DISCRETE CURVATURE CALCULATION FOR FAST LEVEL SET SEGMENTATION

Jan Kybic, Jakub Krátký

Center for Machine Perception,
Department of Cybernetics, Faculty of Electrical Engineering,
Czech Technical University, Prague, Czech Republic
kybic@fel.cvut.cz

ABSTRACT

Fast level set methods replace continuous PDEs by a discrete formulation, improving the execution times. The regularization in fast level set methods was so far handled indirectly via level set function smoothing. We propose to incorporate standard curvature based regularization into fast level set methods and address the problem of efficiently estimating local curvature of a discretized interface in 2D or 3D based on local partial volume. We present two algorithms for incremental partial volume evaluation: the first is recommended for moderate neighborhood sizes, the second has an excellent asymptotic complexity and can be useful for very large neighborhoods. The performance of the proposed methods is compared experimentally with previous approaches.

Index Terms— Image segmentation, level sets, curvature estimation.

1. INTRODUCTION

Level sets are widely used image segmentation methods [1, 2, 3]. They do not impose any parametric model, can handle topology changes, their implementation is relatively simple and directly extends to higher dimensions. They are based on an iterative solution of discretized partial differential equations (PDEs) and are computationally very expensive, typically too slow for real time 2D or interactive 3D applications, even with acceleration techniques such as narrow band methods [4].

Our work is based on a fast discrete level set segmentation method (FLS) [5, 6], described in Section 2, which similarly to other discrete level set methods [7, 8] replace the continuous PDE formulation by a discrete one. In [5], discrete evolution steps are alternated with convolution based smoothing. Besides a certain inelegance, this choice has two drawbacks. First, the computational complexity of the convolution is $O(h^d)$ per boundary point, where h is the Gaussian filter size and d is the dimensionality of the space. Second, the procedure is not stable, sometimes the smoothing steps cancels or almost cancels the effect of the preceding evolution steps, which makes the boundary to oscillate and prevents convergence. In this contribution, we propose to reintroduce curvature based regularization into the FLS method.

Curvature estimation methods are mostly based on local fitting (of e.g. a conic) [9, 10, 11] using an iterative procedure or an eigenvalue system solution which is not only too slow for our purposes but we also found that the estimation is unstable when the curvature inside the neighborhood is high. While curvature regularization is equivalent to boundary length minimization, the discrete boundary length does not approximate the Euclidean length well [12] and the boundary, instead of getting smoother, develops polygonal shapes [13].

We have chosen to estimate local curvature based on the proportion of inner pixels (partial volume) in a neighborhood of radius d around the current point [14], see Section 3. This method extends directly to 3D and the inherent averaging gracefully handles both high curvature points and noisy data. We will also present two efficient incremental methods for local inner pixel counting with computational complexity $O(h^{d-1})$ and $O(h^{d-2})$, respectively, as compared with $O(h^d)$ operations per boundary point for the naive approach.

This work was sponsored by the Czech Ministry of Education, Project MSM6840770012.

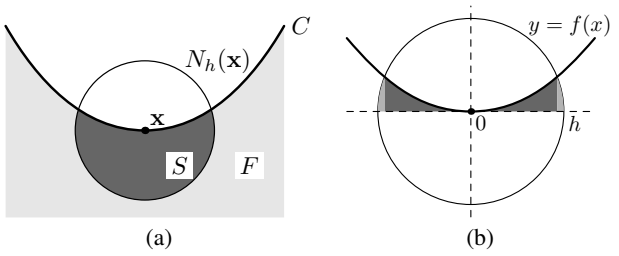


Fig. 1. (a) We show the foreground area F (light gray) and its boundary C with a point $\mathbf{x} \in C$. The local curvature $\kappa(\mathbf{x})$ can be estimated from the area S of the intersection of $F \cap N_h(\mathbf{x})$, where N is a small ℓ_2 neighborhood around $\mathbf{x}$. (b) A detail around point $\mathbf{x}$ shifted to the center of coordinates.

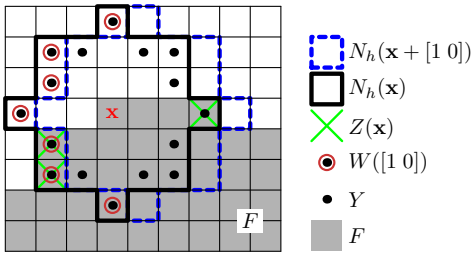


Fig. 2. Detail of the discretized boundary C with the foreground area F in gray and a neighborhood N_h for $h = 3$ (marked by bold lines) centered around a point $\mathbf{x}$ (in red) and the set Y shown by black dots. The difference $W([1\ 0])$ between $N_h(\mathbf{x})$ and shifted $N_h(\mathbf{x}+[1\ 0])$ (in blue and dashed) is marked by additional brown circles and the intersection points Z by green diagonal crosses.

