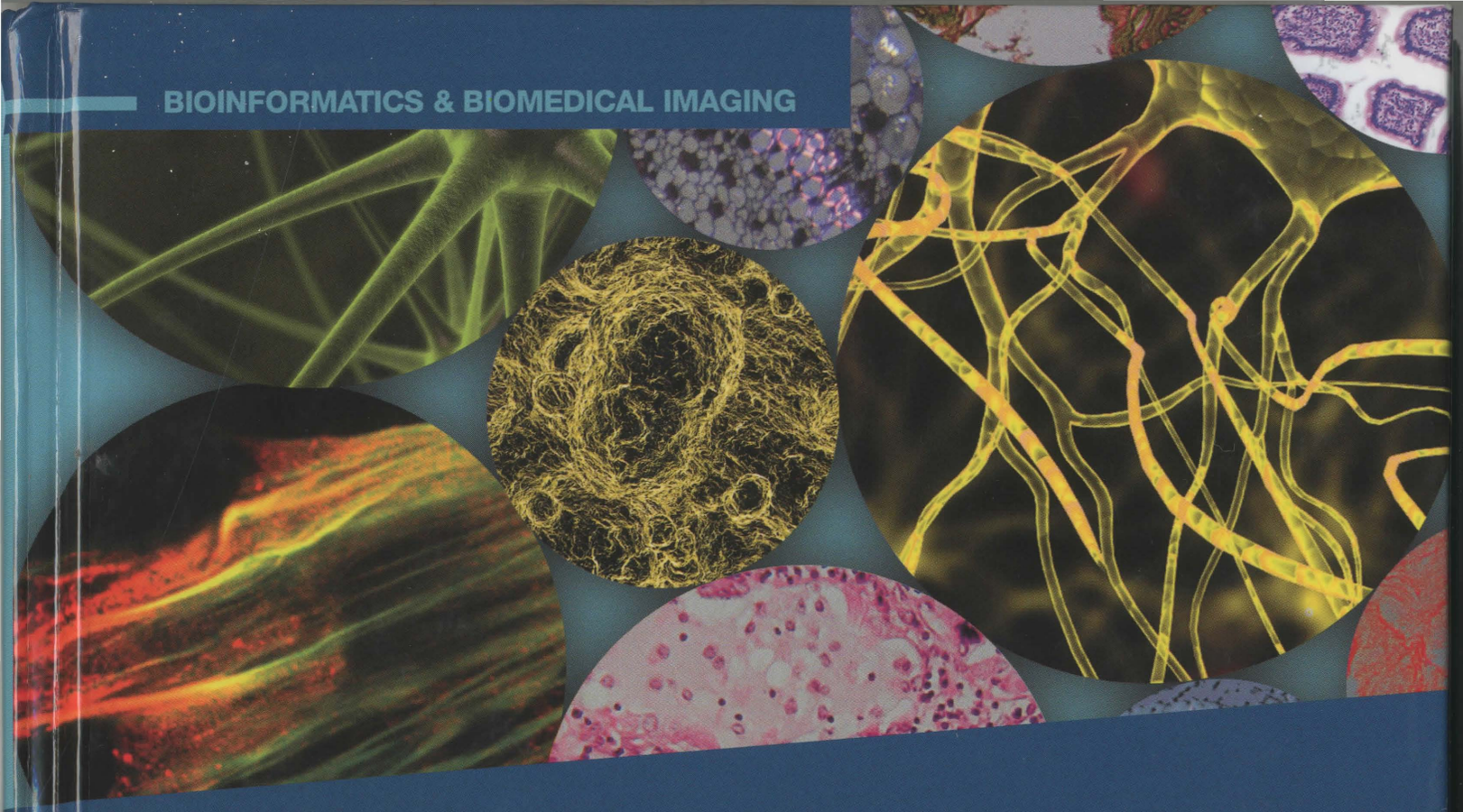
BIOINFORMATICS & BIOMEDICAL IMAGING



High-Throughput Image Reconstruction and Analysis

A. Ravishankar Rao • Guillermo A. Cecchi
editors

CHAPTER 3

Parallel Processing Strategies for Cell Motility and Shape Analysis

K. Palaniappan, F. Bunyak, S. Nath, and J. Goffeney

3.1 Cell Detection

The first step in cell behavior analysis is cell detection. Cell detection returns an initial contour close to actual cell boundaries, that is later refined by the cell segmentation process. A wide range of techniques have been applied for detecting and segmenting biological objects of interest in video microscopy imagery including spatially adaptive thresholding, morphological watershed, mean shift, active contours, graph cuts, clustering, multidimensional classifiers like neural networks, and genetic algorithms. Various factors such as type of the cells, environmental conditions, and imaging characteristics such as zoom factor affect the appearance of cells, and thus choice of detection method. But in general, features used in cell detection can be grouped into three categories: intensity-based, texture-based, and spatio-temporal features. Intensity-based detection approaches (i.e., intensity thresholding [1, 2] or clustering [3]) are suitable for lower resolution or stained images where cells or nuclei appear semi-homogeneous and have distinct intensity patterns compared to the background. For high-resolution unstained images where interior of the cells appear highly heterogeneous or where interior and exterior intensity distributions are similar, features based on spatial texture (i.e., ridgeness measures in [4]) or features based on spatiotemporal features or motion are needed.

In the following section, we describe the flux tensor framework for accurate detection of moving objects (which in this context correspond to moving cells) in time lapse images that effectively handles accurate detection of nonhomogeneous cells in the presence of complex biological processes, background noise, and clutter.

3.1.1 Flux Tensor Framework

The classical spatiotemporal orientation or 3D grayscale structure tensor has been widely utilized for low-level motion detection, segmentation, and estimation [5], since it does not involve explicit feature tracking or correlation-based matching. The traditional structure tensor fails to disambiguate between stationary and moving features such as edges and junctions without an explicit (and expensive) eigendecomposition step at every pixel to estimate a dense image velocity or optical flow field. Flux tensor successfully discriminates between stationary and moving image structures more efficiently than structure tensors using only the trace.

3D Structure Tensors

Structure tensors are a matrix representation of partial derivative information. As they allow both orientation estimation and image structure analysis, they have many applications in image processing and computer vision. 2D structure tensors have been widely used in edge/corner detection and texture analysis; 3D structure tensors have been used in low-level motion estimation and segmentation [6, 7].

Under the constant illumination model, the optic-flow (OF) equation of a spatiotemporal image volume $\mathbf{I}(\mathbf{x})$ centered at location $\mathbf{x} = [x, y, t]$ is given by (3.1) [8] where, $\mathbf{v}(\mathbf{x}) = [v_x, v_y, v_t]$ is the optic-flow vector at $\mathbf{x}$,

$$\begin{aligned}\frac{d\mathbf{I}(\mathbf{x})}{dt} &= \frac{\partial \mathbf{I}(\mathbf{x})}{\partial x}v_x + \frac{\partial \mathbf{I}(\mathbf{x})}{\partial y}v_y + \frac{\partial \mathbf{I}(\mathbf{x})}{\partial t}v_t \\ &= \nabla \mathbf{I}^T(\mathbf{x})\mathbf{v}(\mathbf{x}) = 0\end{aligned} \tag{3.1}$$

and $\mathbf{v}(\mathbf{x})$ is estimated by minimizing (3.1) over a local 3D image patch $\boldsymbol{\Omega}(\mathbf{x}, \mathbf{y})$, centered at $\mathbf{x}$. Note that v_t is not 1 since spatiotemporal orientation vectors will be computed. Using Lagrange multipliers, a corresponding error functional $e_{ls}(\mathbf{x})$ to minimize (3.1) using a least-squares error measure can be written as (3.2) where $W(\mathbf{x}, \mathbf{y})$ is a *spatially invariant* weighting function (e.g., Gaussian) that emphasizes the image gradients near the central pixel [7].

$$\begin{aligned}e_{ls}(\mathbf{x}) = &\int_{\boldsymbol{\Omega}(\mathbf{x},\mathbf{y})} (\nabla \mathbf{I}^T(\mathbf{y})\mathbf{v}(\mathbf{x}))^2 \; W(\mathbf{x}, \mathbf{y}) \; d\mathbf{y} \\ &+ \lambda \left(1 - \mathbf{v}(\mathbf{x})^T \mathbf{v}(\mathbf{x})\right)\end{aligned} \tag{3.2}$$

Assuming a constant $\mathbf{v}(\mathbf{x})$ within the neighborhood $\boldsymbol{\Omega}(\mathbf{x}, \mathbf{y})$ and differentiating $e_{ls}(\mathbf{x})$ to find the minimum, leads to the standard eigenvalue problem (3.3) for solving $\hat{\mathbf{v}}(\mathbf{x})$ the best estimate of $\mathbf{v}(\mathbf{x})$,

$$\mathbf{J}(\mathbf{x}, \mathbf{W})\hat{\mathbf{v}}(\mathbf{x}) = \lambda \hat{\mathbf{v}}(\mathbf{x}) \tag{3.3}$$

The 3D structure tensor matrix $\mathbf{J}(\mathbf{x}, \mathbf{W})$ for the spatiotemporal volume centered at $\mathbf{x}$ can be written in expanded matrix form, without the spatial filter $W(\mathbf{x}, \mathbf{y})$ and the positional terms shown for clarity, as

$$\mathbf{J} = \begin{bmatrix} \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial x}\frac{\partial \mathbf{I}}{\partial x} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial x}\frac{\partial \mathbf{I}}{\partial y} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial x}\frac{\partial \mathbf{I}}{\partial t} d\mathbf{y} \\ \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial y}\frac{\partial \mathbf{I}}{\partial x} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial y}\frac{\partial \mathbf{I}}{\partial y} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial y}\frac{\partial \mathbf{I}}{\partial t} d\mathbf{y} \\ \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial t}\frac{\partial \mathbf{I}}{\partial x} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial t}\frac{\partial \mathbf{I}}{\partial y} d\mathbf{y} & \int_{\boldsymbol{\Omega}} \frac{\partial \mathbf{I}}{\partial t}\frac{\partial \mathbf{I}}{\partial t} d\mathbf{y} \end{bmatrix} \tag{3.4}$$

A typical approach in motion detection is to threshold **trace**(**J**) (3.5); but this results in ambiguities in distinguishing responses arising from stationary versus moving features (e.g., edges and junctions with and without motion), since **trace**(**J**) incorporates total gradient change information but fails to capture the nature of these gradient changes (i.e., spatial only versus temporal).

$$\mathbf{trace}(\mathbf{J}) = \int_{\boldsymbol{\Omega}} \|\nabla I\|^2 d\mathbf{y} \tag{3.5}$$

To resolve this ambiguity and to classify the video regions experiencing motion, the eigenvalues and the associated eigenvectors of $\mathbf{J}$ are usually analyzed [9, 10]. However, eigenvalue decompositions at every pixel are computationally expensive especially if real-time performance is required.

Flux Tensors

In order to reliably detect only the moving structures *without* performing expensive eigenvalue decompositions, the concept of the *flux tensor* is proposed. Flux tensor is the temporal variations of the optical flow field within the local 3D spatiotemporal volume.

Computing the second derivative of (3.1) with respect to t, (3.6) is obtained where, $\mathbf{a}(\mathbf{x}) = [a_x, a_y, a_t]$ is the acceleration of the image brightness located at $\mathbf{x}$.

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{d\mathbf{I}(\mathbf{x})}{dt}\right) = &\frac{\partial^2 \mathbf{I}(\mathbf{x})}{\partial x \partial t}v_x + \frac{\partial^2 \mathbf{I}(\mathbf{x})}{\partial y \partial t}v_y + \frac{\partial^2 \mathbf{I}(\mathbf{x})}{\partial t^2}v_t \\ &+ \frac{\partial \mathbf{I}(\mathbf{x})}{\partial x}a_x + \frac{\partial \mathbf{I}(\mathbf{x})}{\partial y}a_y + \frac{\partial \mathbf{I}(\mathbf{x})}{\partial t}a_t\end{aligned} \tag{3.6}$$

which can be written in vector notation as

$$\frac{\partial}{\partial t}(\nabla \mathbf{I}^T(x)\mathbf{v}(\mathbf{x})) = \frac{\partial \nabla \mathbf{I}^T(\mathbf{x})}{\partial t}\mathbf{v}(\mathbf{x}) + \nabla \mathbf{I}^T(\mathbf{x})\, \mathbf{a}(\mathbf{x}) \tag{3.7}$$

Using the same approach for deriving the classic 3D structure, minimizing (3.6) assuming a constant velocity model and subject to the normalization constraint $\|\mathbf{v}(\mathbf{x})\| = 1$ leads to

$$\begin{aligned}e_{ls}^F(\mathbf{x}) = &\int_{\boldsymbol{\Omega}(\mathbf{x},\mathbf{y})} \left(\frac{\partial(\nabla \mathbf{I}^T(\mathbf{y}))}{\partial t}\mathbf{v}(\mathbf{x})\right)^2 W(\mathbf{x}, \mathbf{y}) \; d\mathbf{y} \\ &+ \lambda \left(1 - \mathbf{v}(\mathbf{x})^T \mathbf{v}(\mathbf{x})\right)\end{aligned} \tag{3.8}$$

Assuming a constant velocity model in the neighborhood $\boldsymbol{\Omega}(\mathbf{x}, \mathbf{y})$, results in the acceleration experienced by the brightness pattern in the neighborhood $\boldsymbol{\Omega}(\mathbf{x}, \mathbf{y})$ to be zero at every pixel. As with its 3D structure tensor counterpart $\mathbf{J}$ in (3.4), the 3D flux tensor $\mathbf{J_F}$ using (3.8) can be written as

$$\mathbf{J_F}(\mathbf{x}, \mathbf{W}) = \int_{\boldsymbol{\Omega}} W(\mathbf{x}, \mathbf{y}) \frac{\partial}{\partial t}\nabla \mathbf{I}(\mathbf{x}) \cdot \frac{\partial}{\partial t}\nabla \mathbf{I}^{\mathbf{T}}(\mathbf{x}) d\mathbf{y} \tag{3.9}$$

and in expanded matrix form as

$$\mathbf{J_F} = \begin{bmatrix} \int_\Omega \left\{\frac{\partial^2 \mathbf{I}}{\partial x \partial t}\right\}^2 dy & \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial x \partial t}\frac{\partial^2 \mathbf{I}}{\partial y \partial t} dy & \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial x \partial t}\frac{\partial^2 \mathbf{I}}{\partial t^2} dy \\ \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial y \partial t}\frac{\partial^2 \mathbf{I}}{\partial x \partial t} dy & \int_\Omega \left\{\frac{\partial^2 \mathbf{I}}{\partial y \partial t}\right\}^2 dy & \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial y \partial t}\frac{\partial^2 \mathbf{I}}{\partial t^2} dy \\ \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial t^2}\frac{\partial^2 \mathbf{I}}{\partial x \partial t} dy & \int_\Omega \frac{\partial^2 \mathbf{I}}{\partial t^2}\frac{\partial^2 \mathbf{I}}{\partial y \partial t} dy & \int_\Omega \left\{\frac{\partial^2 \mathbf{I}}{\partial t^2}\right\}^2 dy \end{bmatrix} \tag{3.10}$$

As seen from (3.10), the elements of the flux tensor incorporate information about temporal gradient changes which leads to efficient discrimination between stationary and moving image features. Thus the trace of the flux tensor matrix, which can be compactly written and computed as

$$\mathbf{trace}(\mathbf{J_F}) = \int_\Omega \left\| \frac{\partial}{\partial t} \nabla \mathbf{I} \right\|^2 dy \tag{3.11}$$

and can be directly used to classify moving and nonmoving regions without the need for expensive eigenvalue decompositions. If motion vectors are needed, then (3.8) can be minimized to get $\hat{\mathbf{v}}(\mathbf{x})$ using

$$\mathbf{J_F}(\mathbf{x}, \mathbf{W})\ \hat{\mathbf{v}}(\mathbf{x}) = \lambda \hat{\mathbf{v}}(\mathbf{x}) \tag{3.12}$$

In this approach the eigenvectors need to be calculated at just moving feature points.

3.1.2 Flux Tensor Implementation

To detect motion blobs, only the trace of flux tensor $\mathbf{trace}(\mathbf{J_F}) = \int_{\Omega(\mathbf{y})} \left\| \frac{\partial}{\partial t} \nabla \mathbf{I} \right\|^2 dy$ needs to be computed. That requires computation of I_{xt}, I_{yt}, and I_{tt} and the integration of squares of I_{xt}, I_{yt}, I_{tt} over the area $\mathbf{\Omega}(\mathbf{y})$. The following notation is adopted for simplicity:

$$I_{xt} = \frac{\partial^2 I}{\partial x \partial t}, \quad I_{yt} = \frac{\partial^2 I}{\partial y \partial t}, \quad I_{tt} = \frac{\partial^2 I}{\partial t \partial t} \tag{3.13}$$

The calculation of the derivatives is implemented as convolutions with a filter kernel. By using separable filters, the convolutions are decomposed into a cascade of 1D convolutions. For numerical stability as well as noise reduction, a smoothing filter is applied to the dimensions that are not convolved with a derivative filter (e.g., calculation of I_{xt} requires smoothing in y-direction, and calculation of I_{yt} requires smoothing in x-direction). I_{tt} is the second derivative in temporal direction; the smoothing is applied in both spatial directions. As smoothing and derivative filters, optimized filter sets presented by Scharr et al. in [11, 12] are used.

The integration is also implemented as an averaging filter decomposed into three 1D filters. As a result, calculation of trace at each pixel location requires

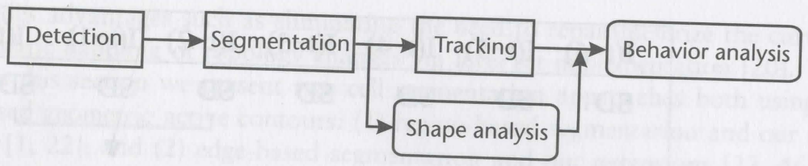


Figure 3.1 General framework for cell shape and behavior analysis.

three 1D convolutions for derivatives and three 1D convolutions for averages in the corresponding spatiotemporal cubes.

A brute-force implementation where spatial and temporal filters are applied for each pixel separately within a spatiotemporal neighborhood would be computationally very expensive since it would have to recalculate the convolutions for neighboring pixels. For an efficient implementation, the spatial (x and y) convolutions are separated from the temporal convolutions, and the 1D convolutions are applied to the whole frames one at a time. This minimizes the redundancy of computations and allows reuse of intermediate results.

