_out()$ procedure be P_1 and the number of operations needed for the deletion of an interior or an exterior point from L_{in} or L_{out} be P_2. Then the total number of operations per iteration is bounded by $(P_1 + P_2)(length(L_{out}) + length(L_{in}))$ where $length(L_{out})$ and $length(L_{in})$ are the number of points in the two point lists. With our algorithm, it is also easy to estimate the typical computation needed to evolve from the initial curve to the final boundary. For a typical curve evolution problem, every point in the domain D is passed at most once. Let the number of grid points between the initial curve and the final boundary be A, the computation can then be bounded by $2A(P_1 + P_2)$, where the factor 2 is used to take into account that every point can be both an inside and outside neighboring grid point once.



(a) Initial Curve; (b) iteration 20; (c) iteration 40; (d) iteration 60;

(e) iteration 79; (f) iteration 99; (g) iteration 119; (h) iteration 139.

Figure 2: The curve evolution process of our fast algorithm for general evolution speeds.

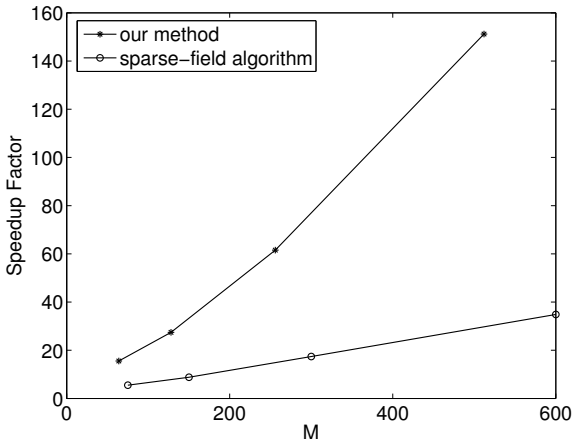


Figure 3: A comparison of the speedup factor to the global level set method between our fast algorithm for general evolution speeds and the sparse-field algorithm.

Compared with previous narrow band methods, our algorithm limits the computation to only the two lists of neighboring grid points. In this sense, it can be viewed as an extreme narrow banding. However, the fundamental difference is that we evolve the curve with no need of solving the level set PDE, 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. Since we do not solve any PDE, there is no need of careful step size control to maintain numerical stability in our method and reinitialization is not an issue. In addition, all our computations are on integers and this also makes our algorithm more efficient.

To illustrate the improvement of our algorithm to previous narrow band algorithms, we compare with the "sparse-field algorithm" in [32]. The sparse-field algorithm is also based on the narrow banding idea and is significantly faster than the original narrow band technique proposed in [3]. Open source implementation of the sparse-field algorithm for imaging applications is available in ITK [1] and this makes further comparisons on image segmentation examples easier.

In [32], a simple example that shrinks a circle with constant speed was implemented with both the global level set method [21] and the sparse-field algorithm. Using the running times in [32] 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. 3. In this paper, we compare our fast algorithm with the global level set method with a similar example. For a domain of size $M \times M$, we shrink a circle of radius $M/3$ with constant speed toward a circle of radius $M/12$. Both circles are centered at $(M/2, M/2)$. In Fig. 2(a), we show the speed field used, together with the initial curve in black, for $M = 512$. The speed is 1 inside the small ball of radius $M/12$ and -1 everywhere else. For the global level set method, we start with 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 circle of interest. Since the speed field is very simple here, no reinitialization is necessary for the global level set method. For different domain sizes, the running times of both algorithms on a 3.2GHz PC are listed in Table II. Unless otherwise noted, all other experiments in this paper are also run on the same computer. We have also plotted the speedup factor of our fast algorithm over the global level set method in Fig. 3. From the results shown in Fig. 3, we can see that our fast algorithm has achieved significantly better speedups than the sparse-field algorithm and the advantage of our fast algorithm grows as the domain size increases. In Fig. 2 (a)-(h), we show the curve evolution process implemented with our fast algorithm for $M = 512$. Our algorithm converges to the small circle automatically after 139 iterations by satisfying part (a) of the stopping condition. Though our algorithm does not maintain the curve as an exact circle during the evolution process, it locates the final circle accurately at a much faster pace, which is desirable for tasks such as segmentation and tracking. For real imaging applications, the speed field is far more complicated than the one used in this example and reinitialization is necessary for the sparse-field algorithm and other previous narrow band algorithms. Taking this

Table 2: Comparison Of Running Times For Different Domain Sizes.

domain size	global level set method	fast algorithm for general speeds
64×64	0.00210s	0.000135s
128×128	0.0156s	0.000569s
256×256	0.144s	0.00234s
512×512	1.527s	0.0101s

into account, the advantage of our algorithm will be more dramatic as we will show later in image segmentation examples.

