Cell Segmentation Using the Level Set Method

L. He, B. Heise, A. Kuijper, Y. Zhou

RICAM-Report 2007-17
Cell Segmentation Using the Level Set Method

Y. Zhou
Mathematical Engineering - CT PP 2
Siemens AG
Otto-Hahn-Ring 6
D-81730 Munich, Germany

A. Kuijper
Johann Radon Institute
for Computational and Applied Mathematics
Altenberger Strasse 69
A-4040 Linz, Austria

B. Heise
Department of Knowledge-Based Mathematical Systems
Johannes Kepler University
Altenberger Strasse 69
A-4040 Linz, Austria

L. He
Johann Radon Institute
for Computational and Applied Mathematics
Altenberger Strasse 69
A-4040 Linz, Austria
# Table of Contents

Table of Contents 3

Abstract 9

Introduction 11

1 Traditional Level Set Method 15
   1.1 Formulation of Interface Propagation 15
      1.1.1 Implicit Function and Signed Distance Function 16
      1.1.2 An Initial Value Formulation 18
   1.2 Theoretical Analysis of Curve Evolution 19
      1.2.1 Stability 20
      1.2.2 Weak Solution and Entropy Condition 21
      1.2.3 Hamilton-Jacobi Equation and Viscosity Solutions 22
   1.3 Classical Level Set Method in Image Segmentation 23
   1.4 Numerical Implementation 24
   1.5 Variational Formulation 25
   1.6 Application in Cell Segmentation 26

2 Clustered Cell Segmentation Based on Iterative Voting and the Level Set Method 29
   2.1 Seed Detection 29
      2.1.1 Iterative Voting 31
      2.1.2 Result for Seed Detection 33
      2.1.3 Parameter Setting and Operation Time 36
   2.2 Level Set Method for Clustered Cell Image 38
      2.2.1 Initial Expansion Level Set Function 38
      2.2.2 Free Expansion Level Set Function 39
      2.2.3 Surface Wrapping Level Set Function 40
      2.2.4 Numerical Implementation 40
      2.2.5 Parameter Setting and Operation Time 42
      2.2.6 Numerical Result for Level Set Method 43
3 Multiphase Level Set Method and its Application in Cell Segmentation

3.1 Active Contour Model Without Edges
   3.1.1 Formulation of Level Set Equation
   3.1.2 Numerical Implementation and Result

3.2 Multiphase Level Set Method
   3.2.1 Formulation of Multiphase Level Set Method
   3.2.2 Numerical Implementation and Application in Cell Segmentation

3.3 A Preprocessing Method and its Application

4 Conclusion and Future Work

4.1 Comments and Conclusion

4.2 Future Work

Bibliography
List of Tables

2.1 Parameter Setting for Seeds Detection Part . . . . . . . . . . . . . . . . . 37
2.2 Operation Time for Seeds Detection Part . . . . . . . . . . . . . . . . . 38
2.3 Parameter Setting for Level Set Method . . . . . . . . . . . . . . . . . 43
2.4 Operation Time for Level Set Method . . . . . . . . . . . . . . . . . . 44
List of Figures

1.1 Curve Propagating with Speed F in Normal Direction . . . . . . . . . . 16
1.2 A Partition for the Domain Ω . . . . . . . . . . . . . . . . . . . . . . 18
1.3 Topological Changes in the Level Set Method . . . . . . . . . . . . . . 19
1.4 Cosine Curve Propagating with Unit Speed . . . . . . . . . . . . . . . 22
1.5 Application for Two Separated Cells . . . . . . . . . . . . . . . . . . 27
1.6 Application for Clustered Cells . . . . . . . . . . . . . . . . . . . . . 28

2.1 Shape of Kernel used in the Method . . . . . . . . . . . . . . . . . . . 31
2.2 Seed Detection Result for Test Image I . . . . . . . . . . . . . . . . . 35
2.3 Seed Detection Result for Test Image II . . . . . . . . . . . . . . . . . 35
2.4 Seed Detection Result for Test Image III . . . . . . . . . . . . . . . . 36
2.5 Result for Test Image I . . . . . . . . . . . . . . . . . . . . . . . . . . 45
2.6 Result for Test Image II . . . . . . . . . . . . . . . . . . . . . . . . . 46
2.7 Result for Test Image III . . . . . . . . . . . . . . . . . . . . . . . . . 46
2.8 Histogram for Test Images . . . . . . . . . . . . . . . . . . . . . . . . 47

3.1 Comparison of 2 Approximation Functions . . . . . . . . . . . . . . . . 52
3.2 Result for Chan-Vese Model . . . . . . . . . . . . . . . . . . . . . . . . 53
3.3 2 Curves Partition the Domain into 4 Regions . . . . . . . . . . . . . . 54
3.4 Cell Segmentation Using Multiphase Chan-Vese Model . . . . . . . . . 57
3.5 Property of Perona Malik Functions . . . . . . . . . . . . . . . . . . . 59
3.6 Segmentation Result without Preprocessing . . . . . . . . . . . . . . . 62
3.7 Segmentation Result with Preprocessing . . . . . . . . . . . . . . . . . 62
Abstract

In this work, the level set method and its application in cell segmentation are investigated. The main task is to segment images with clustered cells provided by the Fuzzy Logic Laboratorium Linz-Hagenberg, part of the Department of Knowledge-Based Mathematical Systems at the Johannes Kepler University, Linz.

Starting with the traditional level set method, the formulation of the level set function and its property are introduced in Chapter 1. The traditional method appears suitable for segmenting cells that don’t stick together. In Chapter 2, a multiphase level set method is set up to segment images with clustered cells. Combined with the iterative voting method, the multiphase level set method can separate clustered cells into individual cells and calculate the area of each cell. In Chapter 3, the Chan-Vese Model is introduced. Instead of solving $n$ PDEs (Partial Differential Equations) as in the previous method, this model only needs two PDEs.

All models introduced in this work are implemented and tested with real cell images. Based on these tests, we draw the conclusion: the multiphase level set method combined with iterative voting method is best suited for segmenting given images with clustered cells. The drawbacks of this method are the time complexity and the space complexity. The Chan-Vese model reduces the operation time and can deal with images that do and do not contain edges. However, it has some limitations in cell segmentation because of the sub-structures present inside cells. A pre-processing method is proposed in this work; better suited pre-processing methods are still under research.
This report describes the work carried out by Yayun Zhou, while she was at the Johannes Kepler University. It was part of the MSc. requirements of the study Industrial Mathematics and supervised by Arjan Kuijper. Valuable input and feedback were given by Bettina Heise (Chapter 2) and Lin He (Chapter 3).
Introduction

Cell segmentation is a popular topic in image analysis. Typically, the population of cells in one image is large. If we want to count the number of the cells, or study the property of certain cell, cell segmentation is necessary and important. In general, a reliable segmentation is hard because images are noisy (both random and speckle noise). Sometimes many cells will cluster together and even overlap in the sample. This complicated situation makes the problem more challenging and interesting. Therefore, a lot of people are working on algorithms for Computer-Based cell segmentation of microscope images. There is a wide variety of approaches focusing on different segmentation problems.

Traditional Methods in Image Segmentation

Nowadays, one of the popular approaches is active contour models or snakes, first introduced by Terzopoulos and Kass [11]. The basic idea in active contour models (or snakes) is to evolve a curve, subject to constraints from a given image $u_0$, in order to detect objects in that image. Usually, the model contains an edge detector, which depends on the gradient of the image $u_0$. The initial curve is around the object to be detected, then the curve moves in the normal dimension to itself and ideally stops at the boundary of the object. The main drawbacks of the original snakes are their sensitivity to initial conditions and the difficulties associated with topological transformations.

Another famous model in image segmentation is formulated by Mumford and Shah [17]. In their paper they proposed a minimization problem, the minimizer for the
functionals is the optimal piecewise smooth approximation $u$ for the given image $u_0$. In practice, solving the Mumford and Shah model is not an easy job, because the functional involves an unknown set $C$ of lower dimension and usually the problem is not convex.

**The Level Set Method in Image Segmentation**

In this work, we focus on the level set method. The level set method is a numerical and theoretical tool for propagating interfaces. It was first introduced by Osher and Sethian [19], [27], and has become a more and more popular theoretical and numerical framework within image processing, fluid mechanics, graphics, computer vision, etc. The level set method is basically used for tracking moving fronts by considering the front as the zero level set of an embedded function, called the level set function. In image processing, it is used for propagating curves in 2D or surfaces in 3D. The applications of the level set method cover most fields in image processing, such as noise removal, image inpainting, image segmentation and reconstruction.

In image segmentation, the level set method has some advantages compared to the active contour model. The level set method conquers the difficulties of topological transformations. The level set approach is able to handle complex topological changes automatically.

The traditional level set method depends on the gradient of the given image to stop the curve evolution. Therefore, it has some drawbacks. Later, some variational level set methods are developed. In a sequence of papers beginning with Chan and Vese [5], [6], the authors propose a different active contour model that does not use the gradient of the image $u_0$ for the stopping process. The stopping term is based on the Mumford-Shah functional [17] for segmentation. The Chan-Vese model can detect contours both with and without gradients. In addition, by using this model and its level set formulation, interior contours are automatically detected, and the initial curve can be anywhere in the image. The liberty of formulation of these level set methods gives us countless possibilities. In this work we have explored some of the
numerous aspects of the level set framework to see how the different image properties can be used for segmentation.

**This Work**

The paper is organized as follows. In the first chapter the traditional level set method and its application in image segmentation is introduced. This method works well in segmenting objects which have clear boundary or contrast with background. Therefore, if the cells do not cluster together and the boundary of cells is complete, it is acceptable to use the traditional level set method.

In the second chapter, we deal with the images in which the cells cluster together and the boundary of the cells are ambiguous. Combining the outcome of automatic point detector with the multiphase level set method, the center of each cell is detected and used as the "seed", in other words, the initial condition for level set method. Then by choosing appropriate level set equation, the fronts of the seeds propagate and finally stop near the boundary of the cells. This method solves the cluster problem and can distinguish individual cells properly, therefore it is useful in cell segmentation. By using this method, we can count the number of the cells and calculate the area of each cell. Furthermore, this information can be used to get the histogram of the cell image. The drawback of this method is the efficiency and speed. It requires a lot of memory and time to solve the PDE for each cell separately. The resource demand is especially obvious when the number of cells increases.

In the third chapter, a multiphase level set method is introduced. Based on the Four-Color Theorem, only two level set equations formally suffice to represent any partition. Motivated by the article of Chan and Vese [30], this method is much faster and more stable than the previous algorithms. Instead of solving $n$ PDEs for $n$ cells, this method only deals with two PDEs. The initial condition for this method is not strict, so the iterative voting part is removed. This model is useful to distinguish different cells. However, this method has some limitations on images with clustered cells
containing sub-structures inside. Certain preprocessing methods can release the limitation. In this work, the Perona Malik filter is chosen. Since the current result is not yet satisfactory enough, other suitable preprocessing methods are still under research.

The fourth chapter contains comments and conclusion. The performances of current models are evaluated and compared, and the future work is proposed.
Chapter 1

Traditional Level Set Method

The level set method is a powerful numerical technique for tracking the moving interfaces. Since it was developed first by Osher and Sethian [19], [27], it has been applied in many fields like physics, chemistry, fluid mechanics, image processing, computer vision and a collection of other areas. In image processing, the level set method has attained outstanding achievements in recent years. It can be used in noise removal, image inpainting, image segmentation and reconstruction. In this chapter, we start with the formulation of interface propagation and introduce the classical level set method in image segmentation. Then the numerical implementation of solving the classical model and its application in cells segmentation will be discussed.