The spatial convolutions required to calculate I_{xt}, I_{yt}, and I_{tt} are I_{xs}, I_{sy}, and I_{ss} where s represents the smoothing filter. Each frame of the input sequence is first convolved with two 1D filters, either a derivative filter in one direction and a smoothing filter in the other direction, or a smoothing filter in both directions. These intermediate results are stored as frames to be used in temporal convolutions, and pointers to these frames are stored in a first-in first-out (FIFO) buffer of size $n_{FIFO} = n_{Dt} + n_{At} - 1$ where n_{Dt} is the length of the temporal derivative filter

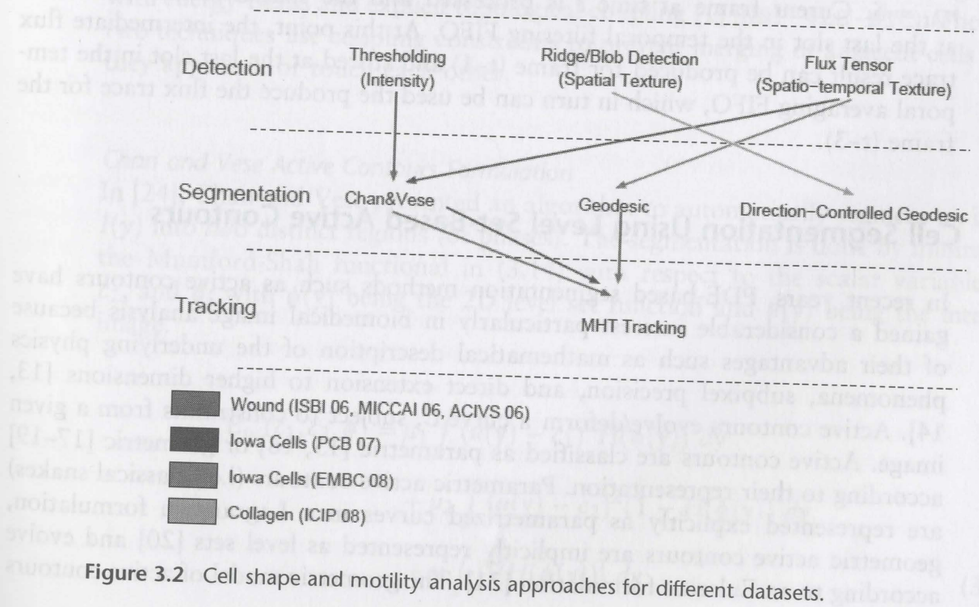


Figure 3.2 Cell shape and motility analysis approaches for different datasets.

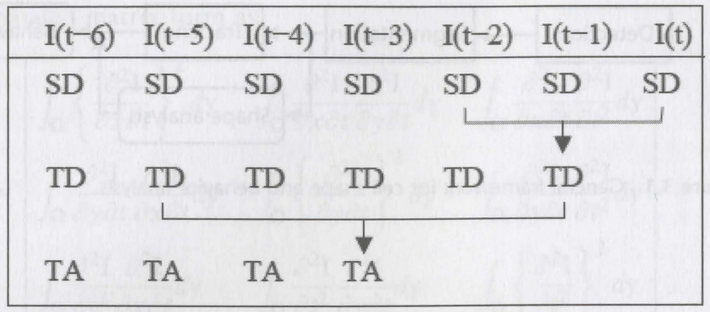


Figure 3.3 Steady state operation of the FIFO for temporal derivative filter of length $n_{Dt} = 3$ and temporal averaging filter of length $n_{At} = 5$. SD: spatial derivatives (I_{xs}, I_{sy}, I_{ss}); TD: temporal derivatives (I_{xt}, I_{yt}, I_{tt}); TA: temporal averages $(\mathbf{trace}(\mathbf{J_F}) = \int_{\Omega(\mathbf{y})} (I_{xt}^2 + I_{yt}^2 + I_{tt}^2) d\mathbf{y})$.

and n_{At} is the length of the temporal averaging filter. For each input frame, three frames I_{xs}, I_{sy}, and I_{ss} are calculated and stored. Once n_{Dt} frames are processed and stored, FIFO has enough frames for the calculation of the temporal derivatives I_{xt}, I_{yt}, and I_{tt}. Since averaging is distributive over addition, $I_{xt}^2 + I_{yt}^2 + I_{tt}^2$ is computed first and spatial averaging is applied to this result and stored in the FIFO structure to be used in the temporal part of averaging. Once flux tensor trace of n_{At} frames are computed, temporal averaging is applied. Motion mask FG_M is obtained by thresholding and post-processing averaged flux tensor trace. Post-processing include morphological operations to join fragmented objects and to fill holes.

Figure 3.3 shows the steady state operation of the FIFO for $n_{Dt} = 3$ and $n_{At} = 5$. Curent frame at time t is processed and the spatial results are stored at the last slot in the temporal filtering FIFO. At this point, the intermediate flux trace result can be produced for frame (t–1) and stored at the last slot in the temporal averaging FIFO, which in turn can be used the produce the flux trace for the frame (t–3).

3.2 Cell Segmentation Using Level Set-Based Active Contours

In recent years, PDE-based segmentation methods such as active contours have gained a considerable interest particularly in biomedical image analysis because of their advantages such as mathematical description of the underlying physics phenomena, subpixel precision, and direct extension to higher dimensions [13, 14]. Active contours evolve/deform a curve C, subject to constraints from a given image. Active contours are classified as parametric [15, 16] or geometric [17–19] according to their representation. Parametric active contours (i.e., classical snakes) are represented explicitly as parametrized curves in a Lagrangian formulation, geometric active contours are implicitly represented as level sets [20] and evolve according to an Eulerian formulation [21]. The geometric model of active contours

provides advantages such as eliminating the need to reparameterize the curve and automatic handling of topology changes via level set implementation [20].

In this section we present two cell segmentation approaches both using level setbased geometric active contours: (1) region-based segmentation and our extensions [1, 22]; and (2) edge-based segmentation and our extensions [23, 46]. The first approach is used for images where cells or nuclei appear as semi-homogeneous blobs with vague borders. The second approach is used for images where the interior of the cells appears highly heterogeneous or where interior and exterior intensity distributions are similar. The type of the cells, environmental conditions, and imaging characteristics such as zoom factor affect the appearance of cells, and thus choice of the approach.

3.2.1 Region-Based Active Contour Cell Segmentation

Segmentation of multiple cells imaged with a phase contrast microscope is a challenging problem due to difficulties in segmenting dense clusters of cells. As cells being imaged have vague borders, close neighboring cells may appear to merge. Accurate segmentation of individual cells is a crucial step in cell behavior analysis since both over-segmentation (fragmentation) and under-segmentation (cell clumping) produce tracking errors (i.e., spurious or missing trajectories and incorrect split and merge events).

In this section, we describe three different region-based level set segmentation methods and their performance for separating closely adjacent and touching cells in a dense population of migrating cells. The three techniques described in this section are all based on the "active contour without edges" energy functional [24] with appropriate extensions. These include a two-phase Chan and Vese algorithm [24] (`CV-2LS`), an N–level set algorithm with energy-based coupling by Zhang et al. [25] (`ZZM-NLS`), and an improved version of of our four-color level set algorithm with energy-based and explicit topological coupling [1] (`NBP-4LS-ETC`). The latter two techniques use coupling constraints to prevent merging of adjacent cells when they approach or touch each other.

Chan and Vese Active Contours Formulation

In [24], Chan and Vese presented an algorithm to automatically segment an image $I(\mathbf{y})$ into *two* distinct regions (or phases). The segmentation is done by minimizing the Mumford-Shah functional in (3.14) with respect to the scalar variables c_1, c_2, and ϕ; with $\phi(\mathbf{y})$ being the 2D level set function and $u(\mathbf{y})$ being the intensity image.

$$\begin{aligned} E_{pc}(c_1, c_2, \phi) = \mu_1 \int (u(\mathbf{y}) - c_1)^2 H(\phi(\mathbf{y}))\, d\mathbf{y} \\ + \mu_2 \int (u(\mathbf{y}) - c_2)^2 (1 - H(\phi(\mathbf{y}))\, d\mathbf{y} \\ + \nu \int |\nabla H(\phi(\mathbf{y}))|\, d\mathbf{y} \end{aligned} \tag{3.14}$$

In (3.14), c_1 and c_2 model the gray values of the two phases used to define the segmentation, with the boundary for the phases described by regularized Heaviside function $H(\phi)$ [24]. ϕ models the interface separating these two phases. The first two terms in the functional aim at maximizing the gray value homogeneity of the two phases, while the last term aims at minimizing the length of the boundary separating these two phases; μ_1, μ_2, and ν are adjustable constants associated with this functional. The Euler-Lagrange equation $\frac{\partial \phi}{\partial t}$, to evolve the level set curve, is

$$\frac{\partial \phi}{\partial t} = \delta(\phi)\left[\nu \ \mathrm{div}\frac{\nabla \phi}{|\nabla \phi|} - \mu_1(u(\mathbf{x}) - c_1)^2 + \mu_2(u(\mathbf{x}) - c_2)^2\right] \tag{3.15}$$

The scalars c_1 and c_2 are updated at each level set iteration step using (3.16) for $\phi(\mathbf{y}) > 0$ and $\phi(\mathbf{y}) < 0$, respectively.

$$c_1 = \frac{\int u(\mathbf{y})H(\phi(\mathbf{y}))\ d\mathbf{y}}{\int H(\phi(\mathbf{y}))\ d\mathbf{y}},\ c_2 = \frac{\int u(\mathbf{y})(1 - H(\phi(\mathbf{y}))\ d\mathbf{y}}{\int (1 - H(\phi(\mathbf{y}))\ d\mathbf{y}} \tag{3.16}$$

A regularized Heaviside function $H_{2,\epsilon}(\phi(\mathbf{y}))$ is used to obtain a numerically stable Dirac delta function $\delta(\phi)$ in (3.15) as described in [24]. A suitable stopping criterion (such as a small change in energy, $|\phi_{i+1} - \phi_i| < k_{thresh}$ where ϕ_{i+1}, ϕ_i are the level set functions at successive iterations) is used to terminate the curve evolution. In [26], in order to segment multiple (i.e., N distinct) objects, Chan and Vese extended their previous two-phase algorithm [24] by using $\lceil \log_2 N \rceil$ level sets.

Coupled N–Level Set Formulation

As observed by Zhang et al. [25], Dufour et al. [27] and Zimmer and Olivo-Marin [28], the two variants of the Chan and Vese algorithm are unsuitable for reliable cell segmentation, since they do not prevent apparent merges in cells. The $\lceil \log_2 N \rceil$ level set formulation improves on the performance of a single level set function, as more number of objects with varying intensities can be efficiently classified. However, it does not prevent under-segmentation (i.e., incorrect merges), when the objects have similar intensities (i.e., cells).

To overcome the drawbacks of classical Chan and Vese level set formulations, while at the same time solving the problem of apparent merging of cells during tracking, Zhang et al. [25] proposed a new model for segmenting cells, using N–level sets. Here, N is the number of cells at a given time instance. An a priori knowledge that cells do not merge during the evolution process was used to guide the segmentation process. This was achieved by a pair-wise energy-based coupling constraint on the level sets evolution process. A similar formulation was used by Dufor et al. in 3D cell segmentation [27]. The energy functional, $E_{nls}(\boldsymbol{c_{in}}, c_{out}, \Phi)$, used to evolve N– coupled level sets is shown below [25]:

$$\begin{aligned} E_{nls}\ (\boldsymbol{c_{in}}, c_{out}, \Phi) &= \mu_{in} \sum_{i=1}^{N} \int_{\Omega} (I - c_{in}^{i})^2 H(\phi_i)\ d\mathbf{y} \\ &+ \mu_{out} \int_{\Omega} (1 - c_{out})^2 \prod_{\substack{i=1 \\ \forall\ i:H(\phi_i)<0}}^{N} (1 - H(\phi_i))\ d\mathbf{y} + \nu \sum_{i=1}^{N} \int_{\Omega} |\nabla H(\phi_i)| d\mathbf{y} \\ &+ \gamma \sum_{i=1}^{N} \sum_{j=i+1}^{N} \int_{\Omega} H(\phi_i) H(\phi_j)\ d\mathbf{y} \end{aligned} \tag{3.17}$$

Here, $\Phi = \{\phi_1, \phi_2, \ldots, \phi_N\}$ represents N–level sets associated with N cells in the image; $\boldsymbol{c_{in}}$ represents the average intensities of cells for $\phi_i \geq 0$, while c_{out} is the average intensity of the background.[1] The first and second terms are homogeneity measures of the foregrounds and background of all level sets. The third term controls the lengths of interfaces $\phi_i = 0$, and minimizes the length of all level sets, while the fourth term of the functional penalizes pair-wise couplings or overlaps between level sets. The constants μ_{in}, μ_{out}, ν, and γ are weights associated with each of the terms.

Coupled 4-Level Set Formulation

The coupled N-level set formulation of Zhang et al. is successful in preventing apparent/false merges. But the $O(N^2)$ complexity of this approach where N is number of cells, makes it impractical for high-throughput screening studies, where each frame in the image sequence may contain hundreds to thousands of cells. Evolving thousands of level sets by solving thousands of separate partial differential equations with millions of coupling interaction terms is currently impractical or too expensive even using specialized hardware or parallel processing computing clusters. In [1], we presented an optimized version of the N–level set algorithm that scales to high number of cells/objects and demonstrated its performance in presence of an external tracking module in [22]. An overview of this optimized version is given below. And major implementation steps are presented in Algorithm 2.

Let N be the number of objects (i.e., cells) in an image. For any given object A_i, let $\mathcal{N}(A_i) = \{A_j, A_{j+1}, \ldots, A_L\}$, $L << N$ represent its L neighbors. Our optimization is based on the fact that: *during cell evolution, false merging in A_i can only occur within this local neighborhood of L objects*, and not over the entire image. Thus, to prevent apparent/false merges, it is sufficient to assign each object A_i to a level set different than the level sets of its neighbors. When the spatial layout of the cells in a frame is described using a planar neighborhood graph $\mathbf{P}$ where the vertices A_i^V represent the cells and the edges A_i^E represent the neighborhood relationship. The problem of assigning level sets to cells becomes equivalent to *graph vertex coloring*.

- *Graph vertex coloring:* assignment of labels or colors to each vertex of a graph such that no edge connects two identically colored vertices.

1. The region *exterior to all level sets* indicates the background.

Now the question becomes, *What is the minimum number of colors, thus the minimum number of level sets needed?* The *Four-Color Theorem* (FCT), states that every planar graph is four-colorable while insuring that neighboring vertices do not share the same color [29–31]. Since our neighborhood graph is a planar graph, the N–level set formulation of Zhang et al. [25] can be effectively reduced to a 4-level set formulation. Assignment of N objects to $k << N$ level sets (k = 4 for our case) is justified by the assumption that objects (in our case cells) have similar, or nearly similar characteristic features. In this case objects/cells can be randomly assigned to any of the four level sets. For a wide variety of applications (including our application of cell segmentation), the upper limit of the chromatic number for planar graphs is four. However, we do acknowledge that there may exist other planar graphs having even smaller chromatic numbers.

In order to evolve the four level sets, we propose minimizing the N–coupled level sets energy functional shown in (3.17), with $N = 4$. In addition, we replace the length minimizer term [third term in (3.17)] with its geodesic equivalent, and incorporate additional constraints in our proposed energy functional $E_{fls}(\boldsymbol{c_{in}}, c_{out}, \Phi)$ as shown below:

$$E_{fls}(\boldsymbol{c_{in}}, c_{out}, \Phi) = \underbrace{\mu_{in} \sum_{i=1}^{4} \int_{\Omega} (I - c_{in}^{i})^2 H(\phi_i)\, d\boldsymbol{y}}_{\text{Homogeneity of Foregrounds}}$$
$$\underbrace{+\mu_{out} \int_{\Omega} (I - c_{out})^2 \prod_{\substack{i=1 \\ \forall\, i:H(\phi_i)<0}}^{4} (1 - H(\phi_i))\, d\boldsymbol{y}}_{\text{Homogeneity of Background}} + \underbrace{\nu \sum_{i=1}^{4} \int_{\Omega} |g(I)\nabla H(\phi_i)|\, d\boldsymbol{y}}_{\text{Geodesic Length Minimizers}}$$
$$\underbrace{+\gamma \sum_{i=1}^{4} \sum_{j=i+1}^{4} \int_{\Omega} H(\phi_i) H(\phi_j)\, d\boldsymbol{y}}_{\text{Energy-based Coupling}} + \underbrace{\eta \left\{ \sum_{i=1}^{4} \int_{\Omega} \frac{1}{2} (|\nabla \phi_i| - 1)^2\, d\boldsymbol{y} \right\}}_{\text{Implicit Redistancing}} \tag{3.18}$$

where $g(I)$ is an edge indicator function given by [32]

$$g(I) = \frac{\alpha}{1 + \beta |\nabla G_\sigma \circledast I|}$$

with constants α, β. $|\nabla G_\sigma \circledast I|$ can also be replaced with a regularized version of our adaptive robust structure tensor (ARST) [33]. The last term in (3.18) enforces the constraint of $|\nabla \phi_i| = 1$, thus helping us avoid explicit redistancing of level sets during the evolution process [34], with η being a constant. As noted by Kim and Lim, addition of a geodesic length term improves the performance of an energy-only Chan and Vese level set formulation by preventing detection of regions with weak edge strengths in [35]. This property was also exploited by Dufour et al. in 3D cell segmentation [27].

Gradient-descent minimization can be used to iteratively solve the level set functions ϕ_i as

$$\phi_i(k+1) = \phi_i(k) + \Delta\tau \frac{\partial \phi_i(k)}{\partial k} \tag{3.19}$$

where k is an artificial evolution time, $\Delta\tau$ is the spacing between successive iterations, and $\frac{\partial \phi_i(k)}{\partial k}$ is the Euler-Lagrange equation associated with the ith level set ϕ_i, given by

$$\frac{\partial \phi_i(k)}{\partial k} = \delta(\phi_i) \left\{ -\mu_{in}(I - c_{in}^{i})^2 + \mu_{out}(I - c_{out})^2 \prod_{\substack{j=1 \\ \forall\, j:H(\phi_j)<0, j\neq i}}^{4} (1 - H(\phi_j)) \right.$$
$$\left. + \nu \left(g(I) \cdot \operatorname{div}\left(\frac{\nabla \phi_i}{|\nabla \phi_i|} \right) + \nabla g(I) \cdot \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \gamma \sum_{j=1; j\neq i}^{4} H(\phi_j) \right\}$$
$$+ \eta \left\{ \nabla^2 \phi_i - \operatorname{div}\left(\frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \right\} \tag{3.20}$$

For numerical stability of Euler-Lagrange equations, the Heaviside functions (and in turn, the Dirac-delta functions) are regularized by a parameter $\epsilon > 0$ [24].