Note that an improvement to our algorithm can be achieved if the evolution speed is dominantly inward or outward. For example, if the speed is -1 for most of the boundary points, the procedure $switch\_out()$ is more frequent than the procedure $switch\_in()$. In this case, we can use only one neighboring grid point list L_{in} since the two lists complement each other. For the example in Fig.2, we have observed a speedup of around 50% with this strategy. Due to space limitation, we will not expand on this topic.

4 Fast Smoothness Regularization

In the application of curve evolution to many imaging problems, the evolution speed F is composed of a data dependent external speed F_{ext} and an internal speed F_{int} for smoothness regularization. A popular choice of F_{int} is the curvature of the curve [7, 18, 29].

In the level set method, the numerical evaluation of curvature using the function ϕ is computationally quite expensive, unless ϕ is chosen as the signed distance function such that the curvature equals the Laplacian of ϕ. From the theory of scale-space [10, 13, 24], we know that the evolution of a function according to its Laplacian is equivalent to Gaussian filtering of the function. Motivated by this observation, we propose a new method of incorporating smoothness regularization in the curve evolution process based on Gaussian filtering of the level set function.

In our method, we separate the evolution due to the data dependent speed F_{ext} and the smoothness regularization into two cycles. This is different from previous approaches [7,18,29] where the speed force due to the curvature term for regularization is added to F_{ext} to form a single evolution speed. No matter how small the weight for the curvature term is, the expensive computation to evaluate the curvature has to be performed at every iteration. With our method, the smoothness regularization is realized with a simple Gaussian filtering that can be approximated with integer operations. Since this smoothing procedure is separated with the evolution driven by F_{ext}, it can be performed much less frequently if the noise level is low. This approach not only reduces the computational cost dramatically, but also enables the curve to get deeper into concave regions while preserving the smoothness of the boundary.

Table 3: The Two-Cycle Fast Algorithm

- Step 1: Initialize the array ϕ, F_{ext}, the two lists L_{out} and L_{in}.
- Step 2 (**cycle one: evolution using the external speed**):
  For i=1:N_a do
  - Compute the external speed F_{ext} for each point in L_{out} and L_{in}.
  - **Outward evolution**. For each point $\mathbf{x} \in L_{out}$, $switch\_in(\mathbf{x})$ if $F_{ext}(\mathbf{x}) > 0$.
  - **Eliminate redundant points in $\mathbf{L_{in}}$**. For each point $\mathbf{x} \in L_{in}$, if $\forall \mathbf{y} \in N(\mathbf{x}), \phi(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\phi(\mathbf{x}) = -3$.
  - **Inward evolution**. For each point $\mathbf{x} \in L_{in}$, $switch\_out(\mathbf{x})$ if $F_{ext}(\mathbf{x}) < 0$.
  - **Eliminate redundant points in $\mathbf{L_{out}}$**. For each point $\mathbf{x} \in L_{out}$, if $\forall \mathbf{y} \in N(\mathbf{x}), \phi(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\phi(\mathbf{x}) = 3$.
  - Check the stopping condition. If it is satisfied, go to Step 3; otherwise continue this cycle.
- Step 3 (**cycle two: smoothing the curve with Gaussian filtering**):
  For i=1:N_g do
  - **Outward evolution**. For every point $\mathbf{x}$ in L_{out}, compute $G \otimes \phi(\mathbf{x})$. If $G \otimes \phi(\mathbf{x}) < 0$, $switch\_in(\mathbf{x})$.
  - **Eliminate redundant points in $\mathbf{L_{in}}$**. For each point $\mathbf{x} \in L_{in}$, if $\forall \mathbf{y} \in N(\mathbf{x}), \phi(\mathbf{y}) < 0$, delete $\mathbf{x}$ from L_{in}, and set $\phi(\mathbf{x}) = -3$.
  - **Inward evolution**. For every point $\mathbf{x}$ in L_{in}, compute $G \otimes \phi(\mathbf{x})$. If $G \otimes \phi(\mathbf{x}) > 0$, $switch\_out(\mathbf{x})$.
  - **Eliminate redundant points in $\mathbf{L_{out}}$**. For each point $\mathbf{x} \in L_{out}$, if $\forall \mathbf{y} \in N(\mathbf{x})$, $\phi(\mathbf{y}) > 0$, delete $\mathbf{x}$ from L_{out}, and set $\phi(\mathbf{x}) = 3$.
- Step 4: If the stopping condition is satisfied in cycle one, terminate the algorithm; otherwise, go back to Step 2.