1.1 Formulation of Interface Propagation

In this section, we formulate the boundary value and initial value PDEs that describe the interface motion. As a starting point, I consider a boundary, either a curve in 2D or a surface in 3D, which separates one region from another. In Fig. 1.1, we take the 2D case as an example and in the following chapters, we only discuss the 2D case, which can be extended to 3D case easily.

Assume that the curve moves in a normal direction with a speed function F, where F may depend on local properties (such as curvature and normal direction), global
properties of the front (such as integrals along the front or associated differential equations), and independent properties (such as underlying fluid velocity). We will introduce the definition of the implicit function, which can be used to describe the interface, then we will derive the formulation of the interface propagation.

Figure 1.1: Curve Propagating with Speed $F$ in Normal Direction, where $L = \text{Local properties}$, $G = \text{Global properties of the fronts}$, $I = \text{Independent properties}$

1.1.1 Implicit Function and Signed Distance Function

In order to set up the mathematical model for this problem, the moving interface needs to be parameterized. The novelty of level set method is that it introduces the high dimensional function $\phi$ to represent the interface implicitly. The idea of implicitly representing interface is as follows [18]: In an explicit interface representation, one explicitly writes down the points that belong to the interface. In an implicit interface representation, one usually defines the interface as an iso-contour of some function.

In 1D space, if we divide the real line into three distinct pieces using points $x =$
−1 and \( x = 1 \), then there are three separate sub-domains: \((-\infty, -1)\), \((-1, 1)\), and \((1, \infty)\). We refer to \( \Omega^- = (-1, 1) \) as the *inside* portion of the domain and \( \Omega^+ = (-\infty, -1) \cup (1, \infty) \) as the *outside* portion of the domain. The border between them, i.e \( \partial \Omega = \{-1, 1\} \), is called the *interface*. In the explicit way, the interface is represented as a set \( \{-1, 1\} \). Alternately, the interface can also be represented implicitly as the iso-contour of some function, for example, the zero contour of function \( \phi(x) = x^2 - 1 \).

Similarly, in two spatial dimensions, we can also define implicit function \( \phi \) to represent the interface. Since the interface is a curve containing infinite points, usually, the function needs to be approximated by discretization. The set of data points defining the implicit function \( \phi \) is called a *grid*. More generally in \( \mathbb{R}^n \), the implicit function \( \phi(\overline{x}) \) is defined on all \( \overline{x} \in \mathbb{R}^n \), and its iso-contour has dimension \( n - 1 \).

There are various implicit functions which can be used to represent interface. In particular, we use a *Signed Distance Function* to implicitly represent the interface. The definitions of a *Distance function* and a *Signed Distance Function* are stated below [18]:

**Definition 1.1.1.** A distance function \( d(\overline{x}) \) is defined as

\[
d(\overline{x}) = \min(|\overline{x} - \overline{x}_I|) \quad \text{for all } \overline{x}_I \in \partial \Omega.
\]

**Definition 1.1.2.** A signed distance function is an implicit function \( \phi \) with \( |\phi(\overline{x})| = d(\overline{x}) \) for all \( \overline{x} \). Thus, \( \phi(\overline{x}) = d(\overline{x}) = 0 \) for all \( \overline{x} \in \partial \Omega \), \( \phi(\overline{x}) = -d(\overline{x}) \) for all \( \overline{x} \in \Omega^- \), and \( \phi(\overline{x}) = d(\overline{x}) \) for all \( \overline{x} \in \Omega^+ \).

Therefore we can represent the interface as well as our domain partition by the notations defined above:

\[
\begin{align*}
\Phi &< 0 \quad \text{if } x \in \Omega^- \\
\Phi &> 0 \quad \text{if } x \in \Omega^+ \\
\Phi &= 0 \quad \text{if } x \in \partial \Omega
\end{align*}
\]  
(1.1.1)
Thus, the region $\Omega^-$ is related to inside and the region $\Omega^+$ is related to outside, the interface between them is represented by the function $\phi = 0$, which is also called zero level set function.

### 1.1.2 An Initial Value Formulation

In previous subsection, we presented the representation for the interface, which is denoted as the zero iso-contour of a function $\phi$. We can link the evolution of this function $\phi$ to the propagation of the front itself through a time-dependent initial value problem. At any time, the front is given by the zero level set of the time-dependent level set function $\phi$. Then we get the following equation:

$$\phi(x(t), t) = 0, \quad (1.1.2)$$

Taking the time derivative for both sides of the equation, by the chain rule, we get

$$\phi_t + \nabla \phi(x(t), t) \cdot x'(t) = 0, \quad (1.1.3)$$

Since $F$ is defined as the speed in the outward normal direction, then $x'(t) \cdot n = F$, where $n = \nabla \phi / |\nabla \phi|$. This yields an evolution equation for $\phi$:

$$\phi_t + F|\nabla \phi| = 0, \text{ given } \phi(x, t = 0). \quad (1.1.4)$$
This is the level set equation given by Osher and Sethian [19].

By using this equation, the level set method can handle topological changes naturally. Fig. 1.3 illustrates the topological change phenomenon [32]. The red surface in the figure is the graph of a level set function $\phi$, and the flat blue region represents the x-y plane. The top row shows a group of bounded regions with a well-behaved boundary. The boundary of the shape is the zero level set of $\phi$. The shape changes topology by splitting in two. It is quite hard to describe this transformation numerically by parameterizing the boundary of the shape and following its evolution. But if one uses a level set function to describe the evolution, it is much easier than working with the shape directly. The PDE defined in Eq. (1.1.4) can be solved numerically.

![Figure 1.3: Topological Changes in the Level Set Method](image)

1.2 Theoretical Analysis of Curve Evolution

After having formulated the equation of motion for a propagating curve in the previous section, we will study the stability and introduce the entropy-satisfying weak solution
in this section.

1.2.1 Stability

We first rewrite Eq. (1.1.4) in terms of individual components \( \vec{x} = (x, y) \) [27]. Here the speed function \( F \) is restricted to depend only on the local curvature \( \kappa \) of the curve:

\[
\begin{align*}
x_t &= F(\kappa) \left( \frac{y_s}{(x_s^2 + y_s^2)^{1/2}} \right), \\
y_t &= -F(\kappa) \left( \frac{x_s}{(x_s^2 + y_s^2)^{1/2}} \right),
\end{align*}
\]

where \( \kappa = \frac{y_s x_s - x_s y_s}{(x_s^2 + y_s^2)^{3/2}} \) is the curvature, and \( s \) is the parameter which parameterizes the curve, \( t \) is the time: the evolving curve can be denoted as \( \vec{x}(s, t) = (x(s, t), y(s, t)) \), where \( 0 < s < S \) and \( \vec{x}(0, t) = \vec{x}(S, t) \). \( x_s, y_s \) is the first order derivative of \( x, y \) with respect to \( s \), and \( x_{ss}, y_{ss} \) is the second order derivative of \( x, y \) with respect to \( s \). This formulation is also called "Lagrangian" representation.

The stability is measured by the Total Variation. Recall that the metric \( g(s, t) = (x_s^2 + y_s^2)^{1/2} \) measures the "stretch" of the parameterization. Defined the total variation of the front as below [27]:

**Definition 1.2.1.** The total variation of the front is defined as

\[
\text{Var}(t) = \int_0^S |\kappa(s, t)| g(s, t) ds.
\]

The Total Variation measures the amount of "wrinkling". If the total variation decreases as the front evolves and the front "smoothes out", the energy of the front dissipates and the flow is stable. Otherwise, it is not. Sethian [26] has proved that the stability is only depending on the sign of \( F(\kappa) \) at \( \kappa = 0 \). There is a lemma stating that:

**Lemma 1.2.1.** Consider a front moving along its normal vector field with speed \( F(\kappa) \), as in Eq. 1.1.4. Assume that the evolving curve is simple, three time differentiable for \( 0 \leq s \leq S \) and \( 0 \leq t \leq T \), and non-convex, so that \( \kappa(s, 0) \) changes sign. Assume that \( F \) is twice differentiable. Then, for \( 0 \leq t \leq T \),
• if $F_\kappa(0) \leq 0$ ($F_\kappa(0) \geq 0$), then
  \[ \frac{d\text{Var}(t)}{dt} \leq 0, \quad \left( \frac{d\text{Var}(t)}{dt} \geq 0 \right), \]

• if $F_\kappa(0) < 0$ ($F_\kappa(0) > 0$) and $\kappa_s(0) \neq 0$, then
  \[ \frac{d\text{Var}(t)}{dt} < 0, \quad \left( \frac{d\text{Var}(t)}{dt} > 0 \right). \]

The proof of this lemma can be found in [26], based on this lemma, we can construct a speed function $F(\kappa) = 1 - \epsilon \kappa$, for $\epsilon > 0$, the derivative $F_\kappa = -\epsilon$ and hence the total variation decays.

### 1.2.2 Weak Solution and Entropy Condition

In the previous lemma, there is an assumption that the front stays smooth, but in many cases, the smoothness is soon lost. A famous example is the cosine curve propagating with unit speed $F(\kappa) = 1$. The initial cosine curve is defined as $\gamma(0) = (1 - s, [1 + \cos 2\pi s]/2)$, then the exact solution of this problem can be written by the equations below:

\[
\begin{align*}
x(s, t) &= \frac{y_s(s, t = 0)}{(x_s(s, t = 0) + y_s(s, t = 0))^{1/2}} t + x(s, t = 0), \\
y(s, t) &= \frac{-x_s(s, t = 0)}{(x_s(s, t = 0) + y_s(s, t = 0))^{1/2}} t + y(s, t = 0).
\end{align*}
\]

Fig. 1.4, taken from Sethian [27], shows that the front develops a sharp corner in finite time. Once the corner develops, the differentiability is lost. Thus, we need a weak solution to beyond the formation of the discontinuity in the derivative. The figure shows two possibilities of weak solution for this problem, one is called "swallowtail", which is generated by letting the front pass through itself, the other is called "entropy solution", which can be viewed as simply remove the "tail" from "swallowtail". The entropy solution is constructed through Huygen’s principle. Another way to obtain the entropy solution is through the notion of an entropy condition posed by Sethian in [25], [26]. The weak solution given by the entropy condition is the weak solution we want, it satisfies most physical phenomena.
1.2.3 Hamilton-Jacobi Equation and Viscosity Solutions

The velocity field $F$ in Eq. (1.1.4) can be generated externally. For example, the self-generated velocity $F$ depends directly on the position of $x$ and the first order derivatives of the level set function $\phi$. In this case, we get the Hamilton-Jacobi equation in the following form:

$$\alpha u_t + H(Du, x) = 0,$$

where $H(Du, x) = F|\nabla u| - (1 - \alpha)$, and $\alpha$ is either zero or one. For this equation, Crandall, Evans and Lions [16] define a weak solution as follows:

**Definition 1.2.2.** A function $u$ is said to be a viscosity solution of Eq. (1.2.2), if for all smooth test function $v$,

- if $u - v$ has a local maximum at a point $(x_0, t_0)$, then

$$v_t(x_0, t_0) + H(Dv(x_0, t_0), x_0) \leq 0,$$  

(1.2.3)

- if $u - v$ has a local minimum at a point $(x_0, t_0)$, then

$$v_t(x_0, t_0) + H(Dv(x_0, t_0), x_0) \geq 0.$$  

(1.2.4)
This viscosity solution has some nice properties which enable it to be a reasonable solution for Hamilton-Jacobi Equations. It has been proven that any classical smooth solution is a viscosity solution and the solution produced by taking the limit of the smooth solution \( u_\epsilon \) as \( \epsilon \) goes to zero is the viscosity solution. The viscosity solution is unique, given appropriate initial condition. Precise statement and proof can be found in [7], [8], [9].