2. LEVEL SET SEGMENTATION

In a continuous formulation, a level set function $\varphi : \mathbb{R}^d \to \mathbb{R}$ defines a foreground area $F = \{\mathbf{x} \in \mathbb{R}^d; \varphi(\mathbf{x}) \leq 0\}$ with a boundary $C = \partial F$ which typically evolves according to

$$\frac{\mathrm{d}C}{\mathrm{d}t}(\mathbf{x}) = v(\mathbf{x})\mathbf{n} = \big(v_{\text{ext}}(\mathbf{x}) + \alpha\kappa(\mathbf{x}) + \lambda\big)\mathbf{n} \tag{1}$$

where v_{ext} is a data dependent speed, κ is a local curvature, $\mathbf{n}$ an outward pointing normal, and α and λ are weights for the curvature and the 'balloon force', respectively. In the discrete case, we replace $\mathbb{R}^d$ by a Cartesian grid $\Lambda \subset \mathbb{Z}^d$.

The fast level set method (FLS) [5] represents the boundary C as two lists of inside and outside boundary points.

$$\begin{aligned} L_{\text{in}} &= \{\mathbf{x} \in \Lambda; \varphi(\mathbf{x}) < 0 \wedge \exists \mathbf{y} \in N^1(\mathbf{x}), \varphi(\mathbf{y}) > 0\} \\ L_{\text{out}} &= \{\mathbf{x} \in \Lambda; \varphi(\mathbf{x}) > 0 \wedge \exists \mathbf{y} \in N^1(\mathbf{x}), \varphi(\mathbf{y}) < 0\} \end{aligned} \tag{2}$$

where $N^1(\mathbf{x}) = \{\mathbf{y} \in \Lambda; \|\mathbf{x} - \mathbf{y}\|_{\ell_1} = 1\}$ is an ℓ_1 neighborhood. FLS further restricts the values of φ to -3 (for points in $F \backslash L_{\text{in}}$), -1 (for points in L_{in}), 1 (for points in L_{out}), and 3 (for all other points).

Only the external speed v_{ext} is used. The segmentation alternates evolution and smoothing phases. In each evolution phase, we first go over all points $\mathbf{x}$ from L_{out}. If a point $\mathbf{x}$ is found such that $v_{\text{ext}}(\mathbf{x}) > 0$ and there is a neighbor $\mathbf{y} \in N^1(\mathbf{x})$ from L_{in} with $v_{\text{ext}}(\mathbf{y}) > 0$, then point $\mathbf{x}$ is moved from L_{out} to L_{in}, updating φ accordingly. This makes the boundary C move outward. Then an equivalent procedure is applied to L_{in}, making C move inward at points where $v_{\text{ext}}(\mathbf{x}) < 0$.

In the smoothing phase, a convolution $\bar{\varphi} = G_\sigma * \varphi$ is evaluated for all points in L_{in} and L_{out}. Where the signs of $\bar{\varphi}$ and φ differ, the points are switched between L_{in} and L_{out} as appropriate.

We propose to omit the smoothing from the FLS algorithm and to use the complete speed v instead of v_{ext} when deciding whether a point should be switched between L_{in} and L_{out}. This way, not only the sign (as in FLS) but also the value of v_{ext} is taken into account, leading to a better trade-off between the regularity and data terms. We shall call the new methods FLSC. Before each evolution phase, we calculate the local mean curvature $\kappa(\mathbf{x})$ at each point from L_{in} and L_{out} as follows.