The two-cycle algorithm we propose is listed in Table II. In cycle one of the algorithm, we run N_a iterations of curve evolution with the speed F_{ext} using the fast algorithm for general speed fields we proposed in the previous section. In the second cycle, we apply Gaussian filtering to the level set function to smooth the curve. We choose an isotropic Gaussian shaped filter G of size $N_g \times N_g \times \cdots \times N_g$ in the K dimensional space. Since we are only interested in the smoothness of the zero level set, we compute the response of ϕ to the filter G at the grid points in L_{out} and L_{in}. To maintain the integer valued level set function ϕ as defined in (3), we do not take the output of the Gaussian filter directly. In stead, we apply a $switch\_in()$ or $switch\_out()$ procedure to a grid point only when the sign of the output is different from the original value of ϕ at that point, since the curve has moved; otherwise, the value of ϕ at the grid point remains the same. One iteration of the smoothing process is summarized as follows:

A smoothing iteration:

- For every grid point $\mathbf{x}$ in L_{out}, compute $G \otimes \phi(\mathbf{x})$. If $G \otimes \phi(\mathbf{x}) < 0$, $switch\_in(\mathbf{x})$;
- For every grid point $\mathbf{x}$ in L_{in}, compute $G \otimes \phi(\mathbf{x})$. If $G \otimes \phi(\mathbf{x}) > 0$, $switch\_out(\mathbf{x})$.

In cycle two, we run the smoothing process for N_g iterations. If the stopping condition is satisfied during the first cycle, we terminate the algorithm; otherwise, we return to the first cycle and continue this iterative process. The stopping condition used here is the same as the one used for the general evolution speeds in the previous section.

In our two-cycle algorithm, there are two regularization parameters: N_a and N_g balancing the external speed and the smoothing effects, but their selection is more intuitive than the regularization parameter for curvature-based smoothing. The parameter N_g is geometrically related with the elimination of small holes in the final result. To eliminate holes with radius smaller than r, we can choose $N_g = 2r$. By changing the parameter N_a, we can control the weight between the evolution driven by the external speed and the smoothness regularization. Normally it is chosen to be bigger than N_g so that we can respect the data term while incorporating smoothness regularization.

To implement the Gaussian filtering procedure, we scale the coefficients of the Gaussian shaped filter and approximate all the computation with integer operations since we only care about the sign of the filtered result. This further speeds up our algorithm.

To demonstrate the advantages of the two-cycle algorithm, we show an image segmentation example in Fig. 4 and 5. The original image is a proton MRI brain image from the ITK software package [1]. It is shown in Fig. 4(a), together with the initial curve. Since the focus of this paper is on reducing the computational cost of the level set method with our fast implementation, we use simple threshold-based speeds in all our image segmentation and tracking examples, but our algorithms can also be applied to general, application dependent, speed fields to achieve similar speedups.