Accurate Neighborhoods Using Generalized Voronoi Diagrams

The success of apparent/false merge prevention in the 4-level sets approach depends on the accuracy of the neighborhood adjacency graph $\mathcal{N}(\mathbf{P})$. In our original formulation [1], we reported Delaunay triangulations as a means to obtain $\mathcal{N}(\mathbf{P})$. The Delaunay triangulation approach has been previously used in many biological image analysis applications, such as in quantitative spatial analysis of complex cellular systems [36], membrane growth analysis [37], histological grading of cervical intraepithelial neoplasia [38], segmentation of tissue architecture [39], diagnosis of malignant mesothelioma [40], bladder carcinoma grading [41], and characterization of muscle tissue [42].

Delaunay triangulation graphs obtained using distances between centroids of objects is adequate to represent the neighborhood relationships of circular objects of similar sizes. But since centroid-based distance measures are insensitive to shape, size, and orientation of objects, graphs derived from these measures *may* lead to incorrect neighborhood relationships for objects of irregular shapes and sizes, and ultimately to the failure of the 4-level sets segmentation (see Figure 3.4). Hence, rather than using centroid-based distance measures [i.e., Delaunay triangulation, and its geometric dual, the ordinary Voronoi diagram (OVD)], we propose using the *generalized Voronoi diagram* (GVD) that takes into account shape, size, and orientation of objects to compute accurate neighborhood relationships. The incorrect neighborhood relationships and subsequent failure of the 4-level sets segmentation illustrated in Figure 3.4 can then be avoided by extracting correct neighborhood relationships using generalized Voronoi boundaries as shown in Figure 3.5.

The GVD in any dimension can be precisely defined using point-to-object distance measures [43, p. 280]. Let $\mathbf{A} = \{A_1, A_2, \ldots, A_N\}$ be a set of arbitrarily shaped objects in a d–dimensional space $\mathbb{R}^d$. Now, for any point $\mathbf{p} \in \mathbb{R}^d$, let $\mathbf{D}(\mathbf{p}, A_i)$ denote a distance measure representing how far the point $\mathbf{p}$ is from the object A_i,

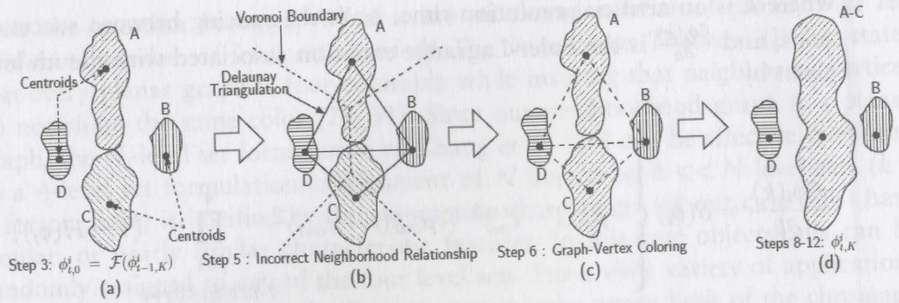


Figure 3.4 (a–d) Incorrect neighborhood relationships, resulting from Delaunay triangulation and its effects on Algorithm 2. Since the neighborhood of the cells A and C is not detected, both cells are assigned the same color, and hence to the same level set. This results in a *false-merge* taking place between cells A and C, at the end of the iteration process.

which is typically the minimum distance from $\mathbf{p}$ to any point in object A_i. The dominance region (also known as influence-zone) of A_i, is then defined as

$$\mathrm{Dom}(A_i, A_j) = \{\mathbf{p} | \mathrm{D}(\mathbf{p}, A_i) \leq \mathrm{D}(\mathbf{p}, A_j), \forall j, j \neq i\} \tag{3.21}$$

A generalized Voronoi boundary, between A_i and Aj, can then be defined as the loci of equidistant points between both objects, $\mathcal{L}(A_i, A_j)$, as

$$\mathcal{L}(A_i, A_j) = \{\mathbf{p} | \mathrm{D}(\mathbf{p}, A_i) = \mathrm{D}(\mathbf{p}, A_j)\} \tag{3.22}$$

with a corresponding influence zone, or Voronoi region, for A_i, $V(A_i)$, as

$$V(A_i) = \bigcap_{i \neq j} \mathrm{Dom}(A_i, A_j) \tag{3.23}$$

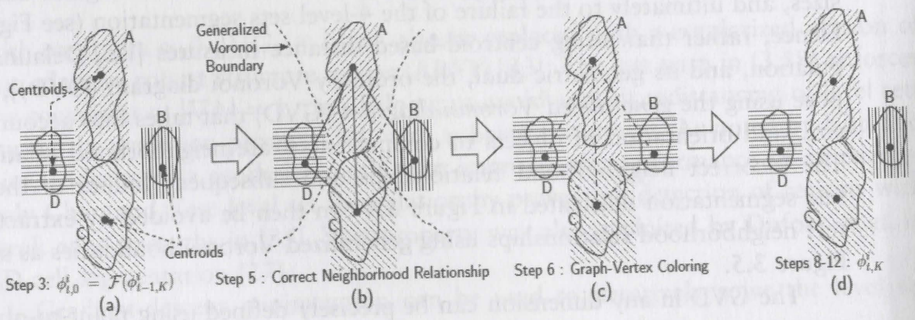


Figure 3.5 (a–d) The *false-merge* problem, shown in Figure 3.4, can be avoided in Algorithm 2 by extracting correct neighborhood relationships using generalized Voronoi boundaries. Cells A and C are correctly identified as neighbors, and hence assigned to different level sets.

Hence, the generalized Voronoi diagram of $\mathbf{A}$, GVD($\mathbf{A}$) is given by the collection of such Voronoi regions, as

$$\begin{aligned} \mathrm{GVD}(\mathbf{A}) &= \bigcup_i V(A_i) \\ &= \bigcup_i \bigcap_{i \neq j} \{\mathbf{p} | \mathrm{D}(\mathbf{p}, A_i) \leq \mathrm{D}(\mathbf{p}, A_j), \forall j, j \neq i\} \end{aligned} \tag{3.24}$$

When $\mathbf{A}$ is a collection of points rather than sized objects, the GVD($\mathbf{A}$) reduces to an ordinary Voronoi diagram, OVD($\mathbf{A}$) (the Delaunay triangulation being the geometric dual of the OVD).

The GVD representation of a set of objects has a number of useful properties: (1) it is a thin set that partitions a space into connected regions; (2) it is homotopic to the number of objects; (3) it is invariant to global transformations applied to all objects in the image; and (4) each region is guaranteed to contain the entire object. For those interested in properties of the OVD for point objects, we direct them to the book by Okabe et al. [43] and the classic survey paper by Aurenhammer [44].

In [45] we present a new algorithm for efficiently and accurately extracting accurate neighborhood graphs in linear time by computing the Hamilton-Jacobi GVD using the exact Euclidean-distance transform with Laplacian-of-Gaussian, and morphological operators. We refer reader to [45] for further information and validation results using both synthetic and real biological imagery. In Section 3.3.3, we present a different GVD construction algorithm designed for implementation on graphical processing units (GPUs).

Spatiotemporal interactions between two neighboring cells can be grouped into four, depending on their spatial configurations (merged into a single connected component or separate components) and temporal configurations (one-to-one overlap, where each cell in the current frame overlaps only with a single cell in the previous frame; or temporally merged, where a single cell in the current frame overlaps with multiple cells in the previous frame). These cases are summarized in Table 3.1 and illustrated in Figure 3.6.

Figure 3.7 shows a sample synthetic case-2 sequence, its 4-level sets segmentation, and corresponding tracking results. Even though the two circles touch/merge for a while (Figure 3.6), the coupled 4-level sets segmentation succeeds in segmenting them as two objects.

Table 3.1 Spatiotemporal Interactions Between Two Neighboring Cells

Case	*Spatial*	*Temporal*	*N-LS*	*N-LS +coupling*	*Comments*
1	Separate	1-to-1 overlap	✓	✓	
2	Merged	1-to-1 overlap	×	✓	
3	Separate	Merged	×	✓×	Depending on temporal sampling rate
4	Merged	Merged	×	✓×	Depending on temporal sampling rate

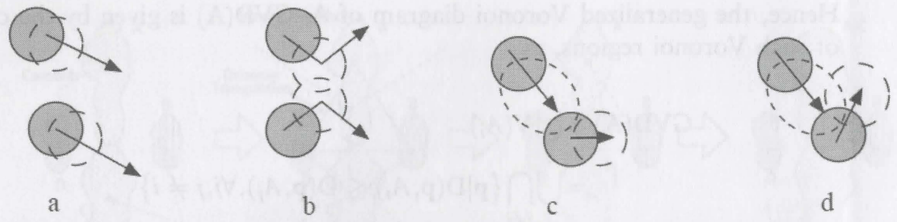


Figure 3.6 Cell-to-cell interaction cases: (a) case 1: separate cells, one-to-one temporal overlap; (b) case 2: merged cells one-to-one temporal overlap; (c) case 3: spatially separate and temporally merged; and (d) case 4: spatially merged and temporally merged.

3.2.2 Edge-Based Active Contour Cell Segmentation

While region-based active contours have been succesfully used in many cell and nucleus segmentation applications [1–3, 24], they require a bimodal image model to discern background and foreground which is not applicable in some high-resolution cell images. Figure 3.8 shows typical high-resolution phase contrast images of cells, where white regions surrounding the cells are the phase halos. A straightforward implementation of [24] converges to the boundaries between phase halo and the rest of the image. For images where intensity inside the cell boundaries is highly heterogeneous, and there is no clear difference between inside and outside intensity profiles, edge-based approaches are used since region-based active contour approaches are not applicable.

In this section, we describe two edge-based level set segmentation methods and their performance for handling phase halos. The first technique is the classical geodesic active contour approach [19]; the second technique is our extension directed edge profile-guided level set active contours. Compared to region-based

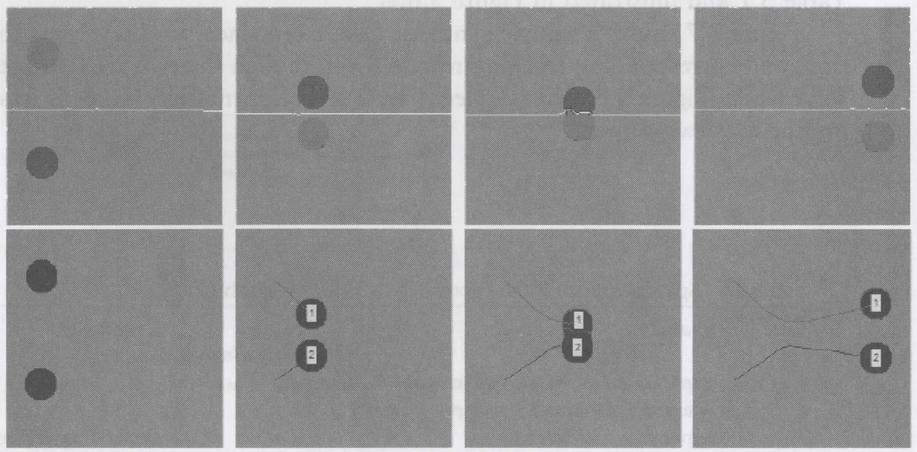


Figure 3.7 Cell-to-cell interaction case 2 test sequence frames #1, 8, 13, 20. Row 1: four color level set segmentation; Row 2: tracking results.

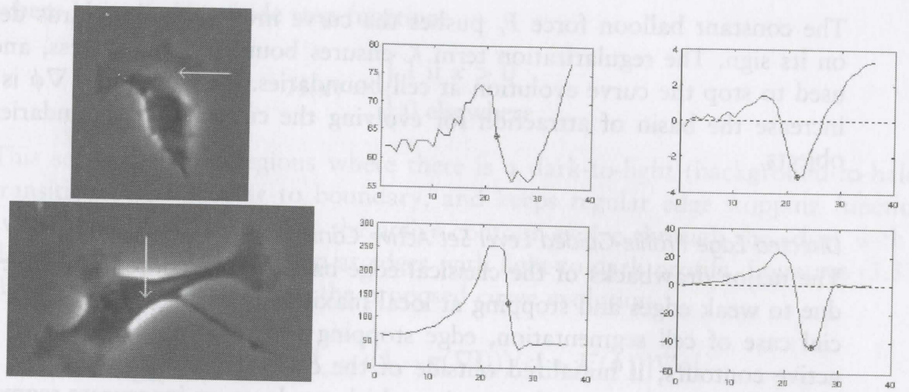


Figure 3.8 Phase contrast cell images. Left to right: sample cell, intensity profile at the marked direction, and derivative of the directed profile. The actual cell boundary point is marked on the graphs.

active contours, edge-based active contours are more sensitive to initialization. Since they are designed to stop at edges, they suffer from: (1) early stopping on background or foreground edges, (2) contour leaking across weak boundaries, and (3) for phase contrast microscopy imagery, early stopping at outer edges of phase halos. The first problem can be avoided by starting the contour evolution close to the object/cell boundaries. Initial contour can be obtained from a detection step using spatial properties as in ridge-based approach in [23], or spatiotemporal properties as in flux tensor-based approach in [46–48] described in Section 3.2. The latter two problems are addressed in our directed edge profile-guided level set active contours below.

Geodesic Level Set Active Contours

In level set based active contour methods, a curve $\mathcal{C}$ is represented implicitly via a Lipschitz function ϕ by $\mathcal{C} = \{(x,y)|\phi(x,y) = 0\}$, and the evolution of the curve is given by the zero-level curve of the function $\phi(t,x,y)$ [24]. In regular geodesic active contours [19] the level set function ϕ is evolved using the speed function,

$$\frac{\partial \phi}{\partial t} = g(\nabla \mathbf{I})(F_c + \mathcal{K}(\phi))|\nabla \phi| + \nabla \phi \cdot \nabla g(\nabla \mathbf{I}) \tag{3.25}$$

where F_c is a constant, $\mathcal{K}$ is the curvature term (3.26), and $g(\nabla \mathbf{I})$ is the edge stopping function, a decreasing function of the image gradient which can be defined as in (3.27).

$$\mathcal{K} = div\left(\frac{\nabla \phi}{|\nabla \phi|}\right) = \frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{\frac{3}{2}}} \tag{3.26}$$

$$g(\nabla \mathbf{I}) = exp(-|\nabla G_\sigma(x,y) * \mathbf{I}(x,y)|) \tag{3.27}$$

The constant balloon force F_c pushes the curve inwards or outwards depending on its sign. The regularization term $\mathcal{K}$ ensures boundary smoothness, and $g(\mathbf{I})$ is used to stop the curve evolution at cell boundaries. The term $\nabla g \cdot \nabla \phi$ is used to increase the basin of attraction for evolving the curve to the boundaries of the objects.

Directed Edge Profile-Guided Level Set Active Contours

The major drawbacks of the classical edge-based active contours are the leakage due to weak edges and stopping at local maximums in noisy images. For the special case of cell segmentation, edge stopping functions used in regular geodesic active contours, if initialized outside of the cells, respond to the outer edges of phase halos and cause early stopping which produces an inaccurate segmentation. When the curve is initialized inside the cells, the texture inside the cell also causes premature stopping. Often phase halo is compensated by normalizing the image, but this weakens the cell boundaries further, especially in the areas where cells are flattened, and leads to leakage of the curve. One remedy is to enforce a shape model, but it fails for cells with highly irregular shapes. Nucleus-based initialization and segmentation [49] is also not applicable to images such as in Figure 3.8. To obtain an accurate segmentation, we propose an approach that exploits the phase halo effect instead of compensating it. The intensity profile perpendicular to the local cell boundary is similar along the whole boundary of a cell; it passes from the brighter phase halo to the darker cell boundary. We propose to initialize the curve outside the cell and phase halo, and guide the active contour evolution based on the desired edge profiles which effectively lets the curve evolve through the halos and stop at the actual boundaries. The existence of phase halo increases the edge strength at cell boundaries. We propose the edge stopping function g_d (3.30) guided by the directed edge profile, that lets the curve evolve through the outer halo edge and stop at the actual boundary edge. As shown in Figure 3.8, if initialized close to the actual boundary, the first light-to-dark edge encountered in the local perpendicular direction corresponds to the actual boundary. By choosing this profile as the stopping criterion, we avoid the outer edge of phase halo. This stopping function is obtained as follows. Normal vector $\vec{N}$ to the evolving contour/surface can be determined directly from the level set function:

$$\vec{N} = -\frac{\nabla \phi}{|\nabla \phi|} \tag{3.28}$$

Edge profile is obtained as the intensity derivative in opposite direction to the normal:

$$\mathbf{I}_{-\vec{N}} = \frac{\nabla \phi}{|\nabla \phi|} \cdot \nabla \mathbf{I} \tag{3.29}$$

Dark-to-light transitions produce positive response in $\mathbf{I}_{-\vec{N}}$. We define the edge profile-guided edge stopping function g_d as

$$g_d(\nabla \mathbf{I}) = 1 - H(-I_{-\vec{N}})(1 - g(\nabla \mathbf{I})) \tag{3.30}$$

where H is the Heaviside step function:

$$H(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ elsewhere} \end{cases}$$

This sets g_d to 1 at regions where there is a dark-to-light (background-to-halo) transition perpendicular to boundary, and keeps regular edge stopping function everywhere else. Thus it lets the active contour evolve through the edges with a dark-to-light profile and stop at edges with light-to-dark profile. Equation (3.31) shows the speed function of the proposed curve evolution.

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= (1 - \mathrm{H}(-\mathbf{I}_{-\vec{N}})(1 - g(\nabla \mathbf{I})))(c\mathbf{I} + \mathcal{K}(\phi))|\nabla \phi| \\ &\quad + \nabla \phi \cdot \nabla(1 - \mathrm{H}(-\mathbf{I}_{-\vec{N}})(1 - g(\nabla \mathbf{I}))) \end{aligned} \tag{3.31}$$

To prevent contour leakage on weak edges, we use a spatially adaptive force that slows down the curve evolution at boundaries. We replace the constant balloon force F_c in (3.25) with an intensity adaptive force F_A:

$$F_A(x, y) = c\mathbf{I}(x, y) \tag{3.32}$$

This intensity adaptive force increases the speed of contraction on bright phase halos and reduces it on thin dark extensions (pseudo-pods) and prevents leaking across weak edges.

