Numerical Simulation of Moving Boundary Problem

Ravi K. Vuta
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/etd-theses

Repository Citation
https://digitalcommons.wpi.edu/etd-theses/710

This thesis is brought to you for free and open access by Digital WPI. It has been accepted for inclusion in Masters Theses (All Theses, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.
Numerical Simulation of Moving Boundary Problem

A Thesis

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial requirement for the

Degree of Master of Science

in

Mechanical Engineering

by

Ravi Kiran Vuta
30 April, 2007

APPROVED:

Prof. Gretar Tryggvason, Major Advisor

Prof. Roger Lui, Co-Advisor

Prof. Mark Richman, Thesis Committee Member

Prof. Cosme Furlong, Graduate Committee Representative
Abstract

Numerical simulation of cell motility is one of the difficult problems in computational science. It belongs to a class of problems which involve moving interfaces between flowing or deforming media. Different numerical techniques are being developed for different application areas and in this work an attempt is made to apply two popular numerical techniques used in the field of computational multiphase flows to a cell motility problem.

An unsteady cell motility problem is considered to simulate numerically based on a two-dimensional mathematical model. Two important numerical methods, the Level set method and the Front tracking methods are applied to the cell motility problem to study several cases and to verify the convergence of the solution. With the assumption of no mechanical or physical obstructions to the cell, the results of the numerical simulations show that the domain shapes converge to a circular shape as they reach the steady state condition. The final steady state velocities with which the domains move and the final steady state area to which they converge are observed to be independent of domain shapes. Moreover all shapes converge to exactly same radius of circle and move with same velocity after reaching steady state condition.
Acknowledgements

I would like to thank my academic advisor Prof. Gretar Tryggvason for his excellent guidance and continuous support throughout this work. I would also like to thank Prof. Roger Lui who had given me the chance to work on this project. I also thank Prof. Mark Richman and Prof. Cosme Furlong for their valuable suggestions as members of my thesis committee.

I would like to thank WPI and staff for supporting me with the infrastructure for carrying on computations needed in this work. I would also like to acknowledge the National Science Foundation (Grant DMS-0456570) and WPI for sponsoring this research.
# Table of Contents

Abstract ........................................................................................................ 1

Acknowledgements ....................................................................................... 2

Table of Contents .......................................................................................... 3

List of Figures ................................................................................................ 4

Notation and Units ........................................................................................ 5

Chapter 1 Introduction .................................................................................. 6

Chapter 2 Mathematical Model
   2.1 Model ................................................................................................. 11

Chapter 3 Numerical Implementation
   3.1 Numerical Methods............................................................................... 16
   3.1.1 Level set Method .......................................................................... 16
   3.1.1.1 Re-initialization ....................................................................... 18
   3.1.2 Front tracking Method .................................................................. 18
   3.2 Numerical Implementation
      3.2.1 Computing parameters involved .............................................. 23
      3.2.1.1 Solving PDE for ‘bundle strength’ .................................... 23
      3.2.1.2 Stability .............................................................................. 24
      3.2.2 Level set method ..................................................................... 25
      3.2.2.1 Marker function .................................................................. 26
      3.2.2.2 Gradient of ‘bundle strength’ .......................................... 27
      3.2.2.3 Computing Curvature ......................................................... 29
      3.2.2.4 Computing Perimeter of domain ...................................... 29
      3.2.2.5 Computing Velocity ......................................................... 30
      3.2.2.6 Advecting the Domain ...................................................... 30
      3.2.2.7 Re-initialization .................................................................. 31
      3.2.3 Front tracking Method .............................................................. 32
      3.2.3.1 Marker function ................................................................. 33
      3.2.3.2 Computing gradient of bundle strength ......................... 33
      3.2.3.3 Computing Curvature ....................................................... 33
      3.2.3.4 Computing Perimeter ....................................................... 33
      3.2.3.5 Computing Normal angle ................................................. 34
      3.2.3.6 Computing Velocity ......................................................... 34
      3.2.3.7 Moving the Interface ......................................................... 35
      3.2.3.8 Restructuring front .............................................................. 35