3. CURVATURE ESTIMATION FROM PARTIAL VOLUME

Consider a point $\mathbf{x}$ on the boundary C of F in 2D (Fig. 1a) and describe the curve C around $\mathbf{x}$ as $y = f(x)$, shifting and turning the coordinate system such that $\mathbf{x}$ is at the origin and $f'(0) = 0$, obtaining a situation in (Fig. 1b). If the neighborhood size h is sufficiently small, we can approximate f as $y = ax^2$, which corresponds to a local curvature $\kappa = f''(0) = 2a$. Then the area of the gray segments in Fig. 1b is

$$\int_{-h}^{h} ax^2 \,\mathrm{d}x = \frac{2ah^3}{3} = \frac{\kappa h^3}{3}$$

If the curvature is sufficiently small ($\kappa h \ll 1$), the light gray wedge-shaped areas in Fig. 1b can be neglected. Consequently, the curvature can be estimated as

$$\kappa = \frac{3\pi}{h}\left(\frac{S}{S_0} - \frac{1}{2}\right) \tag{3}$$

where $S_0 = \pi h^2$ is the area of N_h. It turns out that the approximation is quite accurate, for example if C is a circle and $\kappa h = 0.5$, the error is only $1.2\,\%$ [14].

Algorithm 1: Given $V(\mathbf{x}_0)$ and a neighbor $\mathbf{x}_1$, return $V(\mathbf{x}_1)$

```
V ← V(x0)
foreach w ∈ W(x0 − x1) do
    if x1 + w ∈ F then V ← V + 1
foreach w ∈ W(x1 − x0) do
    if x0 + w ∈ F then V ← V − 1
return V
```

In 3D, the derivation is completely analogous (yet more involved), yielding an estimator of the mean curvature:

$$\kappa = \frac{16}{3h}\left(\frac{V}{V_0} - \frac{1}{2}\right) \tag{4}$$

where V and V_0 are the volumes of $F \cap N$ and N, respectively. We refer an interested reader to reference [14] for a complete derivation including error analysis.

In the discrete case, we define an ℓ_2 neighborhood $N_h(\mathbf{x})$ as

$$N_h(\mathbf{x}) = \{\mathbf{y} \in \Lambda; \|\mathbf{x} - \mathbf{y}\|_{\ell_2} \leq h\} \tag{5}$$

and $V(\mathbf{x}) = |N_h(\mathbf{x}) \cap F|$ and $V_0 = |N_h(\mathbf{x})|$ will be the number of pixels in $N_h(\mathbf{x}) \cap F$ and $N_h(\mathbf{x})$, respectively. Note that V_0 does not depend on $\mathbf{x}$ and can be precalculated.

3.1. Partial volume calculation

We need to evaluate $V(\mathbf{x})$ for all $\mathbf{x}$ from $L_{\text{in}} \cup L_{\text{out}}$. A trivial algorithm (to be called Algorithm 0) has a computational complexity $O(h^d)$ per boundary point going over all points in all neighborhoods.

A better algorithm uses the result $V(\mathbf{x}_0)$ when calculating $V(\mathbf{x}_1)$ for its neighbor $\mathbf{x}_1$. From the definition of $V(\mathbf{x})$ we can derive the following update formula:

$$\begin{aligned} V(\mathbf{x}_1) - V(\mathbf{x}_0) = |\big(N_h(\mathbf{x}_1) \backslash N_h(\mathbf{x}_0)\big) \cap F| - \\ |\big(N_h(\mathbf{x}_0) \backslash N_h(\mathbf{x}_1)\big) \cap F| \end{aligned} \tag{6}$$

The essential idea is to consider only the points which enter and leave the neighborhood N_h as we shift it from $\mathbf{x}_0$ to $\mathbf{x}_1$. The relative coordinates of the entering or leaving points

$$W(\Delta\mathbf{x}) = \{\mathbf{y} \in \mathbb{Z}^d; \mathbf{y} \in N_h(\Delta\mathbf{x}) \wedge \mathbf{y} \notin N_h(\mathbf{0})\} \tag{7}$$

can be precalculated, leading to an Algorithm 1 with complexity $O(h^{d-1})$ per boundary point. The points in L_{in} and L_{out} are examined in a depth first manner without backtracking (Algorithm 2), in order to have a long chain of neighboring points.