3.2.3 GPU Implementation of Level Sets

Scientific Computing on Graphical Processing Units

The power of GPUs that can currently sustain hundreds of gigaflops is increasingly being leveraged for scientific computing. The exponential increase in GPU computing power is being driven by the commercially competitive and hardware intensive gaming market. The data parallel GPU architecture is emerging as a powerful environment for running many closely coupled but independent parallel threads. Both NVIDIA and AMD/ATI are expanding their market beyond entertainment and targeting dual use applications for their single chip multiple core processors with more than a billion transistors from the initial chip-design phase itself; this is often referred to as general purpose GPU (GPGPU) [50]. Early programs developed for GPUs had to use graphical APIs such as OpenGL or NVIDIA's Cg to control the hardware, but current general purpose APIs such as NVIDIA's CUDA, ATI's CTM, and Stanford's Brook for scientific computing [51], which avoid explicit graphics interfaces, are now commonly used for compute intensive tasks.

The GPU architecture is a SIMD pipeline for ingesting vertex and primitive shape information to construct filled, textured, and lit polygons for viewing on the screen. Initially, sets of processors called shader units were specialized for each stage, but with the advent of the Shader Model 4.0 specification the shader processor architecture has been generalized. This change maximizes the utilization of every available processor and reduces the economic cost of developing dedicated units. Three stages of the GPU pipeline are accessible via programming: vertex processing, geometry construction, and pixel shading. During a rendering pass,

a complete set of geometry primitives passes through all these stages. Scientific applications are mapped to the architecture by loading data onto the GPU and performing multiple rendering passes in which different programs can be loaded to each shader stage for each pass.

The pixel (fragment) shader provides the most straightforward parallel processing mechanism for image processing and computer vision problems involving dense arrays of data. Take for example the task of applying a convolution kernel to an image. The image is loaded into texture memory of the GPU and a program is loaded which computes the convolution sum for a single pixel. A rendering target the same $n \times m$ size as the source image is set up to hold the results. On the GPU the rendering target is broken into $n \times m$ pixels which are each processed in parallel. The program takes the source image and the pixel's x/y location as parameters, computes the convolution, and writes the value to the pixel. The result is a filtered image. Additional filter passes are additional rendering passes using the previously generated output as the new input for a different shader program. The limitation of the pixel shader is the support of gather but not scatter, which means the shader can read values of its neighbors from previous time steps or source images but is limited to writing only to the value of the pixel it is currently processing. This makes certain tasks which require pixel level coherence such as connected component labeling less suitable.

Building Blocks for GPU Programs

Our implementation makes use of the pixel shader through the NVIDIA Cg API and a discussion of some of some useful techniques for formatting and handling data is merited. The GPU has rapid and convenient access to a large pool of (texture) memory, but the transfer of data to and from the CPU need to be explicitly coded. During computation a thread has limited access to memory, including (fragment) registers, some shared memory between threads, and a global memory without collision rules that allows general scatter/gather (read/write) memory operations. The data parallel algorithmic building blocks for GPU programming include map, scatter and gather, parallel reduce, parallel prefix scan, sort and search. Map applies a specified function, for example, local average or convolution operator, to all the elements in an array or stream. In CUDA the kernels specify the map or function which are applied in parallel by all executing threads. Another common GPU program building block is the split operation when many threads need to partition data as in sorting or building trees and the compact operator which can be applied to remove null elements from an array. Each thread can allocate a variable amount of storage per thread which is useful in marching cubes isosurface extraction algorithms and geometry creation. Since the scan operator can be used as the basis for implementing many of the other building blocks, we describe it in more detail.

The scan or parallel prefix sum operator is an efficient multithreaded way to compute partial results to build up the desired output based on all the values in the input stream. Given the array $A = [a_0, a_1, , a_{(n-1)}]$ and a binary associative operator $\oplus$ with identity element I, then, the *exclusive* scan produces the output stream [52],

$$scan(A) = [I, a_0, (a_0 \oplus a_1), , (a_0 \oplus a_1 \oplus \ \oplus a_{n-2})] \tag{3.33}$$

For example, if $\oplus$ is addition, then applying scan to the set $A = [9,0,2,1,8,3,3,4]$, returns the set $scan(A) = [0, 9, 9, 11, 12, 20, 23, 26]$. Other common binary functions include min, max, logical AND, and logical OR. The scan operator turns out to be a surprisingly fundamental building block that is useful for a variety of parallel algorithms including histograms, run-length encoding, box-filtering (summed area tables), cross-correlation, distance transform, matrix operations, string comparison, polynomial evaluation, solving recurrences, tree operations, radix sort, quicksort, and lexical analysis. An extremely efficient GPU-based unsegmented scan algorithm implementation that uses $O(n)$ space to accomplish $O(N)$ work is given by Harris et al. [53] that was further extended to segmented scans in [52]. Lectures and resource materials on GPU programming can be found at [54, 55].

Data Structures. The GPU memory model is grid or array based for textures rather than linear, and data can be mapped accordingly for maximum efficiency. One-dimensional arrays are limited by the maximum size of a texture (8,192 elements for the NVIDIA GeForce G80 processor) and also suffer from slower lookup times in GPU memory due to the optimization for two-dimensional textures via caching. Large one-dimensional arrays can be packed into two-dimensional textures and the shader program can perform the address translation to access the data correctly. Two-dimensional floating point or integer data arrays map exactly onto textures with the exception of those that exceed the maximum allowable size limit. At that point the array can be split amongst multiple textures and correct indexing is managed by setting multiple rendering targets. Three-dimensional (or higher) arrays pose a difficulty since currently GPUs can read the data as a texture but the GPU cannot directly generate $x/y/z$ pixels for the pixel shader to operate upon. Lefohn et al. [56] suggested three solutions: (1) use a 3D texture but only operate on a single texture at a time; (2) store each slice as a separate texture in graphics memory with the CPU activating the appropriate slice; or (3) place all data slices into a single 2D texture allowing for operation on the entire 3D array in a single pass.

Framebuffer Objects. A typical rendering pass sends its results to the framebuffer for display on the screen. Framebuffer Object (FBO) extensions were added to the OpenGL specification to enable more efficient rendering directly to texture memory rather than copying information from the framebuffer. The FBO acts like a pointer to allocated areas of texture memory. A single FBO can be used to manage multiple textures called attachments, and the number of attachments is dependent on the GPU. Prior to executing a shader pass, an FBO attachment is bound, which switches rendering from the screen framebuffer to texture memory. After a rendering pass is completed, the result can be used as input for another shader for further processing.

Parallel Reduction. The pixel shader does not have a mechanism for accumulating a result like a sum or product of elements, or other binary associative operations like minimum or maximum value over an array (texture). A solution is to write the texture to CPU memory and iterate over the array of values, but this

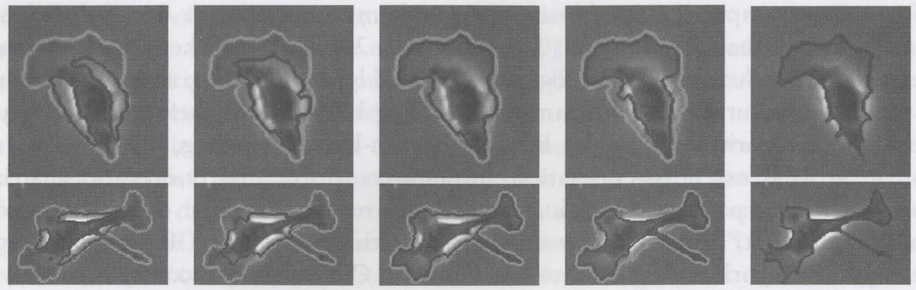


Figure 3.9 Magnified results on sample cells from BAEC dataset. From left to right: Chan and Vese, GAC, GAC with adaptive force, proposed method, and manual ground truth.

is an expensive operation. Using parallel reduction this kind of operation can be performed efficiently on the GPU [57]. The original texture is passed to a shader program parameter and the target rendering texture area is half its size. Every neighborhood of four texels is considered and evaluated for operation of interest. As shown in Figure 3.10 the maximum value of every four texels is written to the target texture. The result is passed back to the shader and the target render area reduced by half again. This operation is performed log N times where N is the width of the original texture and the result is the single maximum value among the texels. A tree-based parallel reduction using the ping-pong method of texture memory access is bandwidth bound. In CUDA, shared memory is used to reduce the bandwidth requirement and with sequential (instead of interleaved) addressing is conflict free. Parallel reduction of N elements requires $\log(N)$ steps that perform the same number of $O(N)$ operations as a sequential algorithm. With P processing cores (P physically parallel threads) the total time complexity is reduced to $O(N/P + \log N)$ compared to $O(N)$ for sequential reduction.

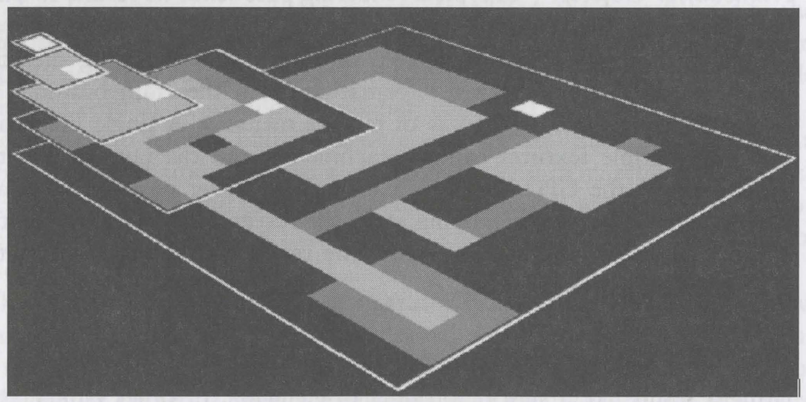


Figure 3.10 An example of using reduction to find the maximum value n the texture. The lightest shaded texel from each set of four neighboring texels is kept until the single white texel is isolated.

Level Sets on GPU

Overview. GPU-based implementation of level set algorithms is an active area of research due to the potential for scalable performance with large datasets or real-time requirements [50, 58–60]. Our GPU level set implementation is based on the Chan and Vese 3.34 formulation with geodesic terms as a series of pixel shader programs based on update equation (3.34).

$$\phi^{t+1} = \phi^t + \partial t \delta_\epsilon(\phi)\left[\mu\nabla \cdot g\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - g\nu - \lambda_1(u_o - c_1)^2 + \lambda_2(u_o - c_2)^2\right] \quad (3.34)$$

Each term is computed as a series of pixel shader programs using FBOs to store intermediate results. For the initialization step the source images are loaded into texture memory using *glTexImage2D* and the signed distance grid ϕ^0 is initialized by computing the distance of each pixel from an arbitrary circle contained within the bounds of the grid. Rather than discuss every term, I will focus on a few examples with accompanying shader code.

Computing the Energy Term. The energy term $-\lambda_1(u_o - c_1)^2 + \lambda_2(u_o - c_2)^2$ is computed in $4 + 2\log N$ rendering passes where N is the width of the source image. This term essentially computes the variance of the pixels in the source image that lie inside and outside of the zero level sets embedded in ϕ^t. The first shader computes

Algorithm 1 Moving Object Detection Using Flux Tensor

Input : Infrared image sequence $\mathbf{I}_{IR}(\boldsymbol{x}, t)$, Filters, trace threshold T_{flux}
Output : Foreground mask sequence $\mathrm{FG}_{\mathrm{M}}(\boldsymbol{x}, t)$

// Filters.Dxy : Spatial derivative filter
// Filters.Sxy : Spatial smoothing filter
// Filters.D1t : Temporal derivative filter (first order)
// Filters.D2t : Temporal derivative filter (second order)
// Filters.Axy : Spatial averaging filter
// Filters.At : Temporal averaging filter
1: $nhDt = \lfloor size(Filters.Dt)/2 \rfloor$
2: $nhAt = \lfloor size(Filters.At)/2 \rfloor$
3: **for** each time t **do**
4: $[\mathbf{I}_{xs}(t), \mathbf{I}_{sy}(t), \mathbf{I}_{ss}(t)] \leftarrow$ ComputeIxsIsyIss($\mathbf{I}(t)$, Filters.Dxy, Filters.Sxy)
5: FIFO $\leftarrow$ Insert(FIFO, t, $\mathbf{I}_{xs}(t), \mathbf{I}_{sy}(t), \mathbf{I}_{ss}(t)$)
6: $t_d = t - nhDt$
7: $[\mathbf{I}_{xt}(t_d), \mathbf{I}_{yt}(t_d), \mathbf{I}_{tt}(t_d)] \leftarrow$ ComputeIxtIytItt(FIFO,td,Filters.D1t, Filters.D2t)
8: $Trace(t_d) = \mathbf{I}_{xt}(t_d)^2 + \mathbf{I}_{yt}(t_d)^2 + \mathbf{I}_{tt}(t_d)^2$
9: $Trace(t_d) =$ SpatialAveraging($Trace(t_d)$, Filters.Axy)
10: FIFO $\leftarrow$ Insert(FIFO, t_d, $Trace(t_d)$)
11: $t_a = t - nhDt - nhAt$
12: $Trace(t_a) =$ TemporalAveraging(FIFO, t_a, $Filters.At$)
13: Initialize motion mask, $FG_M(t_a) \leftarrow 0$
14: $\mathrm{FG}_{\mathrm{M}}(Trace(t_a) > T_{flux}) \leftarrow 1$
15: $\mathrm{FG}_{\mathrm{M}}(t_a) \leftarrow$ PostProcess($\mathrm{FG}_{\mathrm{M}}(t_a)$)
16: **end for**

Algorithm 2 Four-Color Level Set Segmentation

Input : A time-varying sequence of N images and constants in (3.18)
Output : A time-varying sequence of N, "four-colored" masks

1: Initialize the segmentation process on the starting image, using level lines [66]. Isolate cells that may be touching or very close to each other to produce the segmentation mask (used in Step 4).
2: **for** $f = 1 \ldots N$ **do**
3: Project the segmentation mask (i.e., converged level sets) from the previous frame as an initial estimate, in order to speed up convergence (see [66, 94]).
4: Extract connected components from the *binary* segmentation mask.
5: Compute an adjacency graph for these connected components.
6: Apply a suitable graph-vertex coloring algorithm (see [95]) to partition the cells into *at most* four independent sets, and produce a *colored* segmentation mask so that neighboring cells belong to different colors
7: Associate one level set function with each mask color, and calculate the signed distance functions for each of the four initial zero level sets (i.e., ϕ_i^0 at evolution time t=0).
8: **for** $i = 1 \ldots K$ iterations **do**
9: Update c_{in} and c_{out}.
10: Evolve the level set within the narrow band of a cell using Euler-Lagrange equations.
11: Enforce an explicit coupling rule that the narrow band of a cell, ϕ_i *cannot overlap* with the level set of any of its neighbors.
12: **end for**
13: Generate a binary mask from the four-color segmentation and apply morphological filtering to remove spurious fragments.
14: Apply a spatially-adaptive level line-based coarse segmentation to the background (i.e., complement of the dilated colored segmentation mask), in order to detect objects not present in the previous frame (i.e., new objects entering the current frame).
15: **end for**

the Chan and Vese formulation of the regularized Heaviside function shown in (3.35) where x is the cell pixel value in ϕ^t and ϵ is a small constant (Algorithm 1).

$$H_\epsilon(x) = \frac{1}{2} + \frac{1}{\pi} tan^{-1}\left(\frac{x}{\epsilon}\right) \tag{3.35}$$

Table 3.2 Computing the Regularized Heaviside Function

```
void computeHeaviside(float2 coords : WPOS,
                      uniform samplerRECT phi,
                      uniform float epsilon,
                      out float4 color : COLOR)

{
   color = float4(0.0, 0.0, 0.0, 0.0);
   const float invpi = 0.318309886;
   float phiVal = texRECT(phi, coords).r;
   float hSide = 0.5 * (1 + (2 * invpi) * atan(phiVal / epsilon));
   color.r = hSide;
   color.g = 1 - hSide;
}
```


Table 3.3 Summing a Data Array

```
void arraySum2Df(float2 coords :WPOS,
                 uniform samplerRECT inputTex,
                 out float4 color : COLOR)
{
  color = float4(0.0, 0.0, 0.0, 1.0);
  float2 topleft = ((coords-0.5)*2.0)+0.5;
  float2 val1 = texRECT(inputTex, topleft).rg;
  float2 val2 = texRECT(inputTex, topleft+float2(1,0)).rg;
  float2 val3 = texRECT(inputTex, topleft+float2(1,1)).rg;
  float2 val4 = texRECT(inputTex, topleft+float2(0,1)).rg;
  color.rg = (val1 + val2 + val3 + val4);
}
```

The next shader performs a reduction operation to compute the sum of H_ϵ and $1 - H_\epsilon$, which roughly equates to the labeling pixels as interior or exterior to the zero level sets which smoothing numeric discontinuities (Algorithm 2).

The following shader reads in the full size texture array but uses a rendering target texture a quarter of its size. For each pixel its up-sampled northwest neighborhood is read from inputTex and the four values are summed. When the result of the Heaviside shader is provided as initial input, the result is the sum of pixels inside and outside the zero level set after log N iterations.

The rest of the shaders find the average intensities of the source image u_0 inside and outside the zero level set, compute the variance, and finally apply the constants and sum the interior and exterior terms.

Distance Transform. To compute the distance transform over the zero level set we use the jump flooding method described by [61]. They proposed a variation on a flood fill algorithm which provides an inexact result in log n rendering passes with additional passes resulting in fewer errors until it converges to the result of a standard flood fill. In their follow-up paper they offer improvements and variants on the algorithm to reduce errors and demonstrate its applicability to the 3D image domain.