1.3 Classical Level Set Method in Image Segmentation

The level set method defined in Eq. (1.1.4) can be extended to set up a mathematical model for image segmentation. Consider a speed function of the form \( F = \pm 1 - \epsilon \kappa \), where \( \epsilon \) is a constant. The uniform form term \( \pm 1 \) determines the direction of the curve evolution: \( +1 \) means the curve will move outwards and \( -1 \) means the curve will move inwards. The diffusive second term \( \epsilon \kappa \) smoothes out the high curvature regions. Then the above speed function is multiplied with \( g_I \) defined by:

\[
g_I(x, y) = \frac{1}{1 + |\nabla(G_\sigma * I(x, y))|_p}, \quad p \geq 1.
\]  

(1.3.1)

Here the expression \( G_\sigma * I \) denotes the image convolved with a Gaussian smoothing filter whose characteristic width is \( \sigma \). The general formula for Gaussian kernel is stated as: \( G_\sigma(x) = \frac{1}{(\sqrt{2\pi}\sigma)^d} e^{-\frac{x^2}{2\sigma^2}} \). The term \( \nabla(G_\sigma * I(x, y)) \) is essentially zero except where the image gradient changes rapidly, in which case the value becomes very large. Thus, after multiplying this edge detection function \( g_I \), the speed function will become zero when the front evolves near to the boundary. This means that the evolution will stop when it approaches the boundary of objects.

There are various models based on this idea [3], [4], [15], [16], which are slightly different from each other: in some cases \( p = 1 \) and in some cases \( p = 2 \); there might be some other terms added in the above equation to increase the stability or to enhance the boundary. One typical model is the following [27]:

\[
\frac{\partial \phi}{\partial t} + g_I \cdot (1 - \epsilon \kappa)|\nabla \phi| - \beta \nabla P \cdot \nabla \phi = 0.
\]  

(1.3.2)
This equation contains three terms:

- A driving expansion force \( F_{\text{expand}}(x) = g_I(x) = \frac{1}{1 + |\nabla(G\sigma*I(x,y))|} \). Here we notice that the curve expands outwards.

- A surface tension force which depends on the curvature \( F_{\text{curv}}(x) = -g_I(x) \cdot \epsilon \kappa \). Here we can check the sign of \( F_{\kappa} \), and see that it satisfies the stability requirements.

- A force attracting the front towards the boundary, which has a stabilizing effect. Here \( P = -|\nabla(G\sigma*I(x))| \). The coefficient \( \beta \) controls the strength of this attraction.

### 1.4 Numerical Implementation

To solve the Eq. (1.1.4), we need to discretize the domain and apply an appropriate finite difference method. Naively applying central difference approximations to the Lagrangian form of Eq. (1.2.1) will cause oscillation, unless the time step is small enough. There is no time step that can produce a scheme which correctly incorporates the entropy condition [27]. Therefore, in practice, the Lagrangian form is usually abandoned, instead there are some numerical schemes solving the Eulerian formulation defined in Eq. (1.1.4).

We notice that there are some similarities and links between the Hamilton-Jacobi equation and hyperbolic conservation laws [27]. For hyperbolic conservation laws, there are many numerical methods which can be applied to solve the PDEs, such as Godunov’s Method, Roe’s Approximate Riemann Solver, the Enquist-Osher Method, and so on; details can be found in [36]. These methods have been proven to be convergent to the entropy-condition satisfying weak solution. Based on these numerical schemes, the algorithms for solving the traditional level set method are developed. Taking Eq. (1.3.2) as an example, the term \( g_I \cdot |\nabla \phi| \) describes the motion in the normal direction. It must be approximated through the entropy-satisfying schemes discussed above. The term \( \beta \nabla P \cdot \nabla \phi \) corresponds to pure passive advection. This
term may be approximated through simple upwind schemes. The term $g_I \cdot \epsilon \kappa |\nabla \phi|$ is a parabolic contribution to the equation. The most straightforward approach is to use central difference approximations.

When implementing the traditional level set methods, it is numerically necessary to keep the evolving level set function close to a signed distance function \cite{18}, \cite{29}. Reinitialization, a technique for periodically re-initializing the level set function to a signed distance function during the evolution, is used for maintaining stable curve evolution.

### 1.5 Variational Formulation

As an alternative, the evolution PDE of the level set function can be derived directly from the problem of minimizing a certain energy functional defined on the level set function. This type of variational methods are known as variational level set methods \cite{12}, \cite{13}, \cite{24}. The variational level set methods are more convenient for incorporating additional information. As we stated before, reinitialization is a necessary step in numerical implementation for traditional level set methods. Using variational level set method, the reinitialization phase can be embedded in one equation and remove the reinitialization step.

Here we choose a variational level set method model proposed by Li et al. \cite{12} to apply in cell segmentation. They define the following functional:

$$
\varepsilon(\phi) = \mu P(\phi) + \varepsilon_m(\phi),
$$

where

$$
\varepsilon_m(\phi) = \lambda \int_{\Omega} g \delta(\phi) |\nabla \phi| dx dy + \nu \int_{\Omega} g H(-\phi) dx dy
$$

is the external energy; $g = \frac{1}{1 + |\nabla G_{\sigma} + I|^2}$ is the edge indicator function; $\delta$ is the univariate Dirac function, and $H$ is the Heaviside function. $\lambda$ and $\nu$ are constants, where $\lambda > 0$. $P(\phi) = \int_{\Omega} \frac{1}{2} (|\nabla \phi| - 1)^2 dx dy$ is an internal energy term whose task is to remove the reinitialization step. $\mu > 0$ is a parameter controlling the effect of penalizing the deviation of $\phi$ from a signed distance function. The steepest descent process for minimizing this functional
is the following gradient flow:

$$\frac{\partial \phi}{\partial t} = \mu[\Delta \phi - \text{div}\left(\frac{\nabla \phi}{|\nabla \phi|}\right)] + \lambda \delta(\phi)\text{div}\left(g\frac{\nabla \phi}{|\nabla \phi|}\right) + \nu g\delta(\phi).$$  \tag{1.5.2}

This PDE can be solved by applying central difference scheme for spatial partial derivatives and forward difference scheme for temporal partial derivatives. The initialization is not necessarily a signed distance function, but can be arbitrary functions.

### 1.6 Application in Cell Segmentation

Usually, the traditional level set method only works when the boundary is clear and complete because of the boundary detector term. The variational formulation defined in [12] can deal with the image with weak boundaries. Fig. 1.5 shows the evolution of the contour on a 64 × 80 - pixel microscope image of two cells. Sub-figure (a) shows the initial level set function \(\phi_0\) (the red contour). Sub-figure (b) and (c) show the evolution process of the level set function, we can see that the topological change happens after several iteration steps. Finally, in Sub-figure (d), two cells are segmented and the boundary of the cells have been detected accurately\(^1\).

However, this model can only segment cells whose locations are far away from each other. If the cells are clustered together, this model will fail to segment individual cells. We can see the result from Fig. 1.6. After detecting the boundary of the cell cluster, the front stops moving, no matter how many iteration steps are used, it is impossible to segment individual cells from the group.

If the cells are close to each other but don’t stick together, decreasing the length term may get better result than Fig. 1.6. If the boundaries of cells merge together, only using one level set equation is not sufficient to separate individual cells. We will introduce a multiphase level set method in the next chapter, which is designed especially to segment images of clustered cells.

\(^1\)The matlab code is open source from matlab central: "http://www.mathworks.com". The test image in Fig. 1.5 is also from matlab central.
Figure 1.5: Application for Two Separated Cells:  
(a) The initial contour; (b) 50 iterations; (c) 150 iterations; (d) 300 iterations
Figure 1.6: Application for Clustered Cells: (a) The initial contour; (b) 100 iterations; (c) 1000 iterations; (d) 2000 iterations
Chapter 2

Clustered Cell Segmentation Based on Iterative Voting and the Level Set Method

In the previous chapter, we introduced the traditional level set method and its application in cell segmentation. For images with clustered cells, this model is not suitable for segmenting individual cells. In this chapter, we will introduce a method designed especially for these kind of images. The main idea of this method is using an iterative voting method to detect the center of each cell. The detected center will be used to define the initial condition for the level set method. A sequence of level set functions are set up for each cell. Combining with the initial condition, the PDEs for each cell can be solved numerically. This method is tested on different cell images. The performance is stable. As long as the cells are distinguishable by human eyes, the algorithm can segment individual cells properly.

2.1 Seed Detection

The first step for this segmentation method is detecting the center of cells. Since the detected center will be used as the initial condition (“seed”) for the level set method, this step is very important for the performance of the whole method. The number of detected seeds determines the number of cells which can be detected. If there are some cells without seeds, it means that these cells will not be counted in
as candidates. If there are two seeds in one cell, it may cause over-segmentation (a cell is divided into two cells). In brief, the accuracy of the first step will affect the performance of the level set methods.

For the given images in the test data base, the shape of cells is almost elliptic. Therefore, the Hough Transform is a possible choice for seeds detection [1], [21], [33]. The main idea of the Hough Transform is the parameter transform and voting method. The detected feature can be represented by several parameters. For example, if we want to detect a line, the two parameters determining a line are length $r$ and angle $\theta$ from the origin of a normal to the line: an equation of line can be written as $r = x \cos \theta + y \sin \theta$; if we want to detect a circle, the three parameters determining a circle are its center $(x_0, y_0)$ and radius $r$: an equation of circle can be written as $(x - x_0)^2 + (y - y_0)^2 = r^2$.

If we want to detect the center of a circle using Hough Transform, first we need an edge detector to detect the points on the boundary. Then for each point on the boundary, we can try to draw circles passing through this point with different parameters. The transform itself is quantized into an arbitrary number of bins, each representing an approximate definition of a possible circle. Each point in the edge detected image is said to vote for a set of bins corresponding to the circles that pass through it. By simply increasing the value stored in each bin for every point lying on that circle, an array is built up showing which circle fits most closely to the data in the image. By finding the bins with the highest value, the most likely circle can be extracted, and the center of this circle can be read from the bin.

Generally, the Hough Transform can be extended to detect ellipses or arbitrary curves as long as they can be expressed by a set of parameters. Since four parameters are needed for denoting an ellipse, the work load for voting is much larger than line or circle detecting. Therefore, the Hough Transform is not satisfactory in this case. In the next section, we will discuss a better alternative based on similar argument. Other methods for detecting interest point can be found in [14], [22], [23], [31], [34].
2.1.1 Iterative Voting

Since the shape of cells has some important properties, such as symmetry, continuity and closure, we apply an iterative voting method using oriented kernels to detect the candidate position for the center of the cell [35]. The basic idea of this algorithm is also a voting method, but instead of transforming into other parameter spaces like the Hough Transform, it defines a series of kernel that vote iteratively along radial direction. The kernel is cone-shaped, as Fig. 2.1 shows. Applying this kernel along the gradient direction, at each iteration and each grid location, the orientation of the kernel is updated. The shape of the kernel is also refined and focused as the iterative process continues. Finally, the point of interest is selected by certain threshold.