Chapter 4 Results ......................................................................................... 37

Chapter 5 Conclusion ................................................................................... 50

Chapter 6 References .................................................................................... 51
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Schematic representation of steps involved of a crawling cell</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Mathematical model of a cell with involved parameters</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Parameters involved in Front tracking method</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Adding and deleting front elements</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Solving of PDE for bundle strength at grid points near the interface</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Flow chart for Level set method</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>A domain shape and its equivalent distance function</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Initial and modified marker function</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Solution of PDE for circular domain shape</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Curvature field computed for circle case</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Domain shapes with evolution in time</td>
<td>40</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Plot of final curvatures compared for Level set method</td>
<td>42</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>Plot of area of domains with time</td>
<td>43</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Plot of semi-perimeter of domains for Level set method with time</td>
<td>43</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Plot of Velocity of domain vs time</td>
<td>44</td>
</tr>
<tr>
<td>Figure 4.9</td>
<td>Plot of normals computed at front points for circle</td>
<td>45</td>
</tr>
<tr>
<td>Figure 4.10</td>
<td>Curvature computed at front points</td>
<td>46</td>
</tr>
<tr>
<td>Figure 4.11</td>
<td>Evolution of circular shaped domain</td>
<td>46</td>
</tr>
<tr>
<td>Figure 4.12</td>
<td>Normals at front points for ellipse</td>
<td>47</td>
</tr>
<tr>
<td>Figure 4.13</td>
<td>Evolution of elliptical domain</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.14</td>
<td>Plot of normal angle along the boundary points</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.15</td>
<td>Plot of position of domains vs time</td>
<td>49</td>
</tr>
</tbody>
</table>
Notation and Units

\( b \), bundle length \( \mu m / \mu m^2 \)

\( b_0 \), bundle length at cell boundary

\( C_1 \), proportionality constant in Eq. (2.3)

\( C_2 \), proportionality constant in Eq. (2.3)

\( \mathbf{n} \), unit normal vector

\( S \), semi-Perimeter of domain

\( \mathbf{T} \), unit tangent vector

\( V_d \), retraction velocity at the rear

\( V_n \), Normal velocity  in Eq.(2.4)

\( V_p \), rate of Polymerization of actin filaments

\( \kappa \), Curvature \( (\mu m)^{-1} \)

\( \gamma \), depolymerization rate, 1/s

\( \phi \), Normal angle measured from positive x - axis

\( \psi \), Level set function

\( \beta \), proportionality constant  in Eq. (2.6)

\( \Omega \), domain of the cell

\( v \), velocity of cell

\( \partial \Omega \), domain boundary of the cell

\( \sigma \), Cauchy stress tensor

\( \phi \), Normal angle measured from positive x - axis
1. Introduction

Extensive research is going on to solve problems which involve moving interfaces between flowing or deforming media. In this kind of problems, typically, partial differential equations must be satisfied on each side of the interface (often different equations on each side) and these solutions are coupled through relationships or jump conditions that must hold at the interface. These conditions may be in the form of differential equations on the lower-dimensional interface. Often the movement of the interface is unknown in advance and must be determined as part of the solution. Interface shape can be geometrically complex and its topology may change with time. The ability to solve such problems accurately is limited, especially in three space dimensions. Exciting research is currently underway in the development of better algorithms, analysis of the accuracy and stability of such algorithms, and application of these techniques to specific scientific and engineering problems.

The movement of cells along surfaces is fundamental to many important biological processes and has been a subject of intense investigations for several decades. At cellular level, the mechanism of a cell movement involves protrusion at the front part of a cell, graded adhesion of a cell to a surface, followed by detachment and retraction at the cell's rear part. This four-step process is generally accepted by biologists as how eukaryotic cells move [1] which is schematically represented in Figure 1.1 [9]. Many detailed studies have examined each of these steps with sophisticated mathematical and computational models, especially protrusion [2].
Biological scientists have conducted many experiments implementing micro-measurement techniques to study cell motility and shape changes.