For this example, the external speed F_{ext} based on thresholding is defined as:

$$F_{ext} = \begin{cases} 1, & \text{if } f(\mathbf{x}) \in [I_1, I_2]; \\ -1, & \text{otherwise,} \end{cases} \tag{5}$$

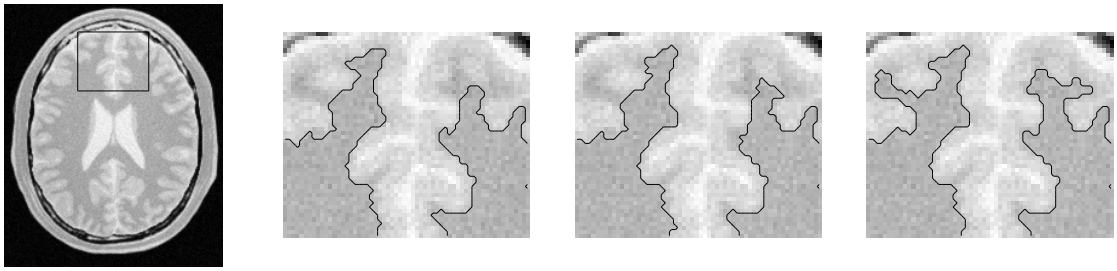
where f denotes the image intensity, and $[I_1, I_2]$ is the range of intensities for the region to be segmented. Here we choose the intensity range of interest as $[150, 180]$. For the curvature-based regularization, the evolution speed is:

$$F = F_{ext} - \lambda\kappa. \tag{6}$$

We implement the curve evolution according to the speed field in (6) with both the sparse-field algorithm [32] in ITK and our fast algorithm for general speed fields. Though both algorithms use the same speed field, slight differences may exist in the final results due to different numerical implementations. The segmentation results from these two algorithms are shown in Fig. 4(b) and (c), respectively. For both algorithms, the parameter λ is tuned to have similar level of small holes in the result, with $\lambda = 1$ for ITK and $\lambda = 0.8$ in our implementation. We also apply our two-cycle algorithm to segment this image using the external speed in (5) and the regularization parameters are chosen as $N_g = 3$ and $N_a = 20$. Our two-cycle algorithm converges automatically by satisfying part (a) of the stopping condition. Its result is shown in Fig. 4(d). To compare the segmentation results of the three methods in more detail, we show a zoomed version of the results in Fig. 5(b), (c) and (d) for a region delineated by the black box in Fig. 5(a). We can see that the results of ITK and our algorithm for general evolution speeds are similar, but the result of the two-cycle algorithm is better in the sense that it gets deeper into concave regions whose intensities are within the range $[I_1, I_2]$. The execution times of the three algorithms are: 1.608s for ITK, 0.0338s for our algorithm for general evolution speeds, and 0.00853s for our two-cycle algorithm. Thus compared with the sparse-field algorithm, we have achieved a speedup factor of 47 and 188 by using our fast algorithm for general speeds and two-cycle algorithm, respectively.

(a) (b) (c) (d)

Figure 4: MRI brain image segmentation results.(a) The original image and the initial curve (the black circle). (b) The result of ITK. (c) The result of our fast algorithm for general evolution speeds. (d) The result of our two-cycle fast algorithm.



(a) (b) (c) (d)

Figure 5: A detailed comparison of the segmentation results. (a) The region of interest is specified by the black box. (b) The result of ITK. (c) The result of our fast algorithm for general evolution speeds. (d) The result of our two-cycle fast algorithm.

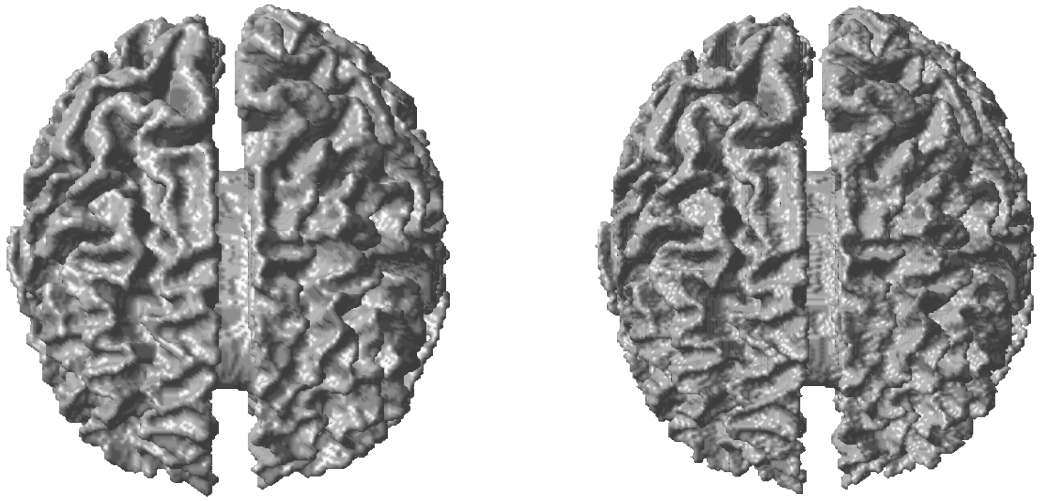
5 Experimental Results in Time Critical Applications