![Figure 2.1: Shape of Kernel used in the Method](image)

Let $I(x,y)$ be the original image, $\alpha(x,y)$ be the voting direction where $\alpha(x,y) := (\cos \theta(x,y), \sin \theta(x,y))$, $(r_{\min}, r_{\max})$ be the radial range, $V$ be the voted image, and $A$ be the voting area defined by
\[ A(x, y; r_{\text{min}}, r_{\text{max}}, \Delta) := \{ (x \pm r \cos \theta, y \pm r \sin \phi, |r_{\text{min}} < r < r_{\text{max}}, \theta(x, y) - \Delta \leq \phi \leq \theta(x, y) + \Delta \} \] (2.1.1)

Let \( G(x, y; \sigma, \alpha, A) \) be a 2D Gaussian kernel with variance \( \sigma \), masked by the local voting area \( A(x, y; r_{\text{min}}, r_{\text{max}}, \Delta) \) and oriented in the voting direction \( \alpha(x, y) \). Then the iterative voting algorithm contains following seven steps; for the detailed implementation see [35]:

1. **Initialize the parameters:** \( r_{\text{min}}, r_{\text{max}}, \Delta_{\text{max}} \) and a sequence \( \Delta_{\text{max}} = \Delta_N > \Delta_{N-1} > \ldots > \Delta_0 = 0 \), set \( n := N \), where \( N \) is the number of iterations, \( \Delta_n = \Delta_{\text{max}} \), fix a low gradient threshold \( \Gamma_g \) and a kernel variance \( \sigma \) depending on the expected scale of salient features.

2. **Initialize the saliency feature image:** Define the feature image \( F(x, y) \) to be the local external force at each pixel of the original image. The external force is often set to the gradient magnitude or maximum curvature, depending upon the type of saliency grouping and the presence of local feature boundaries.

3. **Initialize voting direction and magnitude:** Compute \( \nabla(I) \), its magnitude \( \| \nabla(I) \| \), define a pixel subset \( S := \{ (x, y) | \| \nabla(I) \| > \Gamma_g \} \), for each pixel \( (x, y) \in S \), let the voting direction be

\[ \alpha(x, y) := -\frac{(I_x(x, y), I_y(x, y))}{\| \nabla(I) \|} \] (2.1.2)

4. **Compute the votes:** \( V(x, y; r_{\text{min}}, r_{\text{max}}, \Delta) = 0 \). For all pixels \( (x, y) \in S \), update the vote image:

\[
V(x, y; r_{\text{min}}, r_{\text{max}}, \Delta) = V(x, y; r_{\text{min}}, r_{\text{max}}, \Delta) + \sum_{(u,v) \in A} F(x - \frac{w}{2} + u, y - \frac{h}{2} + v)G(u, v; \sigma, \alpha, A),
\] (2.1.3)

where \( w = \max(u) \) and \( h = \max(v) \) are the maximum dimensions of the voting area.
5. **Update the voting direction**: For grid points \((x, y) \in S\), revise the voting direction:

\[
(u^*, v^*) = \arg\max_{(u,v) \in A(x,y;r_{\min},r_{\max},\Delta)} V(u, v; r_{\min}, r_{\max}, \Delta).
\] (2.1.4)

Let \(d_x = u^* - x, d_y = v^* - y\), and \(\alpha(x, y) = \frac{(d_x, d_y)}{\sqrt{d_x^2 + d_y^2}}\).

6. **Refine the angular range**: Let \(n := n - 1\), repeat steps 4 – 6 until \(n = 0\).

7. **Localize centers of mass**: Localize centers of mass by threshold

\[
\{(x, y)|V(x, y; r_{\min}, r_{\max}, \Delta) > \Gamma_v\},
\] (2.1.5)

This gives a binary image with interested points. These points show the locations of the cell centers.

### 2.1.2 Result for Seed Detection

The algorithm is tested with different images. By adjusting the parameters \(r_{\min}, r_{\max}, nsectors\) and \(thresh\), the seed for each cell is detected accurately. The adjustment of parameters is discussed in the following subsection. Usually, four or five iterations are enough to determine the location of seeds, so this algorithm is more efficient than the Hough Transform and the performance of seed detecting is satisfactory. The following figures show some examples of seed detection.

In Fig. 2.2, the sub-figure (a) is the original image, in which the cells are scattered and parts of the cells are close to each other. The following four sub-figures show the procedure of applying iterative voting method. After the iterative voting procedure, a threshold is applied in order to get the binary image named "loct" (sub-figure (e)), these white spots in the sub-figure "loct" are the location of the seeds. The seeds are marked with different numbers and the result is shown with different gray levels, yielding the sub-figure (f) named "label". The last one shows the seed locations at the background of original image in order to check the relative location. From that
we can see the seed locations are approximately in the middle of each cell, as expected.

In Fig. 2.3, the shape of the cells is not exactly elliptic and all the cells are clustered together. Different from Fig. 2.2, this voting procedure needs five steps. But after all, all seeds are detected and located in the middle of each cell, which can be seen from the last sub-figure.

In Fig. 2.4, the test image is more difficult since some of the cells even overlap each other and the sizes of the cells are different. This algorithm still can detect all the cells and show the location of each cell. Therefore, the performance of the algorithm is stable and satisfactory.
Figure 2.2: Seed Detection Result for Test Image I (crop_138_130)

Figure 2.3: Seed Detection Result for Test Image II (crop_140_120)
2.1.3 Parameter Setting and Operation Time

There are four important parameters in seed detection part, they are $r_{\text{max}}$, $r_{\text{min}}$, $n\text{sectors}$ and $\text{thresh}$. The parameters $r_{\text{max}}$ and $r_{\text{min}}$ determine the voting area, $n\text{sector}$ determines the number of iteration; $\text{thresh}$ determines the value of threshold applied after the voting procedure. Those parameters are set up empirically, but there are still some rules behind it.

- The $r_{\text{max}}$ usually equals to the radius of the cell. If the radii of the cells are different, then choosing the maximum radius is acceptable.

- The value of the $r_{\text{min}}$ is used to compensate the $r_{\text{max}}$, usually it equals to 4 or 5. Together with $r_{\text{min}}$, the voting area is determined.

- The number of iteration $n_{\text{iter}} := n\text{sectors}/8 - 1$, so the value of $n\text{sector}$ should be multiples of 8. For most cases, 4 iterations are enough to determine the location of seeds. If the shape of cell is not regular, then more iteration steps are needed.

- The threshold is very important in the program since it is a judgemental standard. The value of the threshold can vary from 1.0 to around 8.0, it depends on the property of images. The key idea to determine the value is to set the number of false positives as small as possible.
Adjusting parameter needs some experience, but for certain type of cells, the parameters are fixed. Generally speaking, this program is stable and easy to implement. The result is especially good for clustered cells, because the cells have radial symmetric properties but the background doesn’t have this property. Table 2.1 shows some example of parameters setting, from which we can see some of the rules discussed above.

<table>
<thead>
<tr>
<th>Image Name</th>
<th>$r_{min}$</th>
<th>$r_{max}$</th>
<th>nsectors</th>
<th>thresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>crop_138_130</td>
<td>5</td>
<td>19</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>crop_140_120</td>
<td>5</td>
<td>30</td>
<td>48</td>
<td>4</td>
</tr>
<tr>
<td>crop_150_150</td>
<td>5</td>
<td>30</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>crop_176_139</td>
<td>4</td>
<td>25</td>
<td>40</td>
<td>2.5</td>
</tr>
<tr>
<td>crop_200_200</td>
<td>4</td>
<td>25</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>crop_256_256</td>
<td>4</td>
<td>25</td>
<td>40</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Table 2.1: Parameter Setting for Seeds Detection Part

This method is relatively fast since the iteration number is not large. As we expect, the operation time depends on the image size. From the Table 2.2, we can see that the larger the image size is, the more time is needed for the procedure. Iteration number is also an important factor influencing the operation time. crop_140_120 and crop_138_130 have similar size, but the operation time for crop_140_120 is much larger than crop_138_130, because the crop_140_120 needs five iterations while crop_138_130 only needs four iterations. Besides, the parameters $r_{min}$ and $r_{max}$ have influence to operation time as well, because these two parameters determine the voting area. The larger the voting area is, the more operation time is needed. We can see the effect from the Table 2.1 and the Table 2.2. The image size of ”crop_176_139” is larger than ”crop_150_150”, and the iteration numbers for both images are the same, but the operation time of ”crop_176_139” is less than ”crop_150_150”. The reason is that the voting area of ”crop_176_139” is smaller than ”crop_150_150”.


<table>
<thead>
<tr>
<th>Image Name</th>
<th>Size of Image</th>
<th># iter</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>crop_138_130</td>
<td>138 × 130</td>
<td>4</td>
<td>7.812452</td>
</tr>
<tr>
<td>crop_140_120</td>
<td>140 × 120</td>
<td>5</td>
<td>33.826043</td>
</tr>
<tr>
<td>crop_150_150</td>
<td>150 × 150</td>
<td>4</td>
<td>29.108005</td>
</tr>
<tr>
<td>crop_176_139</td>
<td>176 × 139</td>
<td>4</td>
<td>23.037405</td>
</tr>
<tr>
<td>crop_200_200</td>
<td>200 × 200</td>
<td>4</td>
<td>37.543013</td>
</tr>
<tr>
<td>crop_256_256</td>
<td>256 × 256</td>
<td>4</td>
<td>58.740347</td>
</tr>
</tbody>
</table>

Table 2.2: Operation Time for Seeds Detection Part

2.2 Level Set Method for Clustered Cell Image

After detecting the center of each cell, we already know the number of cells. The next step is to detect each candidate cell. This part contains three steps, each step is referred to a level set equation. The initial condition for the level set equation is inside the cell, so the first equation, which describes the interface evolution, should have positive speed $F$. In other words, the front will grow outwards. The model contains some stopping criterion which will stop the evolution when it approaches the edge of detected cells. After this procedure, the interface will stop near the internal boundary of cells. The second step is to set up another level set equation removing the boundary detect term to let the interface keep growing. The only constraint for the second step is that the growing fronts can’t penetrate each other. The second step is only allowed for finite steps to ensure the front exceed the cell boundary. The last step is to move the interface inwards again until it finds the external cell boundary.

2.2.1 Initial Expansion Level Set Function

Motivated by the traditional level set model discussed in the previous chapter, we first choose the level set equation defined as below [28]:

$$
\phi_t + g \cdot (1 - \epsilon \kappa) \cdot |\nabla \phi| - \beta \nabla g \cdot \nabla \phi = 0,
$$

(2.2.1)

where $g = e^{-\alpha |\nabla (G \ast I_0(x))|}$, $\alpha > 0$, $G \ast I_0(x)$ is the convolution of the original image $I_0(x)$ with a Gaussian function $G$. The speed function $F = g \cdot (1 - \epsilon \kappa)$ contains an
inflationary term (+1), which determines the direction of evolution to be outward. The curvature term ($\epsilon \kappa$) regularizes the surface by accelerating the movement of those parts of surface behind the average of the front and slowing down the advanced parts of flow. The parameter $\epsilon$ determines the strength of regularization, if $\epsilon$ is large, it will smooth out front irregularities. For extreme cases, the final front will become a circle. If $\epsilon$ is small, the front will maintain sharp corners. In practice, an intermediate value of $\epsilon$ is chosen that allows the front to have concavities (concavities are possible for cell border), while small gaps and noise are smoothed. The effect of $g$ is to speed up the flow in those areas where the image gradient is low and slowing down where the image gradient is high. Because of this term, the front slows down almost to stop when it reaches the internal cell boundary. The parameter $\alpha$ determines the sensitivity of the flow to the gradient. The extra term $\beta \nabla g \cdot \nabla \phi$ is a parabolic term that enhances the edge effect.