3.2. Advanced partial volume calculation

The set $Y = \bigcup W(\Delta\mathbf{x})$ corresponds to the boundary of N_h (Fig. 2). Let us maintain the set $M(\mathbf{x}) = \{\mathbf{y} \in Y; \mathbf{x} + \mathbf{y} \in F\}$ of inner pixels on Y along the boundary C. The difference between $M(\mathbf{x}_0)$ and $M(\mathbf{x}_1)$ for neighboring $\mathbf{x}_0$ and $\mathbf{x}_1$ will be localized around the intersection of C with $Y(\mathbf{x})$, the rest of M remains unchanged. This observation leads directly to Algorithm 3, which can update $V(\mathbf{x})$ in time $O(h^{d-2})$ per boundary point, i.e. in constant time in 2D and in time $O(h)$ in 3D. Besides $M(\mathbf{x})$, the algorithm needs to maintain and incrementally update the partial sums

$$Q(\mathbf{x}; \Delta\mathbf{x}) = \Big|\{\mathbf{y} \in W(\Delta\mathbf{x});\ \mathbf{x} + \mathbf{y} \in F\}\Big|$$

Algorithm 2: Given a list of points L, calculate $V(\mathbf{x})$ in such an order so that subsequent points are neighbors, if possible.

foreach $\mathbf{x} \in L$ **do**
  if $\mathbf{x}$ *is unseen* **then**
    evaluate $V(\mathbf{x})$ by direct counting
    call `do_point`$(\mathbf{x})$

function `do_point`$(\mathbf{x})$:
if *exists* $\mathbf{y} \in N^\infty(\mathbf{x}) \cap F$ **then**
  evaluate $V(\mathbf{y})$ using $V(\mathbf{x})$
  call `do_point`$(\mathbf{y})$
else return from all nested calls of `do_point`

and the set of boundary intersection points

$$Z(\mathbf{x}) = \left\{\mathbf{y} \in Y;\ \mathbf{x}+\mathbf{y} \in F \wedge \exists \mathbf{z} \in N^\infty(\mathbf{x}+\mathbf{y}), \mathbf{z} \notin F\right\}$$

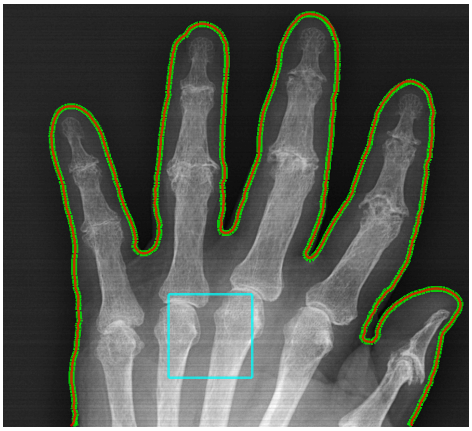
where N^∞ is an ℓ_∞ neighborhood of radius 1.

The update algorithm starts with the points in Z and explores the pixels in Y in a breadth first manner. Each pixel in Y contains links to its neighbors within Y. The breadth first search stops when the ℓ_∞ neighborhood of the point being considered is homogeneous, i.e. all pixels are either from F or $\Lambda \backslash F$. This strategy can fail to find all changed pixels in M if there are two independent boundary segments in $N_h(\mathbf{x})$. In this case we argue that ignoring the component not connected with $\mathbf{x}$ is actually a desired behavior, leading to a curvature estimate only for the current boundary segment.

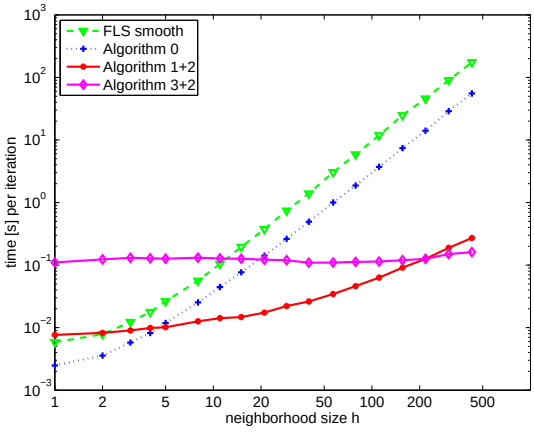
Remarks: (1) Instead of precalculating W for all 3^d-1 moves in the ℓ_∞ sense, we can consider only the $2d$ moves in the ℓ_1 sense and to replace each l_∞ move by up to d ℓ_1 moves. (2) We extend the original four values of φ (see Section 2) with two more classes L'_{in} and L'_{out}, representing points which are part of the inner or outer boundary in the ℓ_∞ sense. This allows for more efficient breadth first search and updating of Z. (3) The coordinates of each point can be represented as a single 32-bit integer, reducing the relevant operation count d times. (4) The image is extended by h pixels in each direction, filled with sentinel values to avoid bounds checking.