In this section, we will present more experimental results of our two-cycle fast algorithm in two time critical applications: 3D image segmentation and real-time video tracking.

5.1 Fast 3D image segmentation

In this experiment, we apply the threshold-based speed in (5) and (6) to a simulated 3D T1 MRI brain image from the BrainWeb [2]. The size of the image is $181 \times 217 \times 181$. Each slice is $1mm$ thick. The noise level is 3% and the intensity non-uniformity is 20%.

We compare the results of our two-cycle algorithm with the sparse-field algorithm implemented in ITK. The parameter λ is chosen as 0.75 for ITK. For our algorithm, we choose $N_g = 3$ and $N_a = 50$. The intensity range for the region of interest is $[120, 160]$. The initial surface is a ball of radius 5 centered at the coordinate $(90, 110, 100)$. Our algorithm converges automatically in 183 iterations by satisfying part (a) of the stopping condition and the running time is 3.398s. The ITK algorithm uses 1174 iterations and the running time is 777.71s. The segmented surface for both methods are shown in Fig. 6(a) and (b). To further compare the results, we have plotted the segmented boundary over 6 axial slices distributed evenly between slice 60 and 120 in Fig. 7. The results of our method and ITK are plotted in white and black color respectively. As we can see, for the most part the two boundaries overlap with each other. For slice 60 and 72, our algorithm has been able to go deeper into regions that ITK can not. This is consistent with the results shown in the last section. Overall, we have obtained comparable 3D segmentation results with ITK and a speedup over the sparse-field algorithm of two orders of magnitude has been achieved.



(a) (b)

Figure 6: The reconstructed surfaces. (a) Our two-cycle algorithm. (b)ITK.

5.2 Real-time video tracking

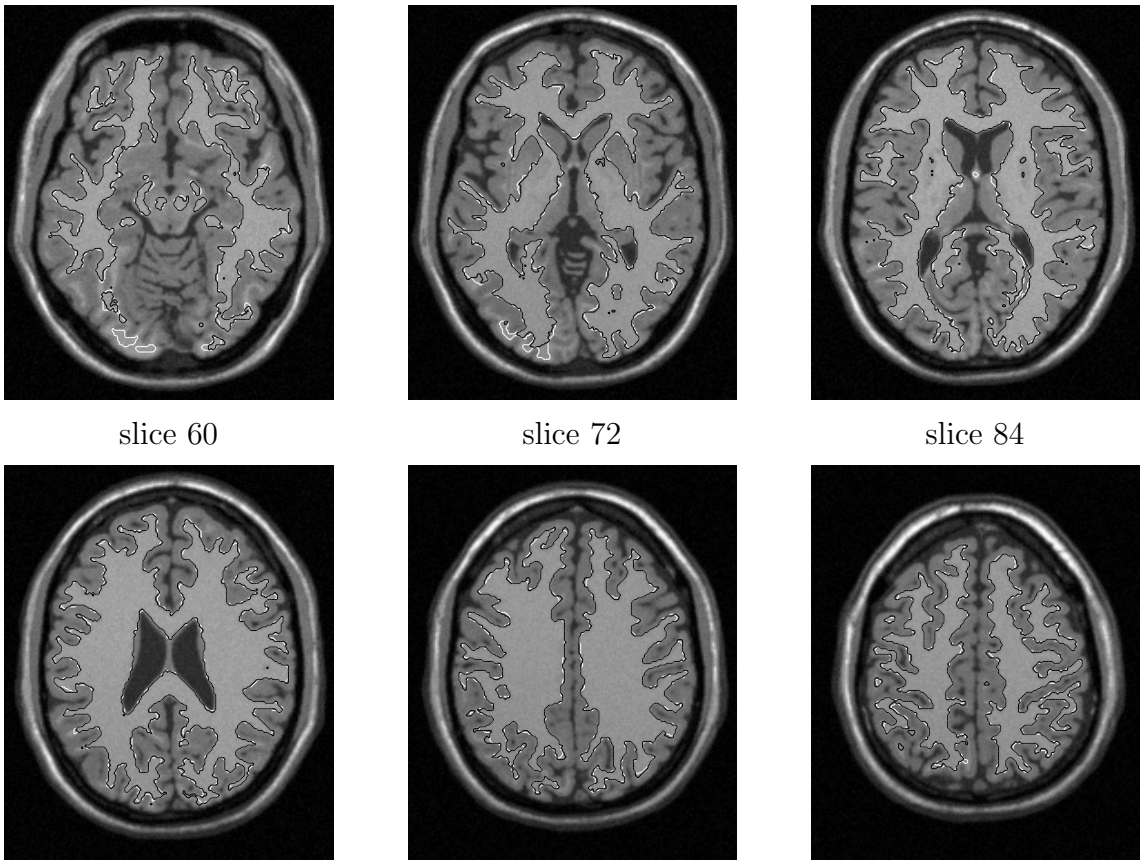
The application of the level set method for video tracking has attracted significant interest and much work has been done in this direction, such as [4,5,8,16,17,22], but to the best of our knowledge, no real-time system has been reported. In this section, we apply our two-cycle fast algorithm to implement a color-based real-time tracking system.