In this step, there is another restriction condition for the equation. The growing front can’t not invade other cells’ region when seed grows. By using this level set equation, the internal boundary of the cells is detected.

### 2.2.2 Free Expansion Level Set Function

The initial expansion level set function usually causes underestimation of cell area. So a second and third step are added to compensate the result. The second step is the free expansion, in which the front is allowed to expand freely and the speed of evolution doesn’t rely on the gradient of original image. The level set equation is simply defined as below:

$$\phi_t + |\nabla \phi| = 0. \quad (2.2.2)$$

Similar as the first step, the growing fronts can not penetrate each other when the front expands. This expansion only needs a number of steps to ensure that all the fronts move beyond the external boundary of cells. The number of iteration depends on the thickness of the cell boundary.
2.2.3 Surface Wrapping Level Set Function

After the free expansion step, the fronts are located outside the cells’ boundary. The last step is to move the front inwards to get the exact location of the external cell boundary. So the speed function \( F = g \cdot (-1 - \epsilon \kappa) \) contains an shrinking term \((-1)\), which determines the direction of evolution to be inward. Similar to the initial expansion flow, \( g = e^{-\alpha|\nabla(G * I_0(x))|} \), \( \alpha > 0 \), \( G * I_0(x) \) is the convolution of the original image \( I_0(x) \) with a Gaussian function \( G \). The effect of this term is to detect the external boundary of the cells. The parameter \( \alpha \) can be different value from the initial expansion. In short, the level set equation is defined as:

\[
\phi_t + g \cdot (-1 - \epsilon \kappa) \vert \nabla \phi \vert = 0. \tag{2.2.3}
\]

2.2.4 Numerical Implementation

First, we define some notations which will be used in the numerical implementation. \( D^{+x} \) is the forward difference approximation for the spatial derivative \( u_x \). Similarly, \( D^{-x} \) is the backward difference approximation and \( D^{0x} \) is the central difference approximation, which are defined respectively as follows:

\[
D^{+x}u \equiv \frac{u(x + h, t) - u(x, t)}{h}, \quad (2.2.4a)
\]
\[
D^{-x}u \equiv \frac{u(x, t) - u(x - h, t)}{h}, \quad (2.2.4b)
\]
\[
D^{0x}u \equiv \frac{u(x + h, t) - u(x - h, t)}{2h}. \quad (2.2.4c)
\]

The operators \( \nabla^+ \) and \( \nabla^- \) are calculated as follows:

\[
\nabla^+ = \left[ \max(D^{-x}_{ij}, 0)^2 + \min(D^{+x}_{ij}, 0)^2 + \max(D^{-y}_{ij}, 0)^2 + \min(D^{+y}_{ij}, 0)^2 \right]^{1/2}, \quad (2.2.5)
\]
\[
\nabla^- = \left[ \max(D^{+x}_{ij}, 0)^2 + \min(D^{-x}_{ij}, 0)^2 + \max(D^{+y}_{ij}, 0)^2 + \min(D^{-y}_{ij}, 0)^2 \right]^{1/2}, \quad (2.2.6)
\]

In general, the numerical implementation for the level set equations defined above is followed the algorithm introduced in [27]. In brief, it can be denoted as the following equation:
\[ \phi_{ij}^{n+1} = \phi_{ij}^n + \Delta t \left[ -\left[ \max(F_{0ij}, 0) \nabla^+ + \min(F_{0ij}, 0) \nabla^- \right] + \left\{ \max(u_{ij}^n, 0) D_{ij}^{-x} + \min(u_{ij}^n) D_{ij}^{+x} \right\} \right] \\
\qquad \left[ + \max(v_{ij}^n, 0) D_{ij}^{-y} + \min(v_{ij}^n) D_{ij}^{+y} \right] \\
\qquad + [\epsilon K_{ij}^n((D_{ij}^{0x})^2 + (D_{ij}^{0y})^2)^{1/2}] \right]. \]  

(2.2.7)

For the initial expansion level set function Eq. (2.2.1), \( F_0 \) refers to \( g \), \( (u, v) \) refers to \(-\beta \nabla g\), and \( K \) refers to \( g \cdot \kappa \) in Eq. (2.2.1).

For the free expansion level set function Eq. (2.2.2), \( F_0 \equiv 1 \) and the other terms are eliminated. So the Eq. (2.2.7) is reduced to:

\[ \phi_{ij}^{n+1} = \phi_{ij}^n - \Delta t \cdot \nabla^+, \]  

(2.2.8)

For the surface wrapping level set function Eq. (2.2.3), \( F_0 \) refers to \(-g\), which is negative, \( K \) refers to \( g \cdot \kappa \) in Eq. (2.2.3) and \( (u, v) \equiv 0 \). So the equation can be rewritten as below:

\[ \phi_{ij}^{n+1} = \phi_{ij}^n + \Delta t \left[ -g \cdot \nabla^- + (\epsilon K_{ij}^n((D_{ij}^{0x})^2 + (D_{ij}^{0y})^2)^{1/2}) \right]. \]  

(2.2.9)

Another restriction condition for the equation is that the front can not invade other cells’ region when the seed grows. But in the iteration procedure, every front is moving independently from other fronts. To avoid the penetration phenomenon, in every iteration step, the outcome is considered as a trial function. By comparing with other fronts in previous steps using the following standard: \( \phi_{m+1}^i = \max\{\phi_{m+1}^{i(trial)}, -\phi_{m}^j\}, \) \( i \leq j \leq n, \) \( i \neq j \), the final movement of the front is determined.

For the three level set equations, a reinitialization phase is necessary. The purpose of reinitialization is to keep the evolving level set function close to a signed distance function during the evolution. It is a numerical remedy for maintaining stable curve
evolution. The reinitialization step is to solve the following evolution equation:

\[
\begin{align*}
\psi_r &= \text{sign}(\phi(t))(1 - |\nabla \psi|), \\
\psi(0, \cdot) &= \phi(t, \cdot).
\end{align*}
\] (2.2.10)

Here, \( \phi(t, \cdot) \) is the solution \( \phi \) at time \( t \). This equation is solved by an iterative method. In this program 5 iterations are used. The result \( \psi \) will be the new \( \phi \) used in the program.

### 2.2.5 Parameter Setting and Operation Time

The parameters in the Eq. (2.2.1) should be chosen carefully, since they will influence the accuracy of final result. The values of the parameters used in the test are chosen empirically. By testing with different images, it was found that the given set of parameters in Table 2.3 can be used for images within a broad range of image characteristics. For the first level set equation, \( \alpha = 0.015, \beta = 0.2, \epsilon = 0.005 \). For the third level set equation, \( \alpha = 0.05, \epsilon = 0.005 \). Since \( \alpha \) determines the sensitivity of the flow to the gradient, for the initial expansion, the value of \( \alpha \) should be small in order to avoid the influence of sub-structures inside the cells. However, for the interface wrapping part, the value of \( \alpha \) should be large in order to get accurate position of external boundary of cells. The time step \( \Delta t \) is also an important parameter, it determines the speed of movement. With too large time step \( \Delta t \), the front can not converge to correct solution, with smaller time step \( \Delta t \), the evolution speed is very slow, as it needs to take more steps to get the right solution. For the first two level set equations, the time step \( \Delta t \) is chosen as 0.1. For the last step, the time step \( \Delta t \) is chosen smaller value to increase the accuracy.

There are several stopping criteria for this method. For instance, it can be checking the volume increases after each iteration. Setting a minimum threshold of volume change can interrupt the flow. Alternatively, a conservatively high number of iterations can be set. In this work, the second method is chosen. The parameter setting and iteration numbers can be found in Table 2.3. The iteration number for the last two steps could be a little different for different images. Usually, the thicker the
membrane of cells is, the larger the iteration number is. The iteration number of the
last level set equation is generally twice the iteration number of the second level set
equation.

<table>
<thead>
<tr>
<th>Flow Name</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\epsilon$</th>
<th>$\Delta t$</th>
<th>$# iter$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Expansion</td>
<td>0.015</td>
<td>0.2</td>
<td>0.005</td>
<td>0.10</td>
<td>200</td>
</tr>
<tr>
<td>Free Expansion</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.10</td>
<td>20~40</td>
</tr>
<tr>
<td>Surface Wrapping</td>
<td>0.05</td>
<td>-</td>
<td>0.005</td>
<td>0.05</td>
<td>40~100</td>
</tr>
</tbody>
</table>

Table 2.3: Parameter Setting for Level Set Method

Table 2.4 shows the operation time for several test images. For comparison sake, all
the test images listed in this table use 200 iterations for initial expansion, 20 iterations
for free expansion and 40 iterations for interface wrapping. The operation time de-
PENDs on the image size. When the image size grows, more memory space is required
and more operation time is needed. The number of cells is another important factor
determining the operation time. For each cell, the program needs to solve one PDE.
When the number of the cells increases, the operation time grows sharply. For an
image of 256 $\times$ 256 pixels and containing 68 cells, the program will run around two
hours. Apparently, this is the drawback of this method, because the model needs to
set up $n$ PDEs for $n$ cells, and solving each PDE needs quite a lot of space and time
resource. When dealing with images with too many cells, the work load is so heavy
that takes too much time to get the result.

2.2.6 Numerical Result for Level Set Method

The program is tested with different types of cell images. The following figures show
how this method works on these real cell images. By setting appropriate parameters,
the level set method can segment cell clusters into individual cells. The first test
image contains scattered cells. In Fig. 2.5, sub-figure (a) shows the initial condition
for level set function. Sub-figure (b) is the result of initial expansion, in which the
The second test image contains clustered cells, similar as the first test image. Fig. 2.6 shows the procedure of fronts evolution. At the end, all the cells are detected and marked with colors. This method distinguishes the location of cells and preserves the shape of cells.

The third test image is more difficult than the other two because of the overlapping phenomenon in the image. However, this method can still detect the location of the cells and find out the area of the cells, the result is shown in Fig. 2.7.

Using this method, the number of cells is known and the area of each cell can be calculated. Fig. 2.8 is the histogram for the test images above. Sub-figure (a) is the histogram for test image ”crop_138_130”, there are 11 cells in the image and they have similar size. There are two cells smaller than others, because they are only half cells. Sub-figure (b) is the histogram for test image ”crop_140_120”, in which contains 17 cells. From this histogram, we can see there are four half cells. Sub-figure (c) is the histogram for test image ”crop_176_139”, 24 cells are detected, and the ratio of the

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Size of Image</th>
<th>Cell</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>crop_138_130</td>
<td>138 × 130</td>
<td>11</td>
<td>150.427114</td>
</tr>
<tr>
<td>crop_140_120</td>
<td>140 × 120</td>
<td>17</td>
<td>235.591945</td>
</tr>
<tr>
<td>crop_150_150</td>
<td>150 × 150</td>
<td>19</td>
<td>358.292041</td>
</tr>
<tr>
<td>crop_176_139</td>
<td>176 × 139</td>
<td>24</td>
<td>455.216153</td>
</tr>
<tr>
<td>crop_200_200</td>
<td>200 × 200</td>
<td>31</td>
<td>1251.933908</td>
</tr>
<tr>
<td>crop_256_256</td>
<td>256 × 256</td>
<td>68</td>
<td>6319.549251</td>
</tr>
</tbody>
</table>

Table 2.4: Operation Time for Level Set Method
detected cells is shown in the figure. However, because of the overlapping problem, the area is not the real area of cells, but the area of the segmented image.