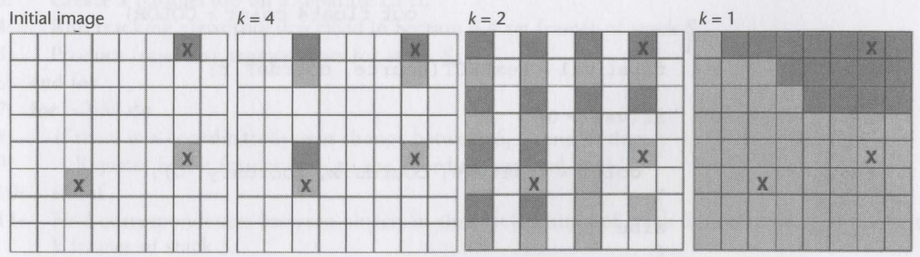


Figure 3.11 Example of performing multiple iterations of the jump flood algorithm. The initial image on the left is composed of three intial seed pixels and the subsequent images show nearest propagated seed pixel for each jump step size *k*.

Algorithm 3 Compute a 2D Neighborhood Adj. Graph

Input : P, a 2D mask with N labeled objects,
T_{LD}, threshold to detect ridges,
T_{HS}, threshold for max. hole size, and
σ, to control smoothing.
Output : $\mathcal{N}(\mathbf{P})$, the adjacency graph of P

1: Remove labeled 8-connected pixels in P that are adjacent to one or more different labels by applying a "*detach operator*".
2: Convert the processed mask into a binary image $\mathcal{B}$.
3: Compute the Euclidean distance transform (EDT), $\mathcal{D}$, of $\mathcal{B}$ using the FH-EDT algorithm [96].
4: Compute $\mathrm{E} = \nabla^2 G_\sigma \circledast \mathcal{D}$, the Laplacian of the smoothed EDT.
5: Obtain a binary image, E_{thr}, from E using a threshold value T_{LD}.
6: Fill holes using T_{HS}, the hole-size threshold.
7: Apply a suitable thinning algorithm (e.g., [97, 98]) on E_{thr} to obtain an image with 1-pixel thick GVD boundaries, E_{thr}^{thn}.
8: Apply any homotopy-preserving algorithm [99] to prune *branches* from the generalized Voronoi diagram, E_{thr}^{thn}.
9: Assign $\mathcal{Q} \leftarrow (\mathrm{E}_{thr}^{thn})^c$; the complementary image of E_{thr}^{thn}
10: Using 4-connectivity, label the connected components of $\mathcal{Q}$.
11: Update the neighborhood relationship map $\mathcal{N}(\mathbf{P})$ by checking a 3×3 neighborhood of each background pixel (i.e., boundary pixels of connected components) in $\mathcal{Q}$. $\mathcal{N}(\mathbf{P})$ is subsequently used in graph-vertex coloring.

The jump flood distance transform uses the filled pixels of a binary image mask as the seeds for the Euclidean distance calculations. For the initial rendering pass every pixel (x, y) in the image computes the minimum distance from its neighbors $(x \pm k,\ y \pm k)$ where k is initially the half width of the image. The minimum distance and the coordinates of the corresponding seed pixel are stored. For successive rendering passes the k is halved and each pixel repeats the operation but bases its distance calculation on the stored coordinates of the seed pixels propagated in earlier steps. Figure 3.11 demonstrates on an 8×8 image with an initial step size $k = 4$ coordinates of the initial seed pixels are propagated.

Table 3.4 Jump Flood Distance Transform Initialization

```
void initJumpFlood2Df(float2 coords : WPOS,
                      uniform samplerRECT source,
                      out float4 color : COLOR)

{
  float val = texRECT(source, coords).r;

  if(val > 0)
  {
    color = float4(0, coords.x, coords.y, 0);
  }
  else
  {
    color = float4(10000, -1, -1, 0);
  }
}
```


Algorithm 4 Tracking Algorithm

Input : A time-varying sequence of N, "colored" masks
Output : Trajectories and temporal cell statistics

1: For each frame $I(y,t)$ at time t, the tracking module receives four foreground mask layers, $\Omega_k(t)$, that correspond to level sets from the "four-color" segmentation algorithm.
2: Connected component analysis is performed on all refined foreground layers such that each $\Omega_k(t)$ is partitioned into n_k disjoint regions

$$\Omega_k(t) = \{\Omega_{k,1}(t), \Omega_{k,2}(t), \ldots, \Omega_k, n_k(t)\}$$

that ideally correspond to n_k individual cells.
3: For each disjoint region $\Omega_{k,i}$, features such as centroid, area, bounding box, and support map are extracted. Region information from all four layers are combined and relabeled as

$$\Omega_i(t), i \in [1 \ldots n], \text{ and } n = \sum_{k=1}^{4} n_k$$

without merging connected regions from different foreground layers. This preserves identities of previously disjoint cells and prevents false trajectory merges. This is similar to the technique described in Step 1 of Algorithm 3.
4: Relabeled region information is arranged in an "Object-Match" graph structure $\mathcal{O}_{\mathcal{GR}}$ that is used in tracking. Nodes in the graph represent objects $\Omega_i(t)$, while edges represent object correspondences.
5: Correspondence analysis searches for potential object matches in consecutive frames. $\mathcal{O}_{\mathcal{GR}}$ is updated by linking nodes that correspond to objects in frame $I(y,t)$ with nodes of potential corresponding objects in frame $I(y,t-1)$. $\mathcal{C}_{\mathcal{M}}(i, j)$, the confidence value for each match is also stored with each link.
6: The trajectory generation module analyzes $\mathcal{O}_{\mathcal{GR}}$ and generates cell trajectories.

Algorithm 5 Distributed Tracking Algorithm

Input : A time-varying sequence of $N-$frames
Output : Trajectories and temporal cell statistics

1: Divide $N-$frame input sequence into M stacks $S_{1\ldots M}$ with $\{n_1, n_2, \ldots, n_M\}$ frames with k overlapping frames between each stack $(S_i(n_i - k + 1 : n_i) = S_{i+1}(1 : k))$.
2: **for** each stack S_i **do**
3: Create a parallel job on a separate CPU.
4: Run tracking algorithm described in Section 3.4 on frames of stack S_i.
5: Produce trajectory segment lists for stack S_i.
6: **end for**
7: **for** i=2:M **do**
8: **if** there is a coordinate system change between S_{i-1} and S_i **then**
9: Register first k frames of S_i to the coordinate system of S_{i-1}.
10: **end if**
11: Find correspondences between objects in the last k frames of stack S_{i-1} to objects in the first k frames of stack S_i.
12: Propagate trajectory labels of objects in stack S_{i-1} to the corresponding objects in S_i.
13: Propagate the new labels in S_i to their children trajectories.
14: **end for**

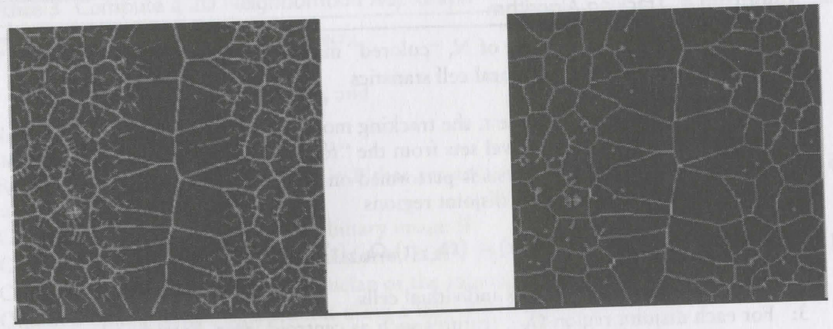


Figure 3.12 The image on the left shows the result of directly applying the Laplacian kernel to the distance transform. The image on the left is the result of thinning which eliminates most of the orphaned pieces.

For our implementation we first generate a binary mask from the zero level set to act as the seed image. For the jump flood distance transform the red, green, and blue texture color channels are used to store the computed distance from its nearest seed pixel, the seed x coordinate and the seed y coordinate. During an initialization step the distance for the seed pixels are set to 0 and their own texture coordinates are set. The nonseed pixels are given a large initial distance and their coordinates are set to (−1, −1) (Algorithm 3).

The second stage of the distance transform applies the description given above as the step size k is halved each rendering pass. Level sets are represented as signed distance fields and the jump flood can be used to produce this result by applying the binary mask used during initialization. After the distance transform is completed the mask is applied and any pixel beneath the mask has its value negated (Algorithm 4).

Generalized Voronoi Diagram. The generalized Voronoi diagram is constructed using the previously computed distance transform by first applying a 3 × 3 Laplacian kernel to find the local maxima with a threshold of 0.5 (Algorithm 5).

The result shown in Figure 3.12 has orphaned fragments with free ends and loops. To clean up the final result a thinning shader is applied to the image. For

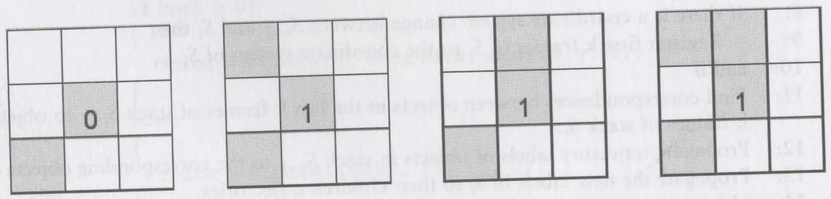


Figure 3.13 A subset of 3 × 3 structuring elements used to thin the Voronoi diagram.

each pixel, its eight neighbors are assigned a value with the upper left as 0 and the lower right as 128. A bit vector is formed with values from 0 to 255 by setting a bit if the corresponding pixel is filled and clearing it if empty. Figure 3.13 shows 4 out of the 256 possible pixel arrangements. The value of the vector is used to index a lookup table of all possible neighborhoods with the corresponding action. A 0 indicates the pixel should be cleared while a 1 leaves the pixel filled. The image after thinning is applied still contains isolated loops which could be eliminated by examining a larger neighborhood.

3.2.4 Results and Discussion

Figure 3.14 shows sample 4-level sets cell segmentation and GVD results. These sample result corresponds to a wound healing image sequence, obtained from phase-contrast microscopy, consisting of 136 frames of dimensions 300 × 300

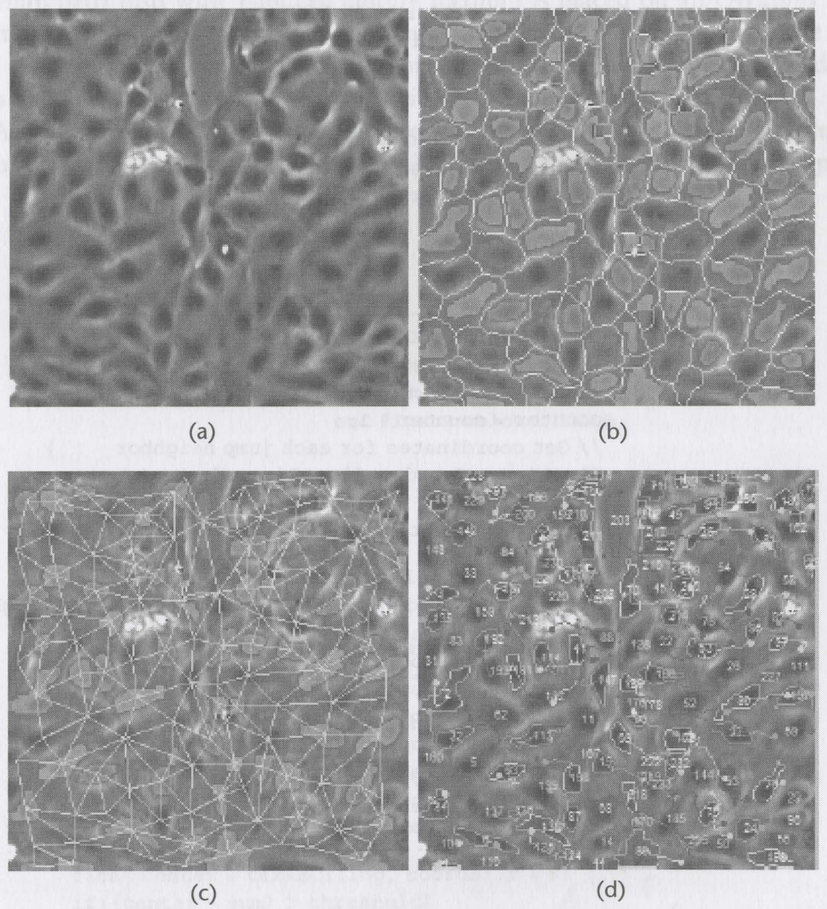


Figure 3.14 Segmentation and tracking results for the wound sequence: (a) original image (frame #135); (b) 4-level set mask + GVD; (c) neighborhood graph; and (d) cell trajectories.

Table 3.5 GPU Level Set Timing Results

Operation	*Timing(%)*
Initialization	0.01
Create masks	2.63
Compute distance transform	34.93
Create GVD	22.92
Redistance ϕ	2.08
Compute regularized dirac	2.51
Compute energy term	18.93
Compute length term	3.80
Compute area term	4.06
Update ϕ	4.98
Render image	3.15

Table 3.6 Jump Flood Distance Transform

```
void jumpFlood2Df(float2 coords : WPOS,
                  uniform float jumpStep,
                  uniform samplerRECT source,
                  out float4 color : COLOR)

{
  float4 currentPixel = texRECT(source, coords);

  //Initialize minimum distance and seed coordinates
  float minDist = currentPixel.r;
  float2 minCoords = currentPixel.gb;

  int stepSign[] = {1,0,-1};

  int counter = 0;

  if( currentPixel.r > 0){
    for(int i = 0; i < 3; i = i + 1){
      for(int j = 0; j < 3; j = j + 1){
        counter = counter + 1;
        // Get coordinates for each jump neighbor
        float2 jumpCoords = float2(coords.x +
          (stepSign[i] * jumpStep),
           coords.y + (stepSign[j] * jumpStep));

        // Get seed pixel coordinate for jump neighbor
        float2 seedCoords = texRECT(source, jumpCoords).gb;

        // Find nearest seed pixel from jump neighbors
        float seedDist = distance(seedCoords, coords);

        if(seedDist < minDist)
        {
          minDist = seedDist;
          minCoords = seedCoords;
        }
      }
    }
  }
  color = float4(minDist, minCoords.x, minCoords.y, 0);
}
```


(40 μm$\times$ 40 μm) with image intensities $I \in [0,255]$. The sequence has been obtained using a monolayer of cultured porcine epithelial cells, as described by Salaycik et al. in [61]. Images were sampled uniformly over a 9:00:48 hour period and acquired using a phase contrast microscope, with a 10$\times$ objective lens, and at a resolution of approximately 0.13 μm per pixel. Figure 3.14(a) shows frame #135 from the original sequence. Figures 3.14(b, c) show 4-level set masks, GVD boundaries, and resulting neighborhood graph.

Some comparative segmentation results for directed edge-profile guided active contours are shown in Figure 3.9. These magnified images correspond to sample Bovine aortic endothelial cells (BAECs; Cambrex East Rutherford, New Jersey) between passages 12 and 16. They were grown in DMEM (Invitrogen, Carlsbad, California) containing 10 streptomycin and kanamycin sulfate (Invitrogen, Carlsbad, California). Cell cultures were maintained in a humidified incubator at 37°C, with 5 every 3 to 4 days. Rat-tail collage I (BD Biosciences, Bedford, Massachusetts) at 0.3 mg/ml in acidic acid was absorbed onto the glass coverslip for 24 hours. Then the coverslip was rinsed in PBS to remove unbound collagen. A glass coverslip in a 35-mm petri dish with 1-ml cell culture medium is placed on an Olympus IX70 inverted microscope. 2 $\times$ 104 cells from cell suspension were added to the petri dish. Images were recorded in phase contrast mode with a CoolSNAPfx digital video camera (Photometrics, Tucson, Arizona), using a 10X objective.

A similar approach has also been used in segmentation of a low density culture of human melanoma cell line WM793 [23]. The proposed method was applied

Table 3.7 Laplacian Kernel

```
void laplacianFilter2Df(float2 coords : WPOS,
                        uniform samplerRECT img,
                        uniform float threshold,
                        out float4 color : COLOR)
{
  //3x3 Laplacian
  // -1 -1 -1
  // -1  8 -1
  // -1 -1 -1

  float sum = 0;
  sum += texRECT(img, float2(coords.x, coords.y + 1)).r;
  sum += texRECT(img, float2(coords.x - 1, coords.y + 1)).r;
  sum += texRECT(img, float2(coords.x + 1, coords.y + 1)).r;
  sum += texRECT(img, float2(coords.x + 1, coords.y)).r;
  sum += texRECT(img, float2(coords.x - 1, coords.y)).r;
  sum += texRECT(img, float2(coords.x, coords.y - 1)).r;
  sum += texRECT(img, float2(coords.x + 1, coords.y - 1)).r;
  sum += texRECT(img, float2(coords.x - 1, coords.y - 1)).r;

  float center = texRECT(img, coords).r * 8;
  if((center - sum) > threshold)
    color = float4(1,0,0,0);
  else
    color = float4(0,0,0,0);
```

to human melanoma cells. The local image variance and the Hamming texture measure are chosen as the robust features for this type of cells since the cells have visibly different variance and texture from the background whereas intensity does not constitute a discriminating feature. Figure 3.1 shows a sample frame from one field in the T25 plastic culture flask of a control experiment with a low density culture of human melanoma cell line WM793 to assess any toxicity to the highthroughput imaging system [3]. Pixel resolution is 0.67 micron 0.59 micron in X and Y using a 20 objective lens.

The GPU level set implementation ran approximately 3 to 15 times faster than a narrow band level set CPU implementation depending on the dimensions of the input images. However, it is quite difficult to make a valid comparison between different implementations on the basis of hardware platforms. Many novices to GPU programming have assumed they would immediately get large speed improvements simply by porting their code to a graphics processor with little regard for the type of problem or the quality of their equipment. As an analysis of the GPU level set implementation, Table 3.5 shows the percentage of total time taken by each operation within an iteration of the level set process. The three most expensive steps are computing the distance transform, creating the generalized Voronoi diagram, and computing the energy term. These operations represent steps that require extra processing to compute results not entirely suited for the GPU. The jump flood distance transform takes $\log N$ (where N is the width of a the side of a square image) rendering passes but still has to consider N^2 pixels each pass compared to CPU image based, such as by distance transforms by Dannielsson [62] and Felzenzwalb and Huttenlocher [63], which complete in linear time. The number of parallel stream processors and rapid memory access offset the cost, but this becomes less effective for increasingly larger images. The energy term requires computing the sum of all pixels within texture arrays twice using parallel reduction. While this method is efficient for the GPU it still requires reading every pixel in the initial step and then a quarter of the number of pixels for each succesive step. Obviously the CPU equivalent is a single for loop over an array of values once. The Voronoi diagram computation expense is accrued during the thinning step to remove spurs and orphaned fragments, but the standard implementation does not suffer in comparision to a typical CPU operation.