Color has been an important cue for tracking human faces and gestures [19,26,28, 33]. In our system, we represent color in the HSV color space and model the hue and saturation components with a 2D Gaussian distribution. For each video sequence, an initial curve is specified over the region of interest in the first frame. The color of all the pixels inside the curve is transformed into the HSV color space. To this transformed data set, we perform a principal component analysis to their hue and saturation components. Let the mean of the data be **hs**, two principal components be $\mathbf{v_1}, \mathbf{v_2}$ and the corresponding eigenvalues be λ_1 and λ_2, we then track all the pixels with the color $\mathbf{f}$ satisfying:

$$\frac{<\mathbf{f}-\mathbf{hs},\mathbf{v_1}>^2}{\lambda_1}+\frac{<\mathbf{f}-\mathbf{hs},\mathbf{v_2}>^2}{\lambda_2}<p \tag{7}$$

where $\mathbf{f}$ is a vector composed of the hue and saturation of the color at a pixel, and p is a threshold. To track this color distribution, the external speed is designed as follows:

$$F_{ext}=\text{sign}\left(p-\frac{<\mathbf{f}-\mathbf{hs},\mathbf{v_1}>^2}{\lambda_1}-\frac{<\mathbf{f}-\mathbf{hs},\mathbf{v_2}>^2}{\lambda_2}\right).$$



slice 60 slice 72 slice 84

slice 96 slice 108 slice 120

Figure 7: A comparison of the segmentation results by our two-cycle algorithm and ITK. White contour: our method; black contour: ITK.

Figure 8: Tracking results of the *Claire* video sequence by our two-cycle fast algorithm.

This speed drives the curve outward if (7) is satisfied and shrinks the curve otherwise. Starting from the second frame, the tracking result of the previous frame is used as the initial curve. All the parameters are fixed for the whole sequence once they are selected at the first frame.

In our first tracking experiment, we apply our two-cycle algorithm to the *Claire* video sequence in CIF format with 168 frames, which is used frequently in the video coding literature. The parameters used here are $p = 2$, $N_a = 20$, and $N_g = 7$. The tracking results for 8 of the 168 frames are plotted as a white contour in Fig. 8. As we can see, our algorithm successfully tracked the face of the female character in the sequence. Note that the smoothness regularization has been able to keep the contour from attracted into dark regions like the eyebrow. The total time being spent on tracking the sequence is 0.502s, which is around 0.003s per frame. This is much faster than the requirement of real-time tracking.