![Diagram of cell motility steps](image)

Figure 1.1 Schematic representation of the steps involved in cell motility [9]

However, as of this writing, a complete satisfactory model of an entire cell movement is still lacking. A brief summary of current mathematical models of cell motility is presented here after in this introduction section. Complexity of the cell movement gives rise to mathematical models that are very difficult to analyze.
Consequently, researchers have either looked at special cases of these models, such as by reducing the model to one-dimension by focusing on a narrow strip of the cell's lamellipodium in the direction of movement or by solving the model equations numerically. The functioning of a cell at the molecular level is extremely complex and it is not possible to integrate all the relevant biochemical processes to develop a mathematical model of cell movement. Models of cell motility are rapidly evolving and most of these models have not been rigorously analyzed. The model considered in this work is a two-dimensional one proposed by Lui [4] which is motivated by the one-dimensional model of Mogilner and Verzi [5]. Many difficulties arise in two-dimensions because now the cell has infinitely many directions in which it can move. As of this writing, the only two-dimensional models we are aware of are by those by Bottino and Fauci [6], Bottino, Mogilner, Roberts, Stewart and Oster [7], and Rubinstein, Jacobson and Mogilner [8]. Bottino and Fauci's model uses immersed boundary method to simulate amoeboid movement while Rubinstein et al.'s model uses finite element method to simulate the movement of a fish keratocyte.

In some cases the evolution equation is the only equation that needs to be solved and this computation can be done on a uniform grid, with the interface then captured by computing the location of the appropriate Level set. Complex geometries and topological changes can often be easily handled with this approach. However, in many cases the evolution equation for the Level set must be coupled with other systems of equations being solved on each side of the interface and issues arise in solving these equations and coupling them across the interface. One might use a fixed
grid for these computations with the interface cutting through the grid or use a deforming grid.

Interface fitting is an approach in which the computational grid deforms in order to follow the motion of an interface. This generally requires the global movement of grid points and raises a variety of issues. For example, issues may include deciding and implementing how best to deform the grid, how to couple this motion with interface conditions and how to accurately solve the differential equations on the moving grids. In immersed interface methods on the other hand, a fixed grid is used, often a uniform cartesian grid with the interface cutting through the grid. Standard numerical methods can then generally be employed away from the interface but they require careful modification near the interface to maintain good accuracy and stability in this region.

A variety of different applications requires the solution of interface problems and has motivated considerable work on these methods in recent years. Multiphase flow problems such as interface stability or bubble dynamics give fluid dynamic problems of this nature. Biological fluid dynamics is a rich source of problems with complex geometry and frequently the interaction of fluids with moving elastic structures. The study of blood flow in flexible tubes, cell dynamics and motility, and the functioning of various physiological mechanisms require solving interface problems. Phase change problems such as dendrite solidification or crystal growth lead to problems of this type. Front tracking method is one of the commonly used methods for multiphase flow problems. The use of connected marker points in simulations of an interface goes back to
Daly (1969), who used the points to compute a fluid surface tension in MAC simulations of Rayleigh-Taylor instability, and to boundary integral simulations of free surface potential flows. Richtmayer & Morton (1967) [19] discussed the advection of shocks by shock fitting but neither showed nor referenced any implications. Glimm and collaborators (Glimm, 1982; Glimm & McBryan, 1985; Chern et al., 1986) [20] extended the preliminary ideas of Richtmayer and Morton (1967) into a general methodology for flows with interfaces. For finite Reynolds number multiphase flows, Tryggvason and Unverdi (1992) [21] developed a method which has been applied to many real world applications.