Generally speaking, this method is useful for segmenting clustered cells. By displaying different colors for different cell regions, people can identify cells easily. However, this method still has some shortage, the level set model still relies on the edge-function depending on the image gradient $|\nabla(G * I_0(x))|$ to stop the curve evolution, so this model can detect only objects with edges defined by gradient. In practice, the discrete gradients are bounded and then the stopping function is never zero on the edges, and the curve may pass through the boundary. This phenomenon also appears in the test images displayed above. This shortage will introduce extra error in calculating cell area. If the image is very noisy, then the isotropic smoothing Gaussian has to be strong, which will smooth the edges too. In the next chapter, we will discuss and test a method that overcomes the use of an edge detection mechanism.

Figure 2.5: Result for Test Image I (crop_138_130): (a) Initial Contour; (b) Contour Result for Initial Expansion (200 iterations); (c) Contour Result for Free Expansion (20 iterations); (d) Contour Result for Surface Wrapping (40 iterations); (e) Phase Result for Surface Wrapping
Figure 2.6: Result for Test Image II (crop.140.120): (a) Initial Contour; (b) Contour Result for Initial Expansion (200 iterations); (c) Contour Result for Free Expansion (10 iterations); (d) Contour Result for Surface Wrapping (20 iterations); (e) Phase Result for Surface Wrapping

Figure 2.7: Result for Test Image III (crop.176.139): (a) Initial Contour; (b) Contour Result for Initial Expansion (200 iterations); (c) Contour Result for Free Expansion (20 iterations); (d) Contour Result for Surface Wrapping (40 iterations); (e) Phase Result for Surface Wrapping
Figure 2.8: Histogram for Test Images
Chapter 3

Multiphase Level Set Method and its Application in Cell Segmentation

In the previous chapter, a method for segmenting clustered cell images was proposed. Since the limitation of that model, it can not be applied for cell images with too many cells and cell images with ambiguous boundaries. In this chapter, we will use the active contour model without edges from Chan and Vese (1999, 2001) [6]. Based on this model and the Four-Color theorem, a multiphase level set method was developed. It has been applied in image segmentation in recent years and become a popular research topic due to its excellent performance. We will start with the theory of this model [2], [6], [30] and briefly introduce its numerical implementation. Then we will discuss the application of this model in real cell images.

3.1 Active Contour Model Without Edges

In the article of Chan and Vese [6], the authors propose a new model for image segmentation based on Mumford-Shah functional and level sets. This model doesn’t depend on the gradient of the image as stopping term. Therefore it can be used in
images with ambiguous boundaries.

3.1.1 Formulation of Level Set Equation

The model can be defined as the following minimization problem [6]:

Given the curve $C = \partial \omega$, with $\omega \in \Omega$ an open subset in $\mathbb{R}^2$, and two unknown constants $c_1$ and $c_2$, denoting $\Omega_1 = \omega$, $\Omega_2 = \Omega / \omega$, $u_0$ is the original image, minimize the following energy with respect to $c_1, c_2$ and $C$:

$$F(c_1, c_2, C) = \mu \cdot \text{Length}(C) + \nu \cdot \text{Area}(\text{inside}(C))$$
$$+ \lambda_1 \int_{\text{inside}(C)} (u_0(x, y) - c_1)^2 dxdy$$
$$+ \lambda_2 \int_{\text{outside}(C)} (u_0(x, y) - c_2)^2 dxdy. \tag{3.1.1}$$

The minimization functional can be reformulated in terms of the level set function $\phi(x)$ with:

$$\begin{align*}
C &= \partial \omega = \{(x, y) \in \Omega : \phi(x, y) = 0\}, \\
\text{inside}(C) &= \omega = \{(x, y) \in \Omega : \phi(x, y) > 0\}, \\
\text{outside}(C) &= \Omega \setminus \overline{\omega} = \{(x, y) \in \Omega : \phi(x, y) < 0\}. \tag{3.1.2}
\end{align*}$$

Then the functional becomes:

$$F(c_1, c_2, \phi) = \mu \int_\Omega \delta(\phi(x, y)) |\nabla \phi(x, y)| dxdy$$
$$+ \nu \int_\Omega H(\phi(x, y)) dxdy$$
$$+ \lambda_1 \int_\Omega (u_0(x, y) - c_1)^2 H(\phi) dxdy$$
$$+ \lambda_2 \int_\Omega (u_0(x, y) - c_2)^2 (1 - H(\phi)) dxdy, \tag{3.1.3}$$

where $H(z)$ is Heaviside function and $\delta$ is the Dirac measure.

To minimize this functional with respect to $\phi$, parameterize the descent direction by an artificial time $t \geq 0$, the equation in $\phi$ is:

$$\frac{\partial \phi}{\partial t} = \delta(\phi) \left[ \mu \cdot \text{div} \left( \nabla \phi \right) - \nu - \lambda_1 (u_0 - c_1)^2 + \lambda_2 (u_0 - c_2)^2 \right] = 0 \quad \text{in } (0, \infty) \times \Omega,$$
$$\phi(0, x, y) = \phi_0(x, y) \quad \text{in } \Omega,$$
$$\frac{\delta \phi}{|\nabla \phi|} = 0 \quad \text{on } \partial \Omega. \tag{3.1.4}$$
and
\[ c_1 = \frac{\int_\Omega u_0(x,y)H(\phi(t,x,y))dxdy}{\int_\Omega H(\phi(t,x,y))dxdy}, \quad c_2 = \frac{\int_\Omega u_0(x,y)(1 - H(\phi(t,x,y)))dxdy}{\int_\Omega (1 - H(\phi(t,x,y)))dxdy}. \] (3.1.5)

3.1.2 Numerical Implementation and Result

The PDE (3.1.4) can be solved numerically using the iterative algorithm from [6]. First, for given \( \phi^n \), calculate the constant \( c_1(\phi^n) \) and \( c_2(\phi^n) \) according to Eq. (3.1.5) respectively. Here, we use the approximation function \( H_{2,\epsilon} \) and \( \delta_{2,\epsilon} \) instead of the Heaviside Function \( H \) and Dirac Function \( \delta \). The approximation function \( H_{2,\epsilon} \) and \( \delta_{2,\epsilon} \) are defined below [6]:

\[ H_{2,\epsilon}(\phi) = \frac{1}{2}(1 + \frac{2}{\pi} \arctan(\frac{\phi}{\epsilon})), \] (3.1.6)

\[ \delta_{2,\epsilon}(\phi) = H_{2,\epsilon} = \frac{1}{\pi \epsilon^2 + \phi^2}. \] (3.1.7)

Fig. 3.1 compares two different approximation functions for Heaviside Function and Dirac Function. Here \( H2 \) and \( D2 \) refer to \( H_{2,\epsilon} \) and \( \delta_{2,\epsilon} \) defined in Eq. (3.1.6) and Eq. (3.1.7) respectively. \( H1 \) and \( D1 \) refer to \( H_{1,\epsilon} \) and \( \delta_{1,\epsilon} \) defined as [18]:

\[ H_{1,\epsilon}(\phi) = \begin{cases} 0 & \phi < \epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{\pi} \sin(\frac{\pi \phi}{\epsilon}) & -\epsilon \leq \phi \leq \epsilon \\ 1 & \epsilon < \phi \end{cases} \] (3.1.8)

\[ \delta_{1,\epsilon}(\phi) = \begin{cases} 0 & \phi < \epsilon \\ \frac{1}{2\epsilon} + \frac{1}{2\epsilon} \cos(\frac{\pi \phi}{\epsilon}) & -\epsilon \leq \phi \leq \epsilon \\ 0 & \epsilon < \phi \end{cases} \] (3.1.9)

As \( \epsilon \to 0 \), both approximations converge to \( H \) and \( \delta \). A difference is that \( \delta_{1,\epsilon} \) has a small support, the interval \([-\epsilon, \epsilon]\), while \( \delta_{2,\epsilon} \) is different of zero everywhere. The reason for choosing these two approximation functions defined in Eq. (3.1.6) and Eq. (3.1.7) is to get the global minimizer. The energy defined in Eq. (3.1.1) is non-convex, therefore, it allows many local minima. Choosing the approximation functions \( H_{1,\epsilon} \) and \( \delta_{1,\epsilon} \), the algorithm sometimes computes a local minimizer of the energy. Using
the approximation functions $H_{2,\epsilon}$ and $\delta_{2,\epsilon}$, the algorithm tends to compute a global minimizer. One of the reasons is that the Euler–Lagrange equation for $\phi$ acts only locally on a few level curves around $\{\phi = 0\}$ using $H_{1,\epsilon}$ and $\delta_{1,\epsilon}$. But using $H_{2,\epsilon}$ and $\delta_{2,\epsilon}$, the equation acts on all level curves. In this way, in practice, we can obtain a global minimizer, independently of the position of the initial curve [6].

The second step for the algorithm is to update $\phi$ using the following equation, here $\nu = 0$, $\lambda_1 = \lambda_2 = 1$:

$$
\phi_{i,j}^{n+1} = \frac{1}{\sigma} \left[ \phi_{i,j}^n + m_1 (C_1 \phi_{i+1,j}^n + C_2 \phi_{i-1,j}^n + C_3 \phi_{i,j+1}^n + C_4 \phi_{i,j-1}^n) \\
+ \Delta t \delta_c (\phi_{i,j}^n) (- (u_{0,i,j} - c_1^2)^2 + (u_{0,i,j} - c_2^2)^2) \right],
$$

(3.1.10)

where $m_1 = \frac{\Delta t}{\tau} \delta_c (\phi_{i,j}) \mu$, $C = 1 + m_1 (C_1 + C_2 + C_3 + C_4)$, $u_0$ is the original image. The discretization is done on a uniform square mesh, hence $h = 1$. Then $\epsilon$ is chosen
as $\epsilon = h = 1$. The coefficients $C_1, C_2, C_3$ and $C_4$ are defined as following:

\begin{align*}
C_1 &= \frac{1}{\sqrt{\left(\frac{\phi_{n+1,j+1} - \phi_{n,j+1}}{h}\right)^2 + \left(\frac{\phi_{n,j+1} - \phi_{n,1,j}}{2h}\right)^2}} \\
C_2 &= \frac{1}{\sqrt{\left(\frac{\phi_{n,j} - \phi_{n-1,j}}{h}\right)^2 + \left(\frac{\phi_{n,j} - \phi_{n,j-1}}{2h}\right)^2}} \\
C_3 &= \frac{1}{\sqrt{\left(\frac{\phi_{n+1,j} - \phi_{n,j-1}}{2h}\right)^2 + \left(\frac{\phi_{n,j-1} - \phi_{n,j-2}}{h}\right)^2}} \\
C_4 &= \frac{1}{\sqrt{\left(\frac{\phi_{n+1,j} - \phi_{n,1,j-1}}{2h}\right)^2 + \left(\frac{\phi_{n,j-1} - \phi_{n,j-2}}{h}\right)^2}}
\end{align*}

(3.1.11) (3.1.12) (3.1.13) (3.1.14)

A computer generated object is used to test the program. Fig. 3.2 shows the evolution procedure. This program is much faster than the method introduced in Chapter 2. For the time step $\Delta t = 0.1$, only 70 iterations are needed for convergence. The size of this artificial image is $110 \times 129$, but the operation time is only $5.361076$ seconds.