3.3 Cell Tracking

Tracking is a fundamental step in the analysis of long-term behavior of migrating cells. Various studies have been reported in the literature highlighting the importance of cell migration analysis such as: [64] in understanding drug treatments effects on cancer cells, [65] in assessing aggressiveness of cancer models, [66, 67] in study of inflammatory diseases, [68] in wound closure (which is important in embryonic development and tissue repair), [69] in quantification of protein activation, and other studies related to migrating cells [70, 71].

A majority of cell migration studies fall into one of the following classes:

- Cell motility through optic-flow analysis;
- Cell tracking through contour evolution;
- Cell tracking by detection and explicit correspondence analysis.

Optic flow methods provide dense motion flows, but not trajectories for individual objects, which is needed for the study of long-term behavior of individual cells. Active contour-based approaches provide trajectory information and are widely used in tracking biological objects with the underlying assumption of small displacements in objects (see [28, 66]). Generally, active contour tracking uses contour from the previous frame as an initial estimate for the current frame and evolves this estimate using velocity field, intensity boundaries, and some additional constraints such as shape priors (see [67]), or by processing 2D image slices taken over time as a spatiotemporal volume and applying 3D active contour segmentation [72].

In this section we present a detection-based cell tracking algorithm that extends our previous work in [2, 22, 73]. Detection-based tracking enables us to efficiently control each cell association and handle large amounts of displacements. Tracking is achieved by resolving frame-to-frame correspondences between multiple cells that are segmented as described in Section 3.3.

Recently, two similar tracking algorithms were presented by Al-Kofahi et al. [74] and by Li et al. [75]. In [74], tracking is implemented to analyze cell lineage. Cells are segmented using a combination of adaptive thresholding and watershed transformation, and tracked using explicit association analysis. Integer programming is employed to find optimal cell associations from a subset of possible associations. In Li et al. [75], active contours (with an auxiliary label function) are used as the first stage for tracking. Graph matching is subsequently implemented on cells and tracks that are unmatched in the first stage. In our approach, graph matching is used for all cells and tracks, without evolving any auxiliary label function. We can use graph matching for all the cells without suffering from matching ambiguities, thanks to the prevention of false merges during 4-color level set segmentation (Section 3.3.1), and the multistage overlap distance $\mathcal{D}_{MOD}$ during tracking.

A detailed description is presented in Algorithm 4. Correspondence analysis and trajectory generation are key components of the tracking algorithm presented above and are further elaborated in the following subsections.

3.3.1 Cell-to-Cell Temporal Correspondence Analysis

Cell-to-cell temporal correspondence analysis consists of four major steps:

1. Object-to-object distance computation;
2. Match confidence matrix construction;
3. Absolute match pruning;
4. Bidirectional relative match pruning.

Appropriate and efficient object-to-object match distance computation is a key component of any tracking process. In the presented method, the distances between two cells Ω_i and Ω_j in consecutive frames are computed using a multistage overlap distance $\mathcal{D}_{MOD}$ (3.36), which consists of three distinct distance functions $\mathcal{D}_{bbx}$,

$\mathcal{D}_{msk}$, and $\mathcal{D}_{olp}$ for three different ranges of cell motion. The distance selection is based on topological spatial relations between objects in consecutive frames, defined similar to the relation in Randell et al.'s region connection calculus (RCC) [76, 77] and Egenhofer et al.'s 9-intersection model [78, 79]. The correspondence analysis is largely based on proximity because intensity, texture, and shape are not distinguishing features for cells since they share the same appearance.

$$\mathcal{D}_{\mathcal{MOD}}(\Omega_\mathrm{i}, \Omega_\mathrm{j}) = \begin{cases} \mathcal{D}_{bbx} & \text{if} \quad \mathrm{bbx}(\Omega_\mathrm{i}) \cap \mathrm{bbx}(\Omega_\mathrm{j}) = \emptyset \\ \mathcal{D}_{msk} & \text{if} \quad \Omega_\mathrm{i} \cap \Omega_\mathrm{j} = \emptyset \\ \mathcal{D}_{olp} & \text{otherwise} \end{cases} \tag{3.36}$$

The *interbounding-box distance* $\mathcal{D}_{bbx}$ (3.37) quantifies long-range displacement between two regions (cells) in consecutive frames by the distance between their bounding boxes denoted as bbx().

$$\mathcal{D}_{bbx}(\Omega_\mathrm{i},\Omega_\mathrm{j}) = K_{bbx} \times \sqrt{\mathcal{D}_{\mathcal{I}}(\mathrm{bbx}_x(\Omega_\mathrm{i}), \mathrm{bbx}_x(\Omega_\mathrm{j}))^2 + \mathcal{D}_{\mathcal{I}}(\mathrm{bbx}_y(\Omega_\mathrm{i}), \mathrm{bbx}_y(\Omega_\mathrm{j}))^2} \tag{3.37}$$

where $\mathrm{bbx}(\Omega) \equiv < inf_x(\Omega), sup_x(\Omega), inf_y(\Omega), sup_y(\Omega) >$ is the minimum bounding box of a region; $\mathrm{bbx}_x(\Omega) \equiv [inf_x(\Omega), sup_x(\Omega)]$ and $\mathrm{bbx}_x(\Omega) \equiv [inf_y(\Omega), sup_y(\Omega)]$) are intervals obtained by projecting the bounding box $\mathrm{bbx}(\Omega)$ onto x and y axes; $\mathcal{D}_{\mathcal{I}}$ is interval distance defined in (3.38); and K_{bbx} is a constant. In the cases of overlapping bbx_x or bbx_y intervals, $\mathcal{D}_{\mathcal{B}}$ corresponds to the minimum edge-to-edge distance; otherwise it corresponds to minimum corner-to-corner distance [Figure 3.15(a)].

$$\mathcal{D}_{\mathcal{I}}(\mathcal{I}_A,\mathcal{I}_B) = \begin{cases} 0 & \text{if } \mathcal{I}_A \cap \mathcal{I}_B \neq \emptyset \\ min\{|\mathcal{I}_A(1) - \mathcal{I}_B(2)|, |\mathcal{I}_B(1) - \mathcal{I}_A(2)|\} & \text{otherwise} \end{cases} \tag{3.38}$$

The *intermask distance* $\mathcal{D}_{msk}$ quantifies mid-range displacement between two regions (cells) in consecutive frames, by the minimum contour-to-contour dis-

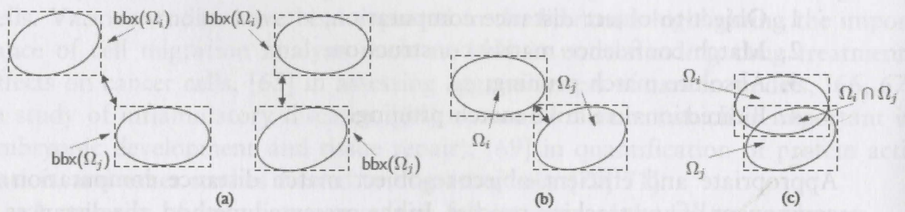


Figure 3.15 Object-to-object distance $\mathcal{D}_{\mathcal{MOD}}$: (a) $\mathcal{D}_{bbx}$: interbounding-box distance; (b) $\mathcal{D}_{msk}$: intermask distance; and (c) $\mathcal{D}_{olp}$: tonal-weighted overlap distance.

tance. This is computed in terms of the minimum number of morphological dilations needed to overlap the two regions [Figure 3.15(b)]. $\mathcal{D}_{msk}$ is defined as

$$\mathcal{D}_{msk}(\Omega_\mathrm{i}, \Omega_\mathrm{j}) = K_{msk} \times inf\{k : \delta_k(\Omega_\mathrm{i}) \cap \Omega_\mathrm{j} \neq 0 \ ; \ \Omega_\mathrm{i} \cap \delta_k(\Omega_\mathrm{j}) \neq 0\} \tag{3.39}$$

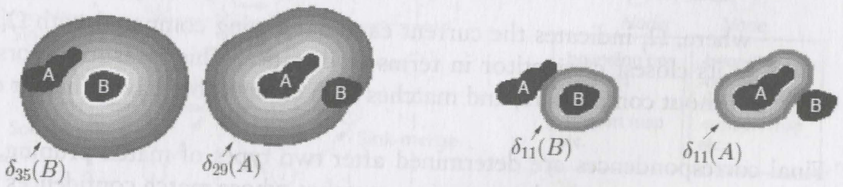
where δ_k denotes k-times dilation with a unit structuring element, and K_{msk} is a constant. $\mathcal{D}_{msk}$ is closely related to Hausdorff distance [80] (Figure 3.16), redefined in [81] using morphological operators as

$$H(A,B) = inf\{k : A \subseteq \delta_k(B) \ ; \ B \subseteq \delta_k(A)\} \tag{3.40}$$

The *tonal-weighted overlap distance* $\mathcal{D}_{olp}$ quantifies small-range displacement between two regions/cells [Figure 3.15(c)] by the degree of their overlap, in terms of shape *and* tonal dissimilarities. In order to emphasize overlap in nuclei (i.e., dark regions with low intensity values), and to de-emphasize cytoplasm overlap (i.e., light regions with high intensity values), overlapping and nonoverlapping regions are weighted by local tonal differences, as

$$\mathcal{D}_{olp}(\Omega_\mathrm{i},\Omega_\mathrm{j}) = K_{olp} \times ; \frac{\int_{\Omega_\mathrm{i} \setminus \Omega_\mathrm{j}} (1 - I_i(y))dy + \int_{\Omega_\mathrm{j} \setminus \Omega_\mathrm{i}} (1 - I_j(y))dy + \int_{\Omega_\mathrm{i} \cap \Omega_\mathrm{j}} |I_i(y) - I_j(y)|dy}{\int_{\Omega_\mathrm{i}} I_\mathrm{i}(y)dy + \int_{\Omega_\mathrm{j}} I_J(y)dy} \tag{3.41}$$

where the intensity images, $I_i(y) = I_i(y,t)$ and $I_j(y) = I_j(y,t-1)$, are scaled such that $I \in [0,1]$, and K_{olp} is a constant. The first two terms in the numerator of (3.41) account for the distance due to uncovered regions in frames at time instants t and $t-1$, respectively. The complement of intensity images are used to obtain higher distances for uncovered low intensity regions (i.e., nuclei). The third term in the numerator accounts for the intensity dissimilarity within the overlapping region.



(a) $A \subseteq \delta_{35}(B)$ (b) $B \subseteq \delta_{29}(A)$ (c) $A \cap \delta_{11}(B) \neq 0$ (d) $\delta_{11}(A) \cap B \neq 0$

Figure 3.16 Hausdorff (a, b) and $\mathcal{D}_{\mathcal{M}}$ (c, d) distances for two cells A and B. $H(A,B) = min(35,29)$, $\mathcal{D}_{\mathcal{M}} = min(11,11)$.

The denominator is used to normalize the distance by the area of the two cells being compared.

This multistage distance measure depends on size and shape similarity of the compared regions, besides their proximity, and thus have several advantages over the widely used centroid distance measure, particularly for mitosis (cell split) cases. During mitosis, epithelial cells often become elongated and split across the minor axis. This produces a big increase in the centroid distance, and for dense cell populations the distances between a cell and its children become comparable to the distances between a cell and its neighboring cells. In such a scenario, a low gating threshold would result in parent-to-children matches being discarded, resulting in discontinuities in cell trajectories. However, if a high gating threshold is used, correspondence ambiguities may arise. Multistage overlap distance measure overcomes these problems. Depending on the appearance and motion characteristics of the cells being tracked, match distance can be modified to incorporate other features (i.e., shape, color, texture, and so forth).

Using the above information, for each frame $I(\boldsymbol{y},t)$ of the image sequence, a match matrix $\mathcal{M}$, and a confidence matrix $\mathcal{C}_{\mathcal{M}}$ are constructed. $\mathcal{M}(\Omega_i,\Omega_j)$ indicates whether the ith object in $I(\boldsymbol{y},t)$, corresponds to the jth object in $I(\boldsymbol{y},t-1)$. All the elements of $\mathcal{M}$ are initially set to one indicating a *match*. $\mathcal{C}_{\mathcal{M}}(\Omega_i,\Omega_j)$ indicates the confidence of this match and consists of weighted sum of two components:

- *Similarity confidence*, $\mathcal{C}_{\mathcal{SIM}}(\Omega_i,\Omega_j)$, a measure of the similarity between the matched objects, defined as

$$\mathcal{C}_{\mathcal{SIM}}(\Omega_i,\Omega_j) = 1 - \frac{\mathcal{D}_{\mathcal{MOD}}(\Omega_i,\Omega_j)}{\mathcal{D}^{\max}_{\mathcal{MOD}}} \tag{3.42}$$

  where the constant $\mathcal{D}^{\max}_{\mathcal{MOD}}$ is used to normalize $\mathcal{D}_{\mathcal{MOD}}$.
- *Separation confidence*, $\mathcal{C}_{\mathcal{SEP}}(\Omega_i,\Omega_j)$, a measure of the competition between possible matches for the current object, defined as

$$\mathcal{C}_{\mathcal{SEP}}(\Omega_i,\Omega_j) = \begin{cases} 1 & \text{no competitor,} \\ 0.5 - \dfrac{\left(\mathcal{D}_{\mathcal{MOD}}(\Omega_i,\Omega_j) - \mathcal{D}_{\mathcal{MOD}}(\Omega_i,\Omega_j^*)\right)}{2\times\max\left(\mathcal{D}_{\mathcal{MOD}}(\Omega_i,\Omega_j),\mathcal{D}_{\mathcal{MOD}}(\Omega_i,\Omega_j^*)\right)} & \text{otherwise} \end{cases} \tag{3.43}$$

  where, Ω_j indicates the current candidate being compared with Ω_i, and Ω_{j^*} is its closest competitor in terms of distance. This measure favors matches without competitors, and matches with competitors having higher distances.

Final correspondences are determined after two types of match pruning, absolute and relative. During absolute pruning, matches whose match confidences are lower than a priori defined threshold T_A are pruned by setting corresponding elements in match matrix $\mathcal{M}$ to 0:

$$\mathcal{C}_{\mathcal{M}}(\Omega_i,\Omega_j) < T_A \Rightarrow \mathcal{M}(\Omega_i,\Omega_j) = 0 \quad \Omega_i \in I(\boldsymbol{y},t), \Omega_j \in I(\boldsymbol{y},t-1) \tag{3.44}$$

During backward relative pruning, for each cell $\Omega_i \in I(\boldsymbol{y},t)$, best matching cell $\Omega_j^* \in I(\boldsymbol{y},t-1)$ is determined. Matches for Ω_i, whose match confidences are lower than $T_R \times \mathcal{C}_{\mathcal{M}}(\Omega_i,\Omega_j^*)$, where T_R is a constant, are pruned:

$$\mathcal{C}_{\mathcal{M}}(\Omega_i,\Omega_j) < T_R \times \mathcal{C}_{\mathcal{M}}(\Omega_i,\Omega_j^*)) \Rightarrow \mathcal{M}(\Omega_i,\Omega_j) = 0 \quad \Omega_i \in I(\boldsymbol{y},t), \Omega_j \in I(\boldsymbol{y},t-1) \tag{3.45}$$

During forward relative pruning, the same relative pruning process is applied in the direction $I(\boldsymbol{y},t-1) \rightarrow I(\boldsymbol{y},t)$ by determining for each cell $\Omega_j \in I(\boldsymbol{y},t-1)$, best matching cell $\Omega_i^* \in I(\boldsymbol{y},t)$, and pruning accordingly.

3.3.2 Trajectory Segment Generation

Information on detected cells (centroid, area, shape, support map, and intensity distribution) and their temporal correspondences (obtained from match matrices $\mathcal{M}$ for each frame) are arranged in a graph structure ObjectMatchGraph $\mathcal{O}_{\mathcal{GR}}$ (Figure 3.17). Nodes in $\mathcal{O}_{\mathcal{GR}}$ represent cells and associated features, while edges represent frame-to-frame cell correspondences and associated match confidence values. The segment generation module traces links on $\mathcal{O}_{\mathcal{GR}}$ to generate trajectory segments. A multihypothesis testing approach with delayed decision is used. Rather than picking a single best match or some subset of the matches with limited information, many possible matches are kept and gradually pruned, as more information becomes available in the decision process. Besides one-to-one object matches, this scheme supports many-to-one, one-to-many, many-to-many, one-to-none, or none-to-one matches that may result from false detections, or false associations, segmentation errors, occlusion, entering, exiting, or dividing of cells. The segment generation module has three main steps:

1. *Node Classification:* The cells/objects in each frame (nodes in the ObjectMatchGraph $\mathcal{O}_{\mathcal{GR}}$) are classified into nine types based on the number of

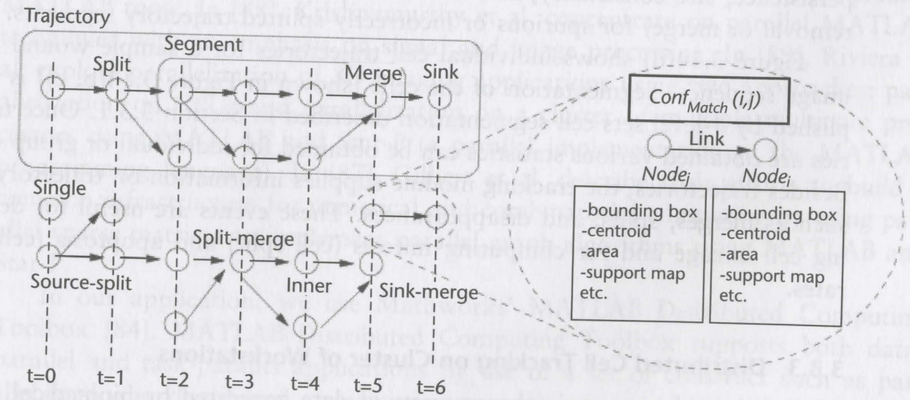


Figure 3.17 ObjectMatchGraph used in cell tracking. Nodes represent detected cells and associated features, while links represent frame-to-frame cell correspondences and associated confidence values.

parent and child objects: *single* (no parent-no child), *source* (no parent-one child), *source-split* (no parent-many children), *sink* (one parent-no child), *inner* (one parent-one child), *split* (one parent-many children), *sink-merge* (many parents-no child), *merge* (many parents-one child), *merge-split* (many parents-many children) (Figure 3.16).