The primary goal of this work is to show that a two-dimensional continuum model of cell motility problem can be solved efficiently by numerical approach. The organization of this report is as follows. Section 2 explains the background of the mathematical model and the numerical methods used. Section 3 explains the numerical implementation of the methods to compute different parameters involved in cell motility model used for the purpose of numerical simulation. Section 4 explains the simulation results for several test cases each with different initial domain shape. This report ends with the conclusion in Section 5.
2. Mathematical Model

2.1 Model

Many crawling cells have length between 10 to 50 microns and consist of a flat lamellipodium at the front edge, which is fraction of a micron in thickness. The lamellipodium is composed of polarized actin filaments that are bundled and networked together and they are responsible for maintaining the cell's shape as well as its movement [9]. The actin filaments are polymers and they undergo constant polymerization and depolymerization. Polymerization at front edge pushes the cell forward. For most eukaryotic cells there are also motor proteins found mostly near the rear of the cell. The motor proteins walk on the actin filaments in one direction causing them to contract, thus pulling the rear end of the cell forward. The relatively thick cell body which contains the nucleus and mitochondria sits near the rear of the cell but it is not responsible for the cell's locomotion. The actin meshwork in the lamellipodium is called the cytoskeleton of the cell. In the model considered, the cell body is ignored and the lamellipodium is modeled as a closed region in \( \Omega \), which we call the domain of the cell.

The two-dimensional model proposed by Lui [4] which is an extension of an earlier one-dimension model is explained in detail here. Inside the cell domain, the density of actin filaments (bundle strength), \( b \) satisfies the mass balance equation

\[
b_t + \nabla \cdot (b \mathbf{v}) = -\gamma b \tag{2.1}
\]

where \( b_t = \frac{\partial b}{\partial t} \) and \( \gamma \) is the rate of depolymerization and \( \mathbf{v} \) is the cytoskeleton velocity. (Notation and units are given in page 5.) As the cell crawls forward, it breaks the adhesion with its surface and there is also
Figure 2.1 Mathematical model of a cell with involved parameters

Resistance to the motion of the actin filaments by the cytoplasm of the cell. These forces manifest itself as drag which is modeled by

\[ \zeta(x,y) \mathbf{v} = \text{drag force on the cell} \]

where \( \zeta \) is called the drag coefficient [7], [5], [11].

Using a force balance [8],

\[ \zeta(x,y) \mathbf{v} = \nabla \cdot \sigma \]  \hspace{1cm} (2.2)

where \( \sigma \) is the Cauchy 2 \times 2 stress tensor. As a first model, one shall make the assumption that \( \sigma \) is isotropic meaning that it remains the same under rotation. Hence
$U\sigma U^T = \sigma$ for any orthogonal matrix $U$. This implies that $\sigma = \theta I$ where $I$ is the identity matrix. In other words, stress is only due to contraction ($\theta < 0$) or expansion ($\theta > 0$) and is the same in all directions. It was further assumed that $\theta = C_1(b_0 - b)$ where $C_1$, $b_0$ are given positive constants. This is to model the fact that actin filaments form bundles and these bundled filaments act like springs which cause the cell to contract [5]. The amount of contractile force resulting from the bundled filaments is proportional to $(b_0 - b)$.

Considering the adhesion to the surface, suppose drag is only due to adhesion and cytoplasm is viscous, then $\xi$ should be an increasing function of its distance from the rear of the cell. This is because adhesion is known to be stronger at the front and weaker at the rear to facilitate the cell's movement forward. In a two dimension problem, it is not clear how to identify the front and rear of the cell. One way to identify the rear is by identifying those boundary points where $v$ is pointing into the cell, i.e., the cell is contracting. In this work, it is assumed that $\xi$ is a constant for the model.