Figure 3.2: Result for Chan-Vese Model: $Size = 110 \times 129$, initial condition: $\phi_0 = -\sqrt{(x - 55)^2 + (y - 60)^2} + 25; \mu = 0.0005 \cdot 255^2$
3.2 Multiphase Level Set Method

The traditional Active Contour Without Edge model can segment image into two regions - background and foreground. In order to segment images into more regions, a Multiphase Level Set Method has been developed. Actually, the level set model set up in Chapter 2 is also a multiphase level set model since there are \( n \) level set equations refer to \( n \) cells. Based on the Four-Color Theorem, only four colors are enough to dye all the regions in a partition. Therefore only two level set functions will suffice to represent any partition, as Fig. 3.3 shows.

![Figure 3.3: 2 Curves Partition the Domain into 4 Regions](image)

3.2.1 Formulation of Multiphase Level Set Method

Similar to the traditional Chan-Vese model, the multiphase level set method can be treated as a minimization problem. The energy functional which will be minimized is defined as follows:

\[
F(c, \phi) = \int_{\Omega} (u_0 - c_{11})^2 H(\phi_1)H(\phi_2) dxdy \\
+ \int_{\Omega} (u_0 - c_{10})^2 H(\phi_1)(1 - H(\phi_2)) dxdy \\
+ \int_{\Omega} (u_0 - c_{01})^2 (1 - H(\phi_1))H(\phi_2) dxdy \\
+ \int_{\Omega} (u_0 - c_{00})^2 (1 - H(\phi_1))(1 - H(\phi_2)) dxdy \\
+ \mu \int_{\Omega} |\nabla H(\phi_1)| + \mu \int_{\Omega} |\nabla H(\phi_2)|,
\]

(3.2.1)
where

\[ c_{11}(\phi) = \text{mean}(u_0) \text{ in } \{(x, y) : \phi_1(t, x, y) > 0, \phi_2(t, x, y) > 0\}, \]  
(3.2.2)

\[ c_{10}(\phi) = \text{mean}(u_0) \text{ in } \{(x, y) : \phi_1(t, x, y) > 0, \phi_2(t, x, y) < 0\}, \]  
(3.2.3)

\[ c_{01}(\phi) = \text{mean}(u_0) \text{ in } \{(x, y) : \phi_1(t, x, y) < 0, \phi_2(t, x, y) > 0\}, \]  
(3.2.4)

\[ c_{00}(\phi) = \text{mean}(u_0) \text{ in } \{(x, y) : \phi_1(t, x, y) < 0, \phi_2(t, x, y) < 0\}. \]  
(3.2.5)

The Euler-Lagrange equations obtained by minimizing Eq. (3.2.1) with respect to \(c\) and \(\phi\) are:

\[
\frac{\partial \phi_1}{\partial t} = \delta \epsilon(\phi_1) \left\{ \mu \cdot \text{div}(\nabla \phi_1) - \left[ \left( (u_0 - c_{11})^2 - (u_0 - c_{01})^2 \right) H(\phi_2) 
+ \left( (u_0 - c_{10})^2 - (u_0 - c_{00})^2 \right)(1 - H(\phi_2)) \right] \right\},
\]  
(3.2.6)

\[
\frac{\partial \phi_2}{\partial t} = \delta \epsilon(\phi_2) \left\{ \mu \cdot \text{div}(\nabla \phi_2) - \left[ \left( (u_0 - c_{11})^2 - (u_0 - c_{01})^2 \right) H(\phi_1) 
+ \left( (u_0 - c_{10})^2 - (u_0 - c_{00})^2 \right)(1 - H(\phi_1)) \right] \right\}.
\]  
(3.2.7)

### 3.2.2 Numerical Implementation and Application in Cell Segmentation

The numerical implementation for this model is similar to the traditional Chan-Vese Model. First thing to do is to calculate the four constant \(c_{11}, c_{10}, c_{01},\) and \(c_{00}\) using the Eqs. (3.2.2) - (3.2.5). Then the values of \(\phi_1, \phi_2\) are updated using the following equations:

\[
\phi_{1,i,j}^{n+1} = \frac{1}{C}[\phi_{1,i,j}^n + m_1(C_1\phi_{1,i+1,j}^n + C_2\phi_{1,i-1,j}^n + C_3\phi_{1,i,j+1}^n + C_4\phi_{1,i,j-1}^n) + \Delta t \delta \epsilon(\phi_{1,i,j}^n)(-\left( (u_{0,i,j} - c_{11})^2 H_c(\phi_{2,i,j}^n) - (u_{0,i,j} - c_{10})^2(1 - H_c(\phi_{2,i,j}^n)) \right) 
+ \left( (u_{0,i,j} - c_{01})^2 H_c(\phi_{2,i,j}^n) + (u_{0,i,j} - c_{00})^2(1 - H_c(\phi_{2,i,j}^n)) \right)],
\]  
(3.2.8)
\[ \phi^{n+1}_{2,i,j} = \frac{1}{D} \left[ \phi^n_{2,i,j} + m_2 (D_1 \phi^n_{2,i+1,j} + D_2 \phi^n_{2,i-1,j} + D_3 \phi^n_{2,j+1,i} + D_4 \phi^n_{2,j-1,i}) \right. \\
+ \Delta t \delta_t(\phi^n_{2,i,j}) (- (u_{0,i,j} - c_{11}^n)^2 H_t(\phi^n_{1,i,j}) + (u_{0,i,j} - c_{10}^n)^2 H_t(\phi^n_{1,i,j})) \\
\left. - (u_{0,i,j} - c_{01}^n)^2 (1 - H_t(\phi^n_{1,i,j})) + (u_{0,i,j} - c_{00}^n)^2 (1 - H_t(\phi^n_{1,i,j})) \right] \] (3.2.9)

Where \( m_1 = \frac{\Delta t}{\lambda} \delta_t(\phi_{1,i,j}) \mu \), \( m_2 = \frac{\Delta t}{\lambda} \delta_t(\phi_{2,i,j}) \mu \), \( C = 1 + m_1 (C_1 + C_2 + C_3 + C_4) \),
\( D = 1 + m_1 (D_1 + D_2 + D_3 + D_4) \), and \( u_0 \) is the original image. The coefficients \( C_1, C_2, C_3, C_4 \), and \( D_1, D_2, D_3, D_4 \), are defined as follows:

\[ C_1 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i,j}}{h}\right)^2 + \left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i-1,j}}{2h}\right)^2}} \] (3.2.10)

\[ C_2 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i,j} - \phi^n_{2,i-1,j}}{h}\right)^2 + \left(\frac{\phi^n_{2,i-1,j} - \phi^n_{2,i-3,j}}{2h}\right)^2}} \] (3.2.11)

\[ C_3 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i,j} - \phi^n_{2,i+1,j}}{2h}\right)^2 + \left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i+3,j}}{h}\right)^2}} \] (3.2.12)

\[ C_4 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i-1,j} - \phi^n_{2,i,j}}{h}\right)^2 + \left(\frac{\phi^n_{2,i,j} - \phi^n_{2,i-3,j}}{2h}\right)^2}} \] (3.2.13)

\[ D_1 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i,j}}{h}\right)^2 + \left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i-1,j}}{2h}\right)^2}} \] (3.2.14)

\[ D_2 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i,j} - \phi^n_{2,i-1,j}}{h}\right)^2 + \left(\frac{\phi^n_{2,i-1,j} - \phi^n_{2,i-3,j}}{2h}\right)^2}} \] (3.2.15)

\[ D_3 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i,j}}{2h}\right)^2 + \left(\frac{\phi^n_{2,i+1,j} - \phi^n_{2,i+3,j}}{h}\right)^2}} \] (3.2.16)

\[ D_4 = \frac{1}{\sqrt{\left(\frac{\phi^n_{2,i-1,j} - \phi^n_{2,i,j}}{2h}\right)^2 + \left(\frac{\phi^n_{2,i-1,j} - \phi^n_{2,i-3,j}}{h}\right)^2}} \] (3.2.17)

This model has some applications in cell segmentation. It is especially suitable for classifying different types of cells. Fig. 3.4 shows the segmentation result\(^1\). There are two types of cells in the image. Using this model, these two types of cells are distinguished and displayed with different colors. Here, the reason of using many

\(^1\)The test image is from [10]
small circles as initial condition is to get a global minimum. Since the energy to be minimized is not convex, there is no uniqueness for the minimizers. For this type of initial condition, the algorithm tends to converge to a global minimizer.

The only varying parameter in the model is the coefficient of the length term $\mu$, this term affects the length of the final contour. The smaller $\mu$ is, the longer the final contour is. So if we want to detect as many objects as possible (for example, segment all cells), the $\mu$ should be small. If we are only interested in large objects and do not want to detect small objects (for example, the sub-structure inside the cell), $\mu$ should be large. However, those two requirements contradict each other. We want to segment each cell and don’t want to detect the sub-structure inside the cell. If we choose large $\mu$, the sub-structure is ignored but the cells will be formed by groups. If we choose small $\mu$ in order to separate individual cells, the sub-structure is also detected and will cause some misjudgment (for example, the sub-structure may be treated as background because of similar gray-level). This dilemma also appears in Fig. 3.4. For the choosing $\mu = 0.015 \cdot 255^2$, the large cells are segmented, but there are still some small cells sticking together. And some of the sub-structures of large cell are still detected (the yellow spots inside the blue cells). For classifying different type of cells, this problem is not serious, but this model is not suitable for segmenting clustered cells, unless an appropriate preprocessing method is applied.

![Figure 3.4: Cell Segmentation Using Multiphase Chan-Vese Model: (a) original image; (b) initial condition; (c) final contour (300 iterations), $\mu = 0.015 \cdot 255^2$; (d) final phase](image-url)
Second, the segmentation criteria for the Chan-Vese Model is the gray level of object. For piecewise constant objects, the performance of this model is excellent. When applying in real cell images, the situation is much more complicated. Real cells always contain sub-structures inside them, which will cause a false segmentation. Furthermore, in some cell images, the gray level difference between the cells and background is not distinct. Simply applying this model can not segment individual cells.

3.3 A Preprocessing Method and its Application

The purpose to apply a preprocessing method is to remove the sub-structures of cells. The ideal situation occurs when the whole cell area becomes constant after applying the preprocessing method.

This step has some similarities to noise removal, although the size of noise structure is much smaller. The simplest idea is using Gaussian Kernel: 
\[ G_\sigma(\vec{x}) = \frac{1}{(\sqrt{2\pi\sigma})^N} e^{-\frac{|\vec{x}|^2}{2\sigma^2}}. \]

Convolving the original image with this kernel, the sub-structure less than \( \sigma \) will be suppressed. But the Gaussian Kernel is isotropic, so the edge is also diffused. If the distance between two cells is order \( \sigma \), they will merge together. So the edge preserving filter proposed by Perona and Malik [20] (P-M) is chosen. It is a nonlinear adaptive diffusion process, where diffusion takes place with a variable diffusion coefficient in order to reduce the smoothing effect near edges. The P-M nonlinear diffusion equation is of the form:

\[ I_t = \nabla \cdot (c(|\nabla I|)\nabla I), \quad (3.3.1) \]

where \( c(|\nabla I|) \) is a decreasing function of the gradient. For different choices of function \( c(|\nabla I|) \), the performance is a little bit different, but in general, the effect of P-M filter is selectively smoothing the image, in other words, to keep "edges" in pictures sharp. Perona and Malik proposed two types of function \( c \):

\[ c_1(\xi) = \frac{1}{1 + \frac{\xi^2}{K^2}}; \quad c_2(\xi) = \exp\left(-\frac{\xi^2}{K^2}\right). \quad (3.3.2) \]
Figure 3.5: (a) (Top to Bottom), A Modified Step edge and its 1st, 2nd and 3rd derivatives; (b) A choice of Function Leads to Edge Enhancement, see text for details

In practice, one can also choose other functions like \( c(\xi) = (1 + \frac{\xi^2}{K^2})^{\beta - 1} \), \( \beta \in (0, \frac{1}{2}) \). In this work, \( c(\xi) = c_1(\xi) \).