Algorithm 3: Given $V(\mathbf{x}_0)$, $Q(\mathbf{x}_0;\cdot)$, $M(\mathbf{x}_0)$ and $Z(\mathbf{x}_0)$, update the values for $\mathbf{x}_1 \in N^1(\mathbf{x}_0)$.

create queue S from $Z(\mathbf{x}_0)$
$Z(\mathbf{x}_1) \leftarrow \emptyset$
$M(\mathbf{x}_1) \leftarrow M(\mathbf{x}_0)$
$Q(\mathbf{x}_1;\cdot) \leftarrow Q(\mathbf{x}_0;\cdot)$
while S *not empty* **do**
  pop $\mathbf{x}$ from S
  foreach $\mathbf{y} \in Y$ *neighbor of* $\mathbf{x}$, *not yet visited* **do**
    if $[\![\mathbf{y} \in M(\mathbf{x}_0)]\!] \neq [\![\mathbf{y} \in M(\mathbf{x}_1)]\!]$ **then**
      update $M(\mathbf{x}_1)$ by adding or removing $\mathbf{y}$
      foreach $\Delta\mathbf{x}$ *such that* $\mathbf{y} \in W(\Delta\mathbf{x})$ **do**
        if $\mathbf{y} \in M(\mathbf{x}_1)$ **then**
          $Q(\mathbf{x}_1;\Delta\mathbf{x}) \leftarrow Q(\mathbf{x}_1;\Delta\mathbf{x}) + 1$
        else $Q(\mathbf{x}_1;\Delta\mathbf{x}) \leftarrow Q(\mathbf{x}_1;\Delta\mathbf{x}) - 1$
    if $\mathbf{y}+\mathbf{x}_1$ *not homogeneous* **then** add $\mathbf{y}$ to S
    if $\mathbf{y}+\mathbf{x}_1 \in L_{in} \cup L'_{in}$ **then** add $\mathbf{y}+\mathbf{x}_1$ to $Z(\mathbf{x}_1)$



(a)

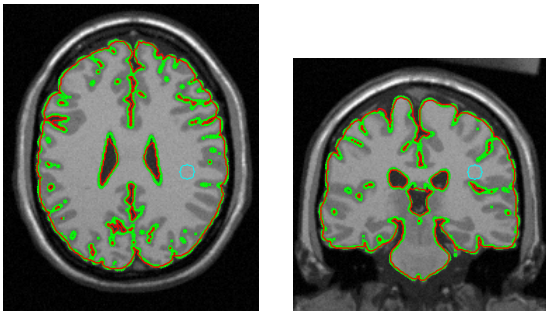


(b)

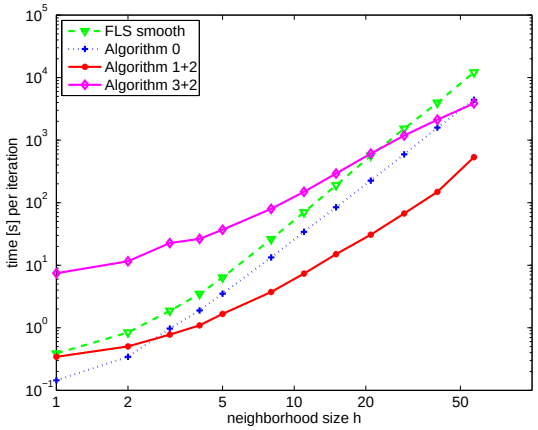
Fig. 3. (a) Segmentation results for a 2D X-ray image. FLSC method results are shown by a wide green line, FLS method by a thinner red line, the cyan rectangle is the initial contour. (b) Time for one iteration of FLS smoothing and FLSC curvature and speed field evaluation with respect to the neighborhood size h.

4. EXPERIMENTS

In the first experiment we have segmented a 2D X-ray image of size 813×897 (Fig 3a) using the FLS method [5] and the proposed FLSC method using Algorithm 1+2. We have used a speed field v_{ext} based on the Chan-Vese criterion [15] scaled to interval $[-1, 1]$, curvature weight $\alpha = 0.5$ and no balloon force ($\lambda = 0$). We have set $h = 5 = 2.5\sigma$ so that both methods consider the same neighborhood size. The FLS method alternated $n_e = 20$ evolution and $n_g = 5$ smoothing steps, each method performed 1000 evolution steps in total. The segmentation results are almost identical. Taking the FLSC result, we have measured the time needed for the FLS smoothing (Section 2) and compared it with the time needed to evaluate the total speed field v using Algorithms 0, 1+2, and 3+2. The results (Fig. 3b) confirm that the time complexity for Algorithm 3 does not depend on h. However, for moderate h, Algorithm 1 is the fastest. For h smaller than about 5, the trivial Algorithm 0 is even faster. The FLS smoothing is slower than Algorithm 0 as floating point opera-