2. *Segment Generation: Trajectory segments* are formed by tracing the nodes of the ObjectMatchGraph *ObjectMatch Graph*. A linked list of inner nodes (objects/cells), starting with a source or split type node and ending with a merge or sink type node, are identified and organized in a data structure called *SegmentList*.
3. *Segment Labeling:* Extracted trajectory segments are labeled using a method similar to connected component labeling. Each segment without a parent is given a new label. Segments that have parents inherit the parents' labels. In case of multiple parents, if the parents' labels are inconsistent, then the smaller label (older trajectory) is kept and a flag is set indicating the inconsistency. Trajectories are formed by joining segments sharing the same label.

Cell Trajectory Validation and Filtering

Factors such as vague cell borders, DNA unwinding during cell division, noise, fragmentation, illumination or focus change, clutter, and low contrast cause segmentation and association problems, which result in missing or spurious trajectory segments and false trajectory merges or splits. Filtering is performed at various levels of processing to reduce the effects of such factors. At the *image level*, small objects that may be due to noise, background clutter, or imaging and segmentation artifacts are removed by using morphological operators. At the *segmentation level*, use of coupling (Section 3.3.1) prevents false merges. At the *correspondence level*, unfeasible cell-to-cell matches with distances above a threshold are eliminated through gating. At the *confidence level*, matches with low confidence values are pruned with absolute and relative prunings. A final validation and filtering module analyzes trajectory segments using accumulated evidence such as temporal persistence, size consistency, and area ratios; and to take appropriate action (i.e., removal or merge) for spurious or incorrectly splitted trajectory segments.

Figure 3.14(d) shows individual cell trajectories for a sample wound healing image sequence. Segmentation of the cells [shown in Figure 3.14(b, c)] is accomplished by 4-level sets cell segmentation described in Section 3.3.1. Once trajectories are obtained various statistics can be obtained for individual or group of cells. Besides trajectories, the tracking module supplies information on trajectory events such as merges, splits, and disappearances. These events are useful for determining cell lineage and for computing mitosis (cell split) and apoptosis (cell death) rates.

3.3.3 Distributed Cell Tracking on Cluster of Workstations

With the rapid increase in the amounts of data generated by biomedical applications, high-performance computing systems are expected to play an increased role. Various high-performance computing strategies exist using systems ranging from

accelerators (or many-core processors) such as graphics processing units (GPUs), cell broadband engines (Cell BEs), field-programmable gate arrays (FPGAs), to cluster computers. An intensive study of the parallel computing research landscape can be found in [82]. A large set of parallel computing case studies dealing with bioinformatics and computational biology related problems can be found in [83].

For high-throughput analysis of cell behaviors, our tracking algorithm (Algorithm 4) is parallelized for cluster computers using parallel and distributed job creation capabilities of MATLAB Distributed Computing Toolbox [84]. A cluster computer is a group of computers generally connected through fast local area networks that work together closely so that in many respects they can be viewed as though they are a single computer [85]. The approach is designed for the University of Missouri Bioinformatics Consortium (UMBC) cluster system [86], but can be adapted to other cluster computers. The UMBC cluster system consists of 128 compute nodes and 512 processors. Each node has 4 or 6 GB of memory and is attached to 50 TB of disk using an Infiniband high-speed interconnect infrastructure for both processor and storage access.

Parallel MATLAB

MATLAB is a widely used mathematical computing environments in technical computing. Very high-level languages (VHLLs) like MATLAB provide native support for complex data structures, comprehensive numerical libraries, and visualization along with an interactive environment for execution, editing, and debugging, resulting in codes short, simple, and readable [87]. Recently, there has been an increasing interest in developing a parallel MATLAB. In a 2005 article Ron Choy and Alan Edelman give an extensive survey of projects working on parallelization of MATLAB by providing communication routines (MPI/PVM), routines to split up work among multiple MATLAB sessions, parallel backend, or by compiling MATLAB scripts into native parallel code. They group and study 27 such projects (including pMatlab, MatlabMPI, MultiMATLAB, and Matlab Parallel Toolbox), some of which are no longer in use. Several recent applications use these parallel MATLAB tools. In [88], Krishnamurthy et al. concentrate on parallel MATLAB techniques with an emphasis on signal and image processing. In [89], Riviera et al. explore parallelization of 3D imaging applications using two approaches: parallelization on GPUs and parallelization on a cluster of multiple multicore processors using MATLAB and Star-P (a parallel implementation of the MATLAB programming language). In [87], Gilbert et al. describe their efforts to build a common infrastructure for numerical and combinatorial computing by using parallel sparse matrices to implement parallel graph algorithms using MATLAB and Star-P.

In our application, we use Mathworks' MATLAB Distributed Computing Toolbox [84]. MATLAB Distributed Computing Toolbox supports both data-parallel and task-parallel applications by use of a set of construct such as parallel for-loops, distributed arrays, message-passing functions, and distributed job creation commands. When used with MATLAB Distributed Computing Server, parallel MATLAB programs can be scaled to cluster computers.

Distributed Cell Tracking

Typically computation or memory extensive biomedical image processing applications involve iterative operations, images with high spatial resolution, or long video sequences. Particularly for the last two cases, parallelization by data partitioning can greatly improve performance. In the case of cell tracking, we chose to partition the data in time. Our decision is motivated by: (1) ease of partitioning, (2) degree of task independence (amount of interprocess communication), and (3) amount of code adaptation efforts needed. Distributed cell tracking module consists of two submodules:

1. Intrastack tracking module;
2. Trajectory stitching or consistent labeling module.

The original N-frame sequence is divided into M stacks with k overlapping frames. Objects/cells in each stack are tracked using our tracking algorithm (Algorithm 4) described in Section 3.4. Tracking of cells in each stack is performed independently in parallel using parallel and distributed job creation capabilities of MATLAB Distributed Computing Toolbox [84]. The major steps of the process are given below, where a task corresponds to the tracking of the cells in the assigned stack:

1. Find a job manager.
2. Create a job.
3. Create tasks.
4. Submit the job to the job queue.
5. Retrive job's results.

Each task produces trajectory segment lists. Beside object level information such as object IDs, centroids, sizes, and support maps, each trajectory segment contains IDs of their parents, IDs of their children, and a label indicating the trajectory to which they belong. Since cells in each stack are tracked independently, cell trajectory labels are not consistent in time. The *Trajectory Stitching/Consistent Labeling module* is a serial process that propagates trajectory labels from the previous stack to consistently relabel trajectory segments in the current stack. This process ensures that trajectory segments belonging to the same cell consistently have the same label in time, independent of how many stacks they cross. This problem is similar to the consistent labeling problem in multicamera tracking systems [90, 91]. Our trajectory stitching/consistent labeling process described in Algorithm 5 and illustrated in Figure 3.18 is originally designed for moving camera surveillance applications, and uses both geometry-based and appearance-based object matching and can handle changing field of views. For cell tracking, only single field of view geometry-based object matching is used. For each stack S_i trajectory segments that start within the first k frames T_i^S and trajectory segments that end within the last k frames T_i^E are identified. To consistently label (or stitch) trajectory segments across stacks, correspondences of the trajectory segments T_{i-1}^E to T_i^S are determined by matching the last instances of the objects in T_{i-1}^E to the first instances of the objects in T_i^S. For the general case both appearance and geometry are used in object matching, but for the case of static camera with temporally overlapping stacks, simple proximity

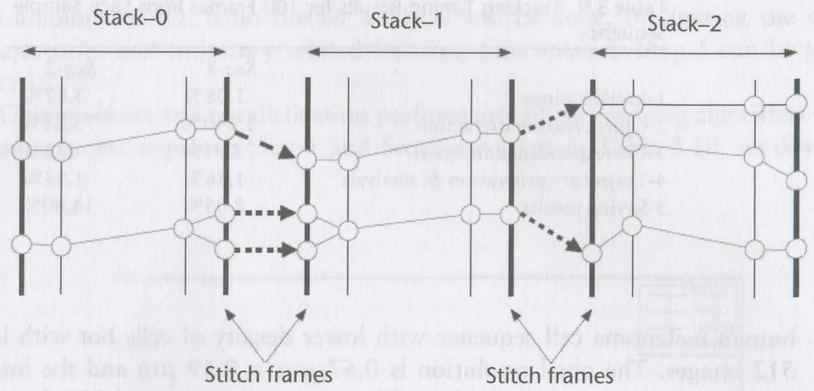


Figure 3.18 Multistack trajectory stitching.

based matching is adequate. Once correspondences are identified, T_{i-1}^E labels are propagated to the corresponding segments in T_i^S and to their children recursively. Temporal stack overlap amount k is typically set to 1 but may be set to $k \geq 1$ depending on trajectory discontinuity tolerance (i.e., temporal loss of the object due to incorrect segmentation, occlusion, and so forth).

3.3.4 Results and Discussion

Many factors—such as length of the image sequence, size of the frames, density and size of the tracked objects—affect the execution time of the tracking program. Here we show performance of our parallel cell tracking program on two sample cell sequences with different characteristics shown in Table 3.8.

Seq-1 is a wound healing image sequence with relatively small 300×300 (40 μm × 40 μm) images but with high density of cells. The sequence has been obtained from phase-contrast microscopy using a monolayer of cultured porcine epithelial cells, as described by Salaycik et al. in [61]. Images were sampled uniformly over a 9:00:48 hour period and acquired using a phase contrast microscope, with a 10× objective lens, and at a resolution of approximately 0.13 μm per pixel. The original sequence consists of 136 frames, but for testing purposes, it has been replicated into a four times longer 544-frame sequence using a forward-backward-forward-backward pattern. Segmentation of the cells is done using our coupled 4-level set cell segmentation program described in Section 3.3.1. Seq-2 is a sample

Table 3.8 Properties of the Image Sequences Used in Parallel Tracking Performance Evaluation

	Seq-1	*Seq-2*
Sequence type	Wound healing	Human Melanoma
Image size	300 × 300	672 × 512
# Frames	544	395
# Objects detected	59,564	6,021
Average density	109 object/frame	15 object/frame

Table 3.9 Tracking Timing Results for 100 Frames from Each Sample Sequence

	Seq-1	*Seq-2*
1-Initializations	1.08%	3.87%
2-Object feature extraction	15.40%	75.21%
3-Correspondence analysis	73.91%	4.29%
4-Trajectory generation & analysis	1.56%	1.83%
5-Saving results	8.05%	14.80%

human melanoma cell sequence with lower density of cells but with larger 672 × 512 images. The pixel resolution is 0.67 μm × 0.59 μm and the images are obtained with a 20× objective lens. The sequence consists of 395 frames from one field in the T25 plastic culture flask of a control experiment with a low density culture of human melanoma cell line WM793 [92]. Cells are detected using flux tensors as described in Section 3.2.1. Segmentation is refined using the multifeature geodesic active contour level set algorithm as described in [46].

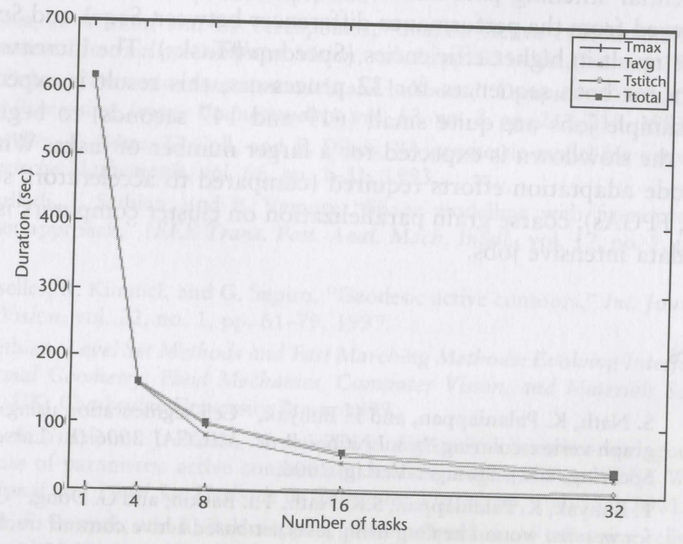
The differences in the characteristics of the two sample test sequences are reflected in the timing of the tracking program (Table 3.9). For Seq-1, which contains a higher number of objects per frame, the majority of the tracking time (73.91%) is spent in Step 3, correspondence analysis. Since the processed images are smaller in size, image processing done to compute object features takes relatively less time (15.40%). For Seq-2, which contains a smaller number of objects in larger images, the majority of the tracking time is spent in image processing done in Step 2, object feature extraction. Trajectory generation and analysis takes a low percentage of the tracking time for both sequences (1.56% and 1.83%) because this step consists of graph operations and does not involve expensive image processing operations. For both sequences, saving results takes considerable amounts of time (8.05% and 14.80%) because for further analysis, besides trajectories we save a number of object features including support maps and intensity distributions, which require

Table 3.10 Timing Results for Parallel Tracking Program

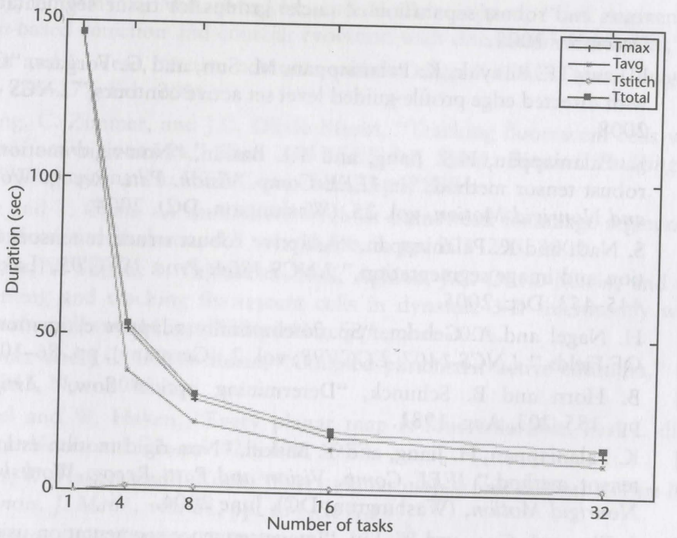
		# *Tasks*				
		1	*4*	*8*	*16*	*32*
Seq-1	$Task_{min}$	618.70	158.63	67.88	34.16	20.22
	$Task_{avg}$	618.70	159.00	83.19	44.35	24.99
	$Task_{max}$	618.70	159.81	99.17	56.05	30.89
	Stitch	0	2.01	2.65	3.84	6.44
	Total	618.70	161.82	101.82	59.89	37.33
	Speedup	1	3.82	6.07	10.33	16.57
Seq-2	$Task_{min}$	146.65	28.17	16.88	11.43	9.14
	$Task_{avg}$	146.65	38.36	22.30	14.31	10.31
	$Task_{max}$	146.65	51.76	29.63	18.10	12.96
	Stitch	0	1.68	1.63	1.73	2.23
	Total	146.65	53.44	31.26	19.83	15.19
	Speedup	1	2.74	4.69	7.39	9.65

large amounts of IO. If no further analysis will be done, by limiting the output to trajectories and trajectory related features, time spent in Step 5 can be greatly reduced.

Timing results and parallelization performance of the tracking algorithm on the two sample test sequences Seq-1 and Seq-2 are given in Table 3.10. As described



(a)



(b)

Figure 3.19 Timing of the parallel tracking algorithm for sample sequences: (a) Seq-1; and (b) Seq-2 (*y*-axes are scaled differently).

in Section 3.4.3, the original sequence is divided into M stacks, tracking of the objects in each stack is assigned to a task, and tasks are run independently in parallel on a cluster computer. Consistent labeling of the trajectories in time is ensured with a sequential stack-to-stack stitching step. Since stack-to-stack consistent trajectory labeling (or stitching) involves correspondence analysis on only $2 \times k \times (M-1) << N$ frames and since label propagation does not involve computationally expensive image processing operations, as seen in Table 3.10, the sequential stitching part does not introduce a considerable overhead. As can be observed from the performance differences between Seq-1 and Seq-2, higher workloads result in higher efficiencies (Speedup/#Tasks). The increase in speedup slows down for both sequences for 32 processors; this result is expected since both of the sample jobs are quite small (619 and 147 seconds) to begin with; for larger jobs the slowdown is expected for a larger number of tasks. With a small amount of code adaptation efforts required (compared to accelerators such as GPUs, Cell BEs, FPGAs), coarse grain parallelization on cluster computers is a good candidate for data intensive jobs.

References

[1] S. Nath, K. Palaniappan, and F. Bunyak, "Cell segmentation using coupled level sets and graph-vertex coloring," in *LNCS - Proc. MICCAI 2006* (R. Larsen, M. Nielsen and J. Sporring, eds.), Springer-Verlag, 2006.

[2] F. Bunyak, K. Palaniappan, S.K. Nath, T.I. Baskin, and G. Dong, "Quantitive cell motility for *in vitro* wound healing using level set-based active contour tracking," *Proc. 3rd IEEE Int. Symp. Biomed. Imaging (ISBI)*, (Arlington, VA), April 2006.

[3] A. Hafiane, F. Bunyak, and K. Palaniappan, "Clustering initiated multiphase active contours and robust separation of nuclei groups for tissue segmentation," *IEEE ICPR'08*, December 2008.

[4] I. Ersoy, F. Bunyak, K. Palaniappan, M. Sun, and G. Forgacs, "Cell spreading analysis with directed edge profile-guided level set active contours," *LNCS - Proc. MICCAI 2008*, 2008.

[5] K. Palaniappan, H.S. Jiang, and T.I. Baskin, "Non-rigid motion estimation using the robust tensor method," in *IEEE Comp. Vision. Patt. Recog. Workshop on Articulated and Nonrigid Motion*, vol. 25, (Washington, DC), 2004.

[6] S. Nath and K. Palaniappan, "Adaptive robust structure tensors for orientation estimation and image segmentation," *LNCS-3804: Proc. ISVC'05*, (Lake Tahoe, Nevada), pp. 445–453, Dec. 2005.

[7] H. Nagel and A. Gehrke, "Spatiotemporally adaptive estimation and segmentation of OF-Fields," *LNCS-1407: ECCV98*, vol. 2, (Germany), pp. 86–102, June 1998.

[8] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intell.*, vol. 17, pp. 185–203, Aug. 1981.

[9] K. Palaniappan, H. Jiang, and T. Baskin, "Non-rigid motion estimation using the robust tensor method," *IEEE Comp. Vision and Patt. Recog. Workshop on Articulated and Nonrigid Motion*, (Washington, DC), June 2004.