Encouraged by this result, we have implemented a real-time skin tracking system using Matlab on a 1.7 GHz PC. A block diagram of our system is shown in Fig. 9. We use the image acquisition toolbox in Matlab to capture the current frame of the video in CIF format from a WebCam, 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 displaying. After that, the next frame from the video camera is captured to continue the tracking process. This system runs comfortably at 24 frames per second. In Fig. 10, we show an example of the tracking results from our system. The parameters used are $p = 9$, $N_a = 20$, and $N_g = 7$. For each frame, the tracking result is plotted as a white contour. From the sequence, we can see that we have tracked the head and hand successfully. Topological changes are handled automatically in this process.

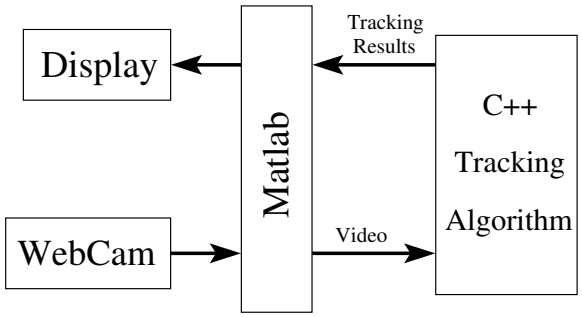


Figure 9: The block diagram of our real-time tracking system.

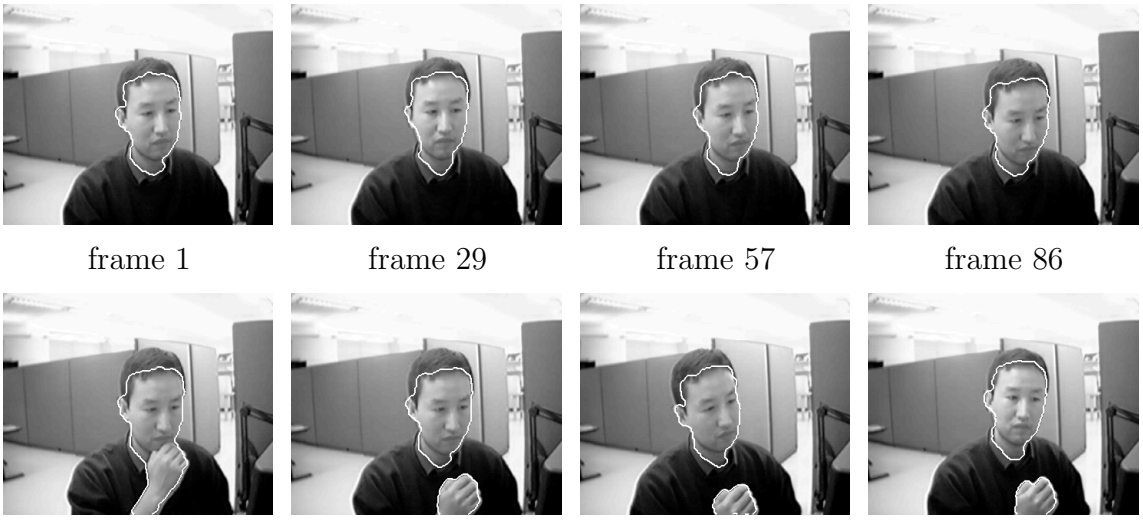


Figure 10: Tracking results of our real-time system for a video sequence taken with a WebCam in our lab.

6 Conclusions

In this paper, we have proposed a fast level set implementation that is suitable for many time critical imaging problems such as 3D segmentation and real-time tracking. Our method is based on simultaneous update of two neighboring grid point lists and a specially designed level set function. Without the need of solving any PDEs, our method still maintains the advantages of the level set method. Compared with previous narrow band algorithms, dramatic speedups have been demonstrated.

References

[1] The Insight Toolkit. [online] Available:http://www.itk.org.

[2] Brainweb: Simulated Brain Database. [online] Available: http://www.bic.mni.mcgill.ca/brainweb.

[3] D. Adalsteinsson and J. Sethian, "A fast level set method for propagating interfaces," *Journal of Computational Physics*, vol. 118, pp. 269–277, 1995.

[4] M. Bertalmio, G. Sapiro, and G. Randall, "Morphing active contours: A geometric approach to topology-independent image segmentation and tracking," *Proc. ICIP*, vol. III, pp. 318–322, 1998.

[5] S. Besson, M. Barlaud, and G. Aubert, "Detection and tracking of moving objects using a new level set based method," *Proc. ICPR*, vol. 3, pp. 1100 – 1105, Sept 2000.