(a)



(b)

Fig. 4. (a) A transverse and coronal slices through a 3D MRI volume with FLS segmentation results in red and FLSC results in green, the initial contour is shown in cyan. (b) Time for one iteration of FLS smoothing and FLSC curvature and speed field evaluation with respect to the neighborhood size h.

tions are involved. However, since in the FLS method smoothing is not applied every iteration, for small h and small amount of smoothing FLS is in fact about twice as fast as our current implementation of FLSC. However, our implementation is not yet fully optimized and is written in a high-level language Ocaml, so we expect that further acceleration is possible.

3D segmentation results are demonstrated on an MRI brain image of size $181 \times 217 \times 181$ (Fig. 4a), using $h = 5$, $\alpha = 0.7$, $n_e = 20$, $n_g = 3$ and 200 evolution steps in total for each method. In this case, some difference between the FLS and FLSC results remain, even after parameter tuning. The FLSC result was chosen as the basis for the subsequent speed comparison. The asymptotic trends are clearly visible in the speed comparison in Fig 4b, with Algorithm 3+2 being the fastest asymptotically and Algorithm 1+2 being the fastest in absolute terms for most practical values of h.

5. CONCLUSIONS

We have described two algorithms for local curvature estimation for 2D and 3D binary objects discretized on a Cartesian grid, which should be applicable in many other areas besides fast level set segmentation.

We have introduced curvature-based regularization into the fast level set method [5], bringing it closer to a standard level set formulations and avoiding its idiosyncrasies while retaining its potential for very fast execution times.

References

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

[2] S. Osher and R. P. Fedkiw, "Level set methods: An overview and some recent results," *Journal of Computational Physics*, vol. 169, no. 2, pp. 463–502, 2001.

[3] P. A. Yushkevich, J. Piven, H. C. Hazlett, R. G. Smith, S. Ho, J. C. Gee, and G. Gerig, "User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability.," *Neuroimage*, vol. 31, no. 3, pp. 1116–1128, July 2006.

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

[5] Y. Shi and W. C. Karl, "A fast level set method without solving PDEs," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 97–100, 2005.

[6] Yonggang Shi and W. Clem Karl, "Real-time tracking using level sets," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2, pp. 34–41, 2005.

[7] Alberto De Santis and Daniela Iacoviello, "A discrete level set approach to image segmentation.," *Signal, Image and Video Processing*, vol. 1, no. 4, pp. 303–320, 2007.

[8] Cristina Darolti, Alfred Mertins, and Ulrich G. Hofmann, "A fast level-set method for accurate tracking of articulated objects with an edge-based binary speed term," in *ACIVS*, Jacques Blanc-Talon, Wilfried Philips, Dan C. Popescu, and Paul Scheunders, Eds. 2007, vol. 4678 of *Lecture Notes in Computer Science*, pp. 828–839, Springer.

[9] Zhang Zhengyou, "Parameter estimation techniques: a tutorial with application to conic fitting," *Image and Vision Computing*, vol. 15, no. 1, pp. 59–76, Jan. 1997.

[10] M. Worring and A.W.M. Smeulders, "Digital curvature estimation," *CVGIP: Image Understanding*, vol. 58, no. 3, pp. 366–382, 1993.

[11] Andrew W. Fitzgibbon, Maurizio Pilu, and Robert B. Fisher, "Direct least square fitting of ellipses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 476–480, 1999.

[12] S.R. Kulkarni, S.K. Mitter, R.J. Richardson, and J.N. Tsitsiklis, "Local versus nonlocal computation of length of digitized curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 7, pp. 711–718, 1994.

[13] L. Yu, Q. Wang, L. Wu, and J. Xie, "A Mumford-Shah model on lattice," *Image and Vision Computing*, , no. 26, pp. 1663–1669, 2008.

[14] J. W. Bullard, E. J. Garboczi, W. C. Carter, and E. R. Fuller Jr., "Numerical methods for computing interfacial mean curvature," *Computational Materials Science*, vol. 4, pp. 103–116, 1995.

[15] T. F Chan and L. A. Vese, "Active contours without edges," *IEEE Trans on Image Processing*, vol. 10, no. 2, pp. 266–277, 2001.