[10] J. Zhang, J. Gao, and W. Liu, "Image sequence segmentation using 3-D structure tensor and curve evolution," vol. 11, pp. 629–641, May 2001.

[11] H. Scharr, "Optimal filters for extended optical flow," *LNCS: First Int. Workshop on Complex Motion*, vol. 3417, (Berlin, Germany), pp. 66–74, Springer-Verlag, Oct. 2004.

[12] H. Scharr, I. Stuke, C. Mota, and E. Barth, "Estimation of transparent motions with physical models for additional brightness variation," *13th European Signal Processing Conference, EUSIPCO*, 2005.

[13] E. Dejnozkova and P. Dokladal, "Embedded real-time architecture for level set-based active contours," *EURASIP Journal on Applied Signal Processing*, no. 17, pp. 2788–2803, 2005.

[14] X. Wang, W. He, D. Metaxas, R. Matthew, and E. White, "Cell segmentation and tracking using texture-adaptive snakes," *Proc. 4th IEEE Int. Symp. Biomed. Imaging (ISBI)*, pp. 101–104, *IEEE Comp. Soc.*, April 2007.

[15] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Journal International Journal of Computer Vision*, vol. 1, pp. 321–331, January 1988.

[16] L. D. Cohen, "On active contour models and balloons," *Computer Vision, Graphics, and Image Processing. Image Understanding*, vol. 53, no. 2, pp. 211–218, 1991.

[17] V. Caselles, F. Catte, T. Coll, and F. Dibos, "A geometric model for active contours," *Numerische Mathematik*, vol. 66, pp. 1–31, 1993.

[18] R. Malladi, J. Sethian, and B. Vemuri, "Shape modelling with front propagation: A level set approach," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 17, no. 2, pp. 158–174, 1995.

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

[20] J.A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge, UK: Cambridge University Press, 1999.

[21] C. Xu, A. Yezzi, and J. Prince, "A summary of geometric level-set analogues for a general class of parametric active contour and surface models," *Proc. IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pp. 104–111, 2001.

[22] S. Nath, F. Bunyak, and K. Palaniappan, "Robust tracking of migrating cells using four-color level set segmentation," *LNCS - Proc. ACIVS 2006* (J. Blanc-Talon, D. Popescu, W. Philips and P. Scheunders, ed.), vol. 4179-0920, pp. 920–932, Springer-Verlag, 2006.

[23] I. Ersoy, F. Bunyak, M.A. Mackey, and K. Palaniappan, "Cell segmentation using Hessian-based detection and contour evolution with directional derivatives," 2008.

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

[25] B. Zhang, C. Zimmer, and J.C. Olivio-Marin, "Tracking fluorescent cells with coupled geometric active contours," *Proc. 2nd IEEE Int. Symp. Biomed. Imaging (ISBI)*, pp. 476–479, Arlington, VA: IEEE Comp. Soc., April 2004.

[26] L. Vese and T. Chan, "A multiphase level set framework for image segmentation using the Mumford and Shah model," vol. 50, no. 3, pp. 271–293, 2002.

[27] A. Dufour, V. Shinin, S. Tajbakhsh, N.G. Aghion, J.C. Olivio-Marin, and C. Zimmer, "Segmenting and tracking fluorescent cells in dynamic 3-D microscopy with coupled active surfaces," vol. 14, pp. 1396–1410, September 2005.

[28] C. Zimmer and J.C. Olivio-Marin, "Coupled parametric active contours," vol. 27, pp. 1838–1842, Nov. 2005.

[29] K. Appel and W. Haken, "Every planar map is four colorable. Part I. discharging," *Illinois. J. Math.*, vol. 21, pp. 429–490, 1977.

[30] K. Appel, W. Haken, and J. Koch, "Every planar map is four colorable. Part II. reducibility," *Illinois. J. Math.*, vol. 21, pp. 491–567, 1977.

[31] N. Robertson, D.P. Sanders, P.D. Seymour, and R. Thomas, "The four color theorem," *J. Combin. Theory, Ser. B*, vol. 70, pp. 2–44, 1997.

[32] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," vol. 22, no. 1, pp. 61–79, 1997.

[33] S.K. Nath and K. Palaniappan, "Adaptive robust structure tensors for orientation estimation and image segmentation," *Proc. 1st Int. Symp. Visual Computing (ISVC)* (G. Bebis, R. Boyle, D. Koracin and B. Parvin, ed.), vol. 3804, pp. 445–453, Lake Tahoe, NV: Springer, 2005.

[34] C. Li, C. Xu, C. Gui, and D. Fox, "Level set evolution without re-initialization: A new variational formulation," *IEEE Comp. Soc.*, vol. 1, pp. 430–436, 2005.

[35] S. Kim and H. Lim, "A hybrid level set segmentation for medical imagery," in *Proc. IEEE Nucl. Sci. Symp.*, vol. 3, pp. 1790–1794, IEEE Nucl. Plasma. Sci. Soc., 2005.

[36] R.F. Gonzalez, M.H. Barcellos-Hoff, and C. de Solórzano, "A tool for the quantitative spatial analysis of complex cellular systems," vol. 14, pp. 1300–1313, September 2005.

[37] C. Indermitte, T. Liebling, M. Troyanov, and H. Clemencon, "Voronoi diagrams on piecewise flat surfaces and an application to biological growth," *Theo. Comp. Sci.*, vol. 263, no. 1-2, pp. 263–274, 2001.

[38] S. Keenan, J. Diamond, W. McCluggage, H. Bharucha, D. Thompson, P. Bartels, and P. Hamilton, "An automated machine vision system for the histological grading of cervical intraepithelial neoplasia (CIN)," *J. Pathol.*, vol. 192, no. 3, pp. 351–362, 2000.

[39] J. Geusebroek, A.W.M. Smeulders, F. Cornelissen, and H. Geerts, "Segmentation of tissue architecture by distance graph matching," *Cytometry*, vol. 35, no. 1, pp. 11–22, 1999.

[40] B. Weyn, G. Wouwer, S.K. Singh, A. Daele, P. Scheunders, E. Marck, and W. Jacob, "Computer-assisted differential diagnosis of malignant mesothelioma based on syntactic structure analysis," *Cytometry*, vol. 35, no. 1, pp. 23–29, 1999.

[41] H. Choi, T. Jarkrans, E. Bengtsson, J. Vasko, K. Wester, P. Malmström, and C. Busch, "Image analysis based grading of bladder carcinoma. Comparison of object, texture and graph based methods and their reproducibility," *Anal. Cell. Path.*, vol. 15, no. 1, pp. 1–18, 1997.

[42] A. Klemenčič, F. Pernuš, and S. Kovačič, "Segmentation of muscle fibre images using voronoi diagrams and active contour models," vol. 3, pp. 538–542, *IEEE Comp. Soc.*, 1996.

[43] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu, *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams*, 2nd Edition, West Sussex, UK: John Wiley & Sons Ltd., 2000.

[44] F. Aurenhammer, "Voronoi diagrams - A survey of a fundamental geometric data structure," *ACM Comp. Surveys*, vol. 23, no. 3, pp. 345–405, 1991.

[45] *Lecture Notes in Computer Science (SCIA 2007)*, vol. 4522, 2007.

[46] I. Ersoy and K. Palaniappan, "Multi-feature contour evolution for automatic live cell segmentation in time lapse imagery," *IEEE EMBC'08*, 2008.

[47] K. Palaniappan, I. Ersoy, and S. K. Nath, "Moving object segmentation using the flux tensor for biological video microscopy," *Lecture Notes in Computer Science (PCM 2007)*, vol. 4810, 2007, pp. 483–493.

[48] F. Bunyak, K. Palaniappan, S. K. Nath, and G. Seetharaman, "Flux tensor constrained geodesic active contours with sensor fusion for persistent object tracking," *Journal of Multimedia*, August 2007.

[49] P. Yan, X. Zhou, M. Shah, and S. Wong, "Automatic segmentation of high throughput RNAi fluorescent cellular images," vol. 12, no. 1, pp. 109–117, 2008.

[50] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[51] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 777–786, 2004.

[52] S. Sengupta, M. Harris, Y. Zhang, and J. Owens, "Scan primitives for GPU computing," *Proceedings of the 22nd ACM SIGGRAPH EUROGRAPHICS Symposium on Graphics Hardware*, pp. 97–106, 2007.

[53] M. Harris, S. Sengupta, and J. Owens, "Parallel prefix sum (scan) with CUDA," in *GPU Gems 3* (H. Nguyen, ed.), Addison Wesley, 2007.

[54] http://www.seas.upenn.edu/~cis665/Schedule.htm.

[55] http://www.gpgpu.org.

[56] A. Lefohn, J. Kniss, and J. Owens, "Implementing efficient parallel data structures on GPUs," in *GPU Gems 2* (M. Pharr, ed.), pp. 521–545, Addison-Wesley, 2005.

[57] M. Harris, "Mapping computational concepts to GPUs," in *GPU Gems 2* (M. Pharr, ed.), pp. 493–508, Addison-Wesley, 2005.

[58] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker, "A streaming narrow-band algorithm: Interactive deformation and visualization of level sets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, pp. 422–433, July/August 2004.

[59] P. Labatut, R. Keriven, and J.-P. Pons, "A GPU implementation of level set multiview stereo"

[60] Y. Shi and W. Karl, "A real-time algorithm for the approximation of levelset-based curve evolution," *IEEE Transactions on Image Processing*, vol. 17, pp. 645–656, May 2008.

[61] G. Rong and T. Tan, "Jump flooding in GPU with applications to voronoi diagram and distance transform," *ACM Symposium on Interactive 3D Graphics and Games*, pp. 109–116, 2006.

[62] P. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980.

[63] P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions,"

[64] X. Chen, X. Zhou, and S.T.C. Wong, "Automated segmentation, classification, and tracking of cancer cell nuclei in time-lapse microscopy," vol. 53, pp. 762–766, April 2006.

[65] C. Hauwer, F. Darro, I. Camby, R. Kiss, P. Ham, and C. Decaestecker, "In vitro motility evaluation of aggregated cancer cells by means of automatic image processing," *Cytometry*, vol. 36, no. 1, pp. 1–10, 1999.

[66] D.P. Mukherjee, N. Ray, and S.T. Acton, "Level set analysis for leukocyte detection and tracking," vol. 13, pp. 562–672, April 2001.

[67] N. Ray and S.T. Acton, "Motion gradient vector flow: An external force for tracking rolling leukocytes with shape and size constrained active contours," vol. 23, pp. 1466–1478, Dec. 2004.

[68] R. Farooqui and G. Fenteany, "Multiple rows of cells behind an epithelial wound edge extend cryptic lamellipodia to collectively drive cell-sheet movement," *J. Cell Science*, vol. 118, pp. 51–63, Jan. 2005.

[69] F. Shen, L. Hodgson, A. Rabinovich, O. Pertz, K. Hahn, and J.H. Price, "Functional proteometrics for cell migration," *Cytometry Part A*, vol. 69A, pp. 563–572, June 2006.

[70] O. Debeir, P.V. Ham, R. Kiss, and C. Decaestecker, "Tracking of migrating cells under phase-contrast video microscopy with combined mean-shift processes," vol. 24, pp. 697–711, June 2005.

[71] X. Ronot, A. Doisy, and P. Tracqui, "Quantitative study of dynamic behavior of cell monolayers during *in vitro* wound healing by optical flow analysis," *Cytometry*, vol. 41, no. 1, pp. 19–30, 2000.

[72] D. Padfield, J. Rittscher, N. Thomas, and B. Roysam, "Spatio-temporal cell cycle phase analysis using level sets and fast marching methods," *Proc. 1st Work. Micro. Imag. Anal. App. Bio. (MIAAB)* (D. Metaxas, R. Whitaker, J. Rittscher and T. Sebastian, eds.).

[73] F. Bunyak and S.R. Subramanya, "Maintaining trajectories of salient objects for robust visual tracking," *Proc. 2nd Int. Conf. Image Anal. Recog. (ICIAR)* (M.S. Kamel and A.C. Campilho, eds.), vol. 3212, pp. 820–827, Toronto, ON: Springer, 2005.

[74] O. Kofahi, R. Radke, S. Goderie, Q. Shen, S. Temple, and B. Roysam, "Automated cell lineage construction: A rapid method to analyze clonal development established with murine neural progenitor cells," *Cell Cycle*, vol. 5, no. 3, pp. 327–335, 2006.

[75] K. Li, E.D. Miller, L.E. Weiss, P.G. Campbell, and T. Kanade, "Online tracking of migrating and proliferating cells imaged with phase-contrast microscopy," *Proc. IEEE Comp. Vis. Patt. Recog. Workshop (CVPRW-06)*, pp. 65–72, IEEE Comp. Soc., 2006.

[76] D. A. Randell, Z. Cui, and A. Cohn, "A spatial logic based on regions and connection," *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference* (B. Nebel, C. Rich, and W. Swartout, eds.), pp. 165–176, San Mateo, California: Morgan Kaufmann, 1992.

[77] A. G. Cohn and N. M. Gotts, "A theory of spatial regions with indeterminate boundaries," in *Topological Foundations of Cognitive Science* (C. Eschenbach, C. Habel, and B. Smith, eds.), 1994.

[78] M. Egenhofer and R. Franzosa, "Point-set topological spatial relations," *International Journal for Geographical Information Systems*, vol. 5, no. 2, pp. 161–174, 1991.

[79] E. Clementini, J. Sharma, and M. J. Egenhofer, "Modeling topological spatial relations: strategies for query processing," *Computers and Graphics*, vol. 6, pp. 815–822, 1994.

[80] D. Huttenlocher, D. Klanderman, and A. Rucklige, "Comparing images using the Hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 850–863, September 1993.

[81] J. Serra, "Hausdorff distances and interpolations," in *Mathematical morphology and its Applications to Image and Signal Processing (Proc. ISMM'98)*.

[82] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec. 2006.

[83] A. Y. Zomaya, *Parallel Computing for Bioinformatics and Computational Biology, (Wiley Series on Parallel and Distributed Computing)*, Wiley-Interscience, 2005.

[84] "The mathworks distributed computing toolbox," http://www.mathworks.com/products/distribtb/index.html.

[85] "Computer cluster," http://en.wikipedia.org/wiki/Computer_cluster.

[86] "University of Missouri Bioinformatics Consortium (UMBC)," http://umbc.rnet.missouri.edu.

[87] J. Gilbert, V. Shah, and S. Reinhardt, "A unified framework for numerical and combinatorial computing," *Computing in Science & Engineering*, vol. 10, pp. 20–25, March-April 2008.

[88] A. Krishnamurthy, J. Nehrbass, and S. S. J.C. Chaves, "Survey of parallel matlab techniques and applications to signal and image processing," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*, vol. 4, April 2007.

[89] D. Rivera, D. Schaa, M. Moffie, and D. R. Kaeli, "Exploring novel parallelization technologies for 3-d imaging applications," *International Symposium on Computer Architecture and High Performance Computing*, pp. 26–33, 2007.

[90] A. Gilbert and R. Bowden, "Incremental, scalable tracking of objects inter camera," *Computer Vision and Image Understanding*, vol. 111, p. 4358, 2008.

[91] S. Calderara, A. Prati, and R. Cucchiara, "Hecol: Homography and epipolarbased consistent labeling for outdoor park surveillance," *Computer Vision and Image Understanding*, vol. 111, pp. 21–42, 2008.

[92] P. Davis, E. Kosmacek, Y. Sun, F. Ianzini, and M. Mackey, "The large scale digital cell analysis system: An open system for non-perturbing live cell imaging," *J. Microscopy*, vol. 228, no. 3, p. 296308, 2007.

[93] N. Paragiois and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," vol. 22, pp. 266–280, March 2000.

[94] E.J. Cockayne and A.J. Thomason, "Ordered colourings of graphs," *J. Comb. Theory - Ser. B*, vol. 32, pp. 286–292, 1982.

[95] P. Felzenswalb and D. Huttenlocher, "Distance transforms of sampled functions," Tech. Rep. TR2004-1963, Dept. of Comp. Sci., Cornell University, Ithaca, NY, Sept. 2004.

[96] A. Rosenfeld, "A characterization of parallel thinning algorithms," *Inform. Control*, vol. 29, pp. 286–291, 1975.

[97] J.M. Cychosz, "Efficient binary image thinning using neighborhood maps," in *Graphics Gems IV*, pp. 465–473, San Diego, CA: Academic Press Professional, Inc., 1994.

[98] L. Lam, S. Lee, and C.Y. Suen, "Thinning methodologies—A comprehensive survey," vol. 14, pp. 869–885, Sept. 1992.

Today's bioimaging technologies generate mountains of biological data that can simply overwhelm conventional analysis methods. This groundbreaking book helps researchers blast through the computational bottleneck with high-performance computing (HPC) techniques that are blazing the way to never-before bioimaging, image analysis, and data mining capabilities, and revolutionizing the study of cellular and organ-level systems. It provides state-of-the-art methods and algorithms for building complex models from large biological datasets and solving data analysis and interpretation problems.

This innovative volume surveys the latest image acquisition knife-edge scanning microscopy, and 4D imaging of multicomponent biological systems. It introduces parallel processing for biological applications. Researchers learn advanced parallelization techniques for decomposing a problem domain and mapping it onto a parallel processing architecture using the message-passing interface (MPI) and OpenMP. Case studies show how these techniques have been successfully used in simulation tasks, data mining, and graphical visualization of biological datasets. This unique book also covers methods for developing scalable biological image databases and for facilitating greater interactive visualization of large image sets.

A. Ravishankar Rao is a research staff member with the Biometaphorical Computing Group at the IBM T.J. Watson Research Center, Yorktown Heights, New York. He is also an associate editor of the journals *Pattern Recognition* and *Machine Vision and Applications*. Dr. Rao received his B.Tech degree in electrical engineering from the Indian Institute of Technology (Kanpur), and his M.S. and Ph.D. degrees in computer engineering from the University of Michigan, Ann Arbor. He was named a master inventor at IBM Research in 2004.

Guillermo A. Cecchi is a research staff member with the Biometaphorical Computing Group at the IBM T.J. Watson Research Center, Yorktown Heights, New York. Dr. Cecchi received his M.Sc. in physics at the University of La Plata, Argentina, and his Ph.D. in physics and biology at The Rockefeller University. He also served as a postdoctoral fellow at Cornell University.

ISBN-13: 978-1-59693-295-1
ISBN-10: 1-59693-295-3

ARTECH HOUSE
BOSTON | LONDON
www.artechhouse.com