From Eq. (2.2), $v = -\frac{C_1 \nabla b}{\xi}$. Under these assumptions, $b$ satisfies the porous medium type equation

$$
\begin{cases}
  b_t = \frac{C_1}{\xi} \nabla^2 \left( \frac{b^2}{2} \right) - \gamma b & \text{in } \Omega \\
  b = b_0 & \text{on } \partial \Omega
\end{cases}
$$

(2.3)

where $\partial \Omega$ denotes the boundary of $\Omega$. In Eq. (2.3), the domain $\Omega$ moves with time so we need to write down the equation that governs its movement. It is well known [11] that Level set function, $\psi$ satisfies the Hamilton-Jacobi equation
\[ \frac{\partial \psi}{\partial t} + V_n |\nabla \psi| = 0 \]  

(2.4)

Where \( V_n \) is the normal velocity, i.e. the speed at the boundary of \( \Omega \) in the direction of the outward normal \( \mathbf{n} \). From Eq. (2.3) and the maximum principle, \( b \leq b_0 \) in \( \Omega \). Since \( b \) is constant on the boundary, \( \mathbf{n} = \frac{\nabla b}{|\nabla b|} \).

Since, \( \mathbf{v} = -C_1 \frac{\nabla b}{\xi} \) we have \( \nu = \mathbf{v} \cdot \mathbf{n} = -C_1 \nabla b/\xi \).

We should also add the effect of polymerization at the cell boundary and hence \( V_n = V_p + \nu \) where \( V_p \) is the rate of polymerization. Thus we arrive at the formula

\[ V_n = V_p - C_1 \frac{|\nabla b|}{\xi} \]  

(2.5)

The moving boundary problem (MBP) consists of Eq. (2.3), Eq. (2.4) and Eq. (2.5).

Returning to the model, different cells will have different \( V_p \) and it controls the shape of the cell. Since the cells maintain their area as they move, \( V_p \) should depend inversely on the cell's area so that protrusion is slower for larger cells and faster for smaller cells. \( V_p \) may also include a stochastic term to account for the random changes in directions of some cells often observed in the laboratory.

The rate of polymerization \( V_p \) should be positive near the front of the cell and negative near the rear. To prescribe \( V_p \) so that the cell moves in the positive \( x \) direction, we introduce the normal angle \( \phi \) which is defined as the angle between the vector \( \hat{x}_1 = (1,0) \) and \( \mathbf{n} \) at a boundary point. Let \( V_p(\phi) = A + B \cos \phi \) where \( B \) is a positive constant.
Then the maximum value of $V_p$ occurs at $\phi = 0$ where $V_p(0) = A + B$ and the minimum value occurs at $\phi = \pi$ where $V_p(\pi) = A - B$. Therefore, $\phi = 0$ is the front and $\phi = \pi$ is the back. As mentioned above, $V_p$ should be inversely proportional to the area of the cell. Hence it was assumed that $A + B = 2\beta J_1S^2$ and $A - B = -V_d$ where $\beta$ and $V_d$ are positive constants and $2S$ is the perimeter of the cell. Including a negative curvature term in $V_p$ to minimize pseudopods at the cell boundary, the formula for $V_a$ is

$$V_a = \left( \frac{\beta}{S^2} - \frac{V_d}{2} \right) + \left( \frac{\beta}{S^2} + \frac{V_d}{2} \right) \cos \phi - \kappa - \frac{C |\nabla b|}{\xi} \quad (2.6)$$

Lui [4] further demonstrated that under certain assumptions on the protrusion rate, solutions of the MBP converge to a traveling domain solution as $t$ goes to 1. The existence of traveling domain solution has recently been established by Choi and Lui [2]. The MBP may be solved numerically by a variety of methods. In the next sections of this chapter, we illustrate Level set method and Front tracking methods. Level set method is especially suited if $V_p$ depends on the local curvature of the boundary.
3. Numerical Implementation

3.1 Numerical Methods

A variety of moving interface problems requires different solution techniques to be employed based on the nature of the parameters involved. Cell motility problem is similar to the problems one comes across in the multiphase flows in computational fluids engineering. The two important numerical methods employed in this work are popular in simulating fluid flow problems. The two methods implemented here are Level set and Front tracking methods. In the Level set method, an interface is represented by a Level set of some function for which an evolution equation must then be derived and solved. The two methods are explained in detail in this section.