[6] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *Int'l Journal of Computer Vision*, vol. 22, no. 1, pp. 61–79, 1997.

[7] T. Chan and L. Vese, "Active contours without edges," *IEEE Trans. Image Processing*, vol. 10, pp. 266–277, Feb 2001.

[8] D. Freedman and T. Zhang, "Acitve contours for tracking distributions," *IEEE Trans. Image Processing*, vol. 13, pp. 518–526, Apr 2004.

[9] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky, "Fast geodesic active contours," *IEEE Trans. Image Processing*, vol. 10, pp. 1467–1475, Oct 2001.

[10] A. Hummel, "Representations based on zero-crossings in scale-space," in *Proc. CVPR*, pp. 204–209, 1986.

[11] J.Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci*, vol. 93, no. 4, pp. 1591–1595, 1996.

[12] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contour models," in *Proc. ICCV*, (Boston,USA), pp. 810–815, 1995.

[13] J. Koenderink, "The structure of images," *Biol. Cybern.*, vol. 50, pp. 363–370, 1984.

[14] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker, "A streaming narrow-band algorithm: interactive computation and visualization of level sets," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, pp. 422–433, Jul/Aug 2004.

[15] R. Malladi, J. Sethian, and B. Vemuri, "Shape modeling with front propagation: a level set approach," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 158–175, Feb 1995.

[16] A. Mansouri, "Region tracking via level set PDEs without motion computation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, pp. 947–961, Jul 2002.

[17] D. Mukherjee, N. Ray, and S. Acton, "Level set analysis for leukocyte detection and tracking," *IEEE Trans. Image Processing*, vol. 13, pp. 562–572, Apr 2004.

[18] D. Mumford and J. Shah, "Boundary detection by minimizing functionals," in *Proc. CVPR*, (San Francisco), pp. 22–26, Jun 1985.

[19] N. Oliver, A. Pentland, and F. Berarad, "Lafter: Lips and face real time tracker," in *Proc. CVPR*, pp. 123–129, 1997.

[20] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, 2002.

[21] S. Osher and J. Sethian, "Fronts propagation with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of computational physics*, vol. 79, pp. 12–49, 1988.

[22] N. Paragios and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 266–280, Mar 2000.

[23] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics*, vol. 155, pp. 410–438, 1999.

[24] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, Jul 1990.

[25] F. Precioso, M. Barlaud, T. Blu, and M. Unser, "Smoothing B-spline active contour for fast and robust image and video segmentation," *Proc. ICIP*, Sept 2003.

[26] Y. Raja, S. McKenna, and S. Gong, "Tracking and segmenting people in varying lighting conditions using colour," in *Proc. Int'l Conf. Automatic Face and Gesture Recognition*, pp. 228–233, 1998.

[27] J. Sethian, *Level set methods and fast marching methods : evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1999.

[28] L. Sigal, S. Sclaroff, and V. Athitsos, "Skin color-based vidoe segmentation under time varying illumination," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 26, pp. 862–877, Jul 2004.

[29] A. Tsai, A. Yezzi, and A. Willsky, "Curve evolution implementation of the Mumford-Shah functional for image segmentation, denoising, interpolation, and magnification," *IEEE Trans. Image Processing*, vol. 10, pp. 1169–1186, Aug 2001.

[30] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 1528–1538, Sep 1995.

[31] J. Weickert, B. Romeny, and M. Viergever, "Efficient and reliable scheme for nonlinear diffusion filtering," *IEEE Trans. Image Processing*, vol. 7, pp. 398–410, Mar 1998.

[32] R. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int'l Journal of Computer Vision*, vol. 29, pp. 203–231, Oct 1998.

[33] J. Yang, L. Weier, and A. Waibel, "Skin-color modeling and adaptation," in *Proc. Asian Conf. Computer Vision*, vol. 2, pp. 687–694, 1998.

[34] X. Zeng, L. Staib, R. Schultz, and J. Duncan, "Segmentation and measurement of the cortex from 3D MR images using coupled surfaces propagation," *IEEE Trans. Med. Imag.*, vol. 18, pp. 927–937, Oct 1999.