We take the 1D case to explain the reason why the Perona Malik Filter can enhance the edge. We model an edge as a step function convolved with a Gaussian. Without loss of generality, assume that the edge is aligned with the y axis, and the edge is oriented in such a way that \( I_x > 0 \), as Fig. 3.5(a) shows. Let \( h(I_x) = c(I_x) \cdot I_x \), Eq. (3.3.1) becomes:

\[
I_t = \frac{\partial}{\partial x}(c(I_x) \cdot I_x) = h'(I_x) \cdot I_{xx}.
\]  
(3.3.3)

We are interested in the variation in time of the slope of the edge: \( \frac{\partial}{\partial t}(I_x) \). If \( c(\cdot) > 0 \), the function \( I(\cdot) \) is smooth, then by changing the order of differentiation, we get:

\[
\frac{\partial}{\partial t}(I_x) = \frac{\partial}{\partial x}(I_t) = \frac{\partial}{\partial x} \left( \frac{\partial}{\partial x} h(I_x) \right) = h'' I_{xx} + h' I_{xxx},
\]  
(3.3.4)

In 1D the edge is only one point with maximum slope of inflection, so at this point \( I_{xx} = 0 \), and \( I_{xxx} \ll 0 \) (see Fig. 3.5(a)). Then in a neighborhood of this point, \( \frac{\partial}{\partial t}(I_x) \) has sign opposite to \( h'(I_x) \). If \( h'(I_x) > 0 \), the slope of the edge will decrease with time; if, on the contrary \( h'(I_x) < 0 \), the slope will increase with time. Fig. 3.5(b) shows one example of such function \( h(I_x) = \frac{1}{1+(\frac{I_x}{K})^\alpha} \cdot I_x \), where \( \alpha > 0 \). The parameter \( K \) is a threshold for the intensity gradient: In regions where the gradient is small compared
to \( K, h'(I_x) > 0 \), then \( \frac{\partial}{\partial t} (I_x) < 0 \), the edges will be blured. On the other hand, if the gradient is large compared to \( K, h'(I_x) < 0 \), the filter will de-blur the edges.

These functions defined in Eq. (3.3.2) have the following common characteristics:

1. \( c(\xi) > 0 \) for all \( \xi \);

2. The parameter \( K \) defines a positive critical value \( z(K) \) such that \( \partial_\xi (\xi \cdot c(\xi)) > 0 \) for all \( |\xi| < z(K) \) and \( \partial_\xi (\xi \cdot c(\xi)) < 0 \) for all \( |\xi| > z(K) \);

3. Both \( c(\xi) \) and \( \partial_\xi (\xi \cdot c(\xi)) \) tend to 0 as \( \xi \) goes to infinity.

Therefore, these functions can be used as the candidate function to remove the noises and enhance the edge.

In Perona and Malik’s report, they propose a numerical scheme to solve the Eq. (3.3.1):

\[
I^{n+1}_{i,j} = I^n_{i,j} + \frac{\Delta t}{h} (c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I)_{i,j},
\]  

(3.3.5)

where \( \Delta t \) and \( h \) are the time step and spatial grid size. \( \nabla_N I, \nabla_S I, \nabla_E I, \nabla_W I \) and \( c_N, c_S, c_E, c_W \) are defined as below:

\[
\nabla_N I_{i,j} \equiv I_{i-1,j} - I_{i,j},
\]  

(3.3.6a)

\[
\nabla_S I_{i,j} \equiv I_{i+1,j} - I_{i,j},
\]  

(3.3.6b)

\[
\nabla_E I_{i,j} \equiv I_{i,j+1} - I_{i,j},
\]  

(3.3.6c)

\[
\nabla_N I_{i,j} \equiv I_{i,j-1} - I_{i,j},
\]  

(3.3.6d)

\[
c^n_{N,i,j} = c(\|(\nabla_N I)_{i,j}^n\|),
\]  

(3.3.7a)

\[
c^n_{S,i,j} = c(\|(\nabla_S I)_{i,j}^n\|),
\]  

(3.3.7b)

\[
c^n_{E,i,j} = c(\|(\nabla_E I)_{i,j}^n\|),
\]  

(3.3.7c)

\[
c^n_{W,i,j} = c(\|(\nabla_W I)_{i,j}^n\|),
\]  

(3.3.7d)
This numerical scheme is not exactly the discretization of Eq. 3.3.1, but a similar equation:

$$I_t = (c(I_x)I_x)_x + (c(I_y)I_y)_y$$  \hfill (3.3.8)

This discretization scheme preserves the property of the continuous Eq. (3.3.1) that the total amount of brightness in the image is preserved. Therefore we use this numerical scheme in this work.

Figs. 3.6 and 3.7 compare the use of the preprocessing. Fig. 3.6 is the result applying multiphase Chan-Vese Model without any preprocessing method. Sub-figure (a) is the original image, from which we can see the sub-structures inside the cells. Sub-figure (c) and (d) are the final result. It is difficult to tell which part is cell because of the sub-structure influence.

When applying the Perona Malik method on the original image, the sub-structures are removed and the result is shown in Fig. 3.7 (b). Then applying the multiphase Chan-Vese model, the cells and background are segmented. The blue part is cell and the remaining is background. The program runs 131.468651 seconds for 2000 iterations and the Perona Malik method only needs 0.114614 seconds for 30 iterations. The total operation time is much smaller than the method introduced in Chapter 2.

The Perona Malik preprocessing method performs well for images with high contrast (the gray level difference between cells and background is distinct). If the gray-level of cell is similar to the background, then even after applying the Perona Malik method to remove the sub-structures and noise, the result is still not satisfactory. Furthermore, if the cells in the image are too close to each other or even overlap each other, using this preprocessing method does not give an improved image.
Figure 3.6: Segmentation Result without Preprocessing: (a) original image; (b) initial condition; (c) final contour (2000 iterations), \( \mu = 0.0015 \cdot 255^2 \); (d) final phase

Figure 3.7: Segmentation Result without Preprocessing: (a) original image; (b) result after P-M filter \((K=10, 30\) iterations); (c) initial result; condition; (d) final contour (2000 iterations), \( \mu = 0.0015 \cdot 255^2 \); (e) final phase
Chapter 4

Conclusion and Future Work

In previous chapters, we introduced the level set method and its application on cell segmentation. In this chapter, we will shortly review the work which has been done and try to draw a conclusion based on the results. The methods we tried are only small part of numerous level set methods. There are still a lot of possibilities which may yield acceptable cell segmentations for the ones causing problems in this work. So we will propose some future work, which may improve the performance and make up the drawbacks of current methods.

4.1 Comments and Conclusion

The task for me was to segment cells in image, especially for the case with clustered cells. We chose the level set method to deal with this kind of problems. First, we introduced the model of the traditional level set method and analyzed its property. The traditional level set method can be used in segmenting cells which have clear boundaries and do not stick together. This traditional level set method cannot segment clustered cells, but only detect the boundary for groups.

Then, we introduced the multiphase level set method, combining it with the iterative voting method to segment clustered cells. The iterative voting method detects the seeds of each cell and defines it as the initial condition for the level set method. This
step determines the number of the detected cells and will influence the performance of the level set method. Therefore, the parameters of this step should be chosen carefully. The rule of parameter setting is also discussed in Chapter 2.

The second step is to apply a level set method. For each cell, a level set equation is set up to simulate the seed growing procedure. In order to increase the accuracy, a sequence of level set equations are used. The effect of the first level set equation is to detect the internal boundary of cells. The effect of the second level set equation is to release the constraint and let the fronts grow outside the cells. The effect of the last level set equation is to move inwards the fronts and detect the external boundary of cells. For the whole process, reinitialization is applied every 10 iterations, and the reinitialization part contains 5 steps. After these three steps, the area of cells is detected and can be used for further process.

The drawbacks for this approach are the time complexity and space complexity. To segment \( n \) cells, \( n \) PDEs should be solved, which requires a lot of time. When the number of cells increases, the operation time increases sharply and the requirement for memory also grows. Therefore, this method is not suitable for real time application. Besides, in theory, the evolution of front should stop when it approaches to the boundary, because the gradient near the boundary should approach to infinity. But in practise, the gradient is finite number, if the boundary is ambiguous, the front may not stop exactly at the boundary but pass a little bit. This phenomenon will cause some error in calculating cell area.

The Chan-Vese model can overcome the two drawbacks of the second method. Since the Chan-Vese model doesn’t depend on the boundary detect term to stop the evolution, it can deal with images with or without edges. Besides, in the multiphase Chan-Vese model, only two level set equations are enough to represent all partitions in a domain. In stead of solving \( n \) PDEs in the second method, only 2 PDEs need to be solved. So obviously it will reduce the operation time.
This model has nice performance in detecting different cell types. However, this model still has some limits. One problem is the influence of sub-structures inside cells. The segmenting criterion for Chan-Vese Model is the gray-level of the image. When the sub-structures have different gray-level compared to other area inside cell, the sub-structures may be treated as different objects or even background. This problem can be solved by applying an appropriate preprocessing method. In this work, the Perona Malik filter is chosen as the preprocessing method. The task of the Perona Malik filter is to remove the small sub-structures inside cells, as well as to preserve the edge of cells.

If the gray-level of cells are similar to the background, it is difficult to distinguish cells from background. Besides, if the cells locate too close, or even overlap, the Chan-Vese Model has problems to segment them. The Chan-Vese model doesn’t have a unique minimum, so the initial condition may affect the final result. In practice, an initial condition with many small curves is chosen and appropriate approximation functions for Heaviside Function and Dirac Function are used in order to attain a global minimum.

4.2 Future Work

In this work, we described and tested different level set models for cell segmentation. Some of the methods give good segmentation results, but there is still some extra work to be done. For the multiphase level set method combined with the iterative voting method, the operation time and memory requirement are two major problems. One idea to reduce the operation time is to use a narrow band algorithm, in which only the points near the zero level set are updated and stored. By using this method, the program doesn’t need to update all points in the image, but only calculates the points near the front. The time complexity and space complexity will decrease, however, presenting the location of fronts in the program needs non-trivial work.

The Chan-Vese Model has nice performance in segmenting cells with high contrast.
For images with complicated sub-structures inside cells, this model will have problems. We have tried several preprocessing methods to solve the problem, but the result is not always satisfactory. More appropriate preprocessing methods are still under research. Furthermore, constrained by the CFL condition, the time step of Chan-Vese Model can’t be too large, so in some cases, it needs thousands steps to converge. There are some fast method solving this model without solving Partial Differential Method. So combining the appropriate preprocessing method and fast algorithm for Chan-Vese Model [10], this method will gain better convergence performance.
Bibliography