3.1.1 Level set Method

The Level set method, introduced by Osher and Sethian (1988) and further developed by Sussman, Smereka, and Osher (1994) for multiphase flow simulation has emerged as the main alternative to the Volume-Of-Fluid method for the direct advection of a marker function. As other marker function methods, Level set methods make no assumption about the connectivity of the interface. For example, if the interface separating the domain from the surrounding atmosphere should undergo a topological transition, or form a sharp corner or crusp, there is no user intervention or extra coding necessary in order to continue the computation. Besides being robust, Level set methods allow one to accurately represent interfacial normal and curvature.

This section describes the equations and their accompanying discretization for advecting and reinitializing the Level set function. In Level set method, the boundary of the cell is described by the zero level curve of a distance function \( \psi(x, y, t) \) and we let
\[ \psi > 0 \] in the domain of the cell denoted by \( \Omega \) and the interface separating two phases is represented using the Level set function \( \psi \). We can define \( \psi \) to be positive in the domain and negative outside the domain. In other words,

\[
\psi(x, y, t) = \begin{cases} 
+ d, & x, y \text{ in the domain} \\
- d, & x, y \text{ outside}
\end{cases}
\] (2.7)

where \( d \) represents the normal distance to the interface at time \( t \).

The Level set function is then advected using second order Runge-Kutta/ENO method. The Level set function \( \psi \) is advected by

\[
\frac{\partial \psi}{\partial t} + \mathbf{u} \cdot \nabla \psi = 0,
\]

where \( \mathbf{u} \) is normal velocity vector.

\[
\mathbf{u} = \left( V_n \cos \phi, V_n \sin \phi \right)
\] (2.8)

The Level set equation is derived using the fact that the Level set function should be constant along the particle paths. In other words,

\[
\frac{d \psi(x, y, t)}{dt} = 0,
\] (2.9)

For certain simple initial conditions and the values for the velocity \( \mathbf{u} \), one can solve Eq. (2.9) by inspection. For more complicated initial conditions, the values of \( \psi \) and velocity field \( \mathbf{u} \) can be computed by solving Eq. (2.9) numerically.

As we continue in time, the values of Level set function are gradually smeared off and Level set function will no longer be a distance function. But as we need an exact distance function to implement this method accurately, we need to modify the Level set
function to be a distance function after some number of time steps. ENO method is
employed here again, to reinitialize with higher order accuracy.

3.1.1.1 Reinitialization

While the $\psi = 0$ contour is accurately advected in this way, the Level set function
off the interface will, in general, not remain as a distance function. Since preserving the
Level set function as a distance function (at least in the vicinity of the interface) is
important for computing the parameters involved in the mathematical equation for
velocity. Without reinitialization, the magnitude of the gradient of the Level set function
$|\nabla \psi|$ can become very large or very small near the zero Level set of $\psi$. The large
gradients in the Level set function lead to an overall loss of accuracy of Eq. (2.9) and in
variables that may depend on $\psi$.

3.1.2 Front tracking Method

The second numerical method is the Front tracking method. In the Front tracking
method, instead of advecting the marker function by reconstructing the location of the
interface, we can simply use a marker point that is moved with the imposed velocity. We
can also construct smoother approximations to the step function by setting the cell values
as a function of the distance to the interface.

Front tracking, where connected marker points are used to track the boundary
between different fluids or phases, can be done in several different ways. It is necessary
to decide how the front is represented and managed as it stretches and deforms; how the
interface is advanced in time; how the interface interacts with the underlying grid used to
solve the equations governing the problem.
The way in which the front interacts with the underlying fixed grid is what distinguishes between the various Front tracking methods. Several issues must be decided, including how information is transmitted between the front and the grid and how the updates of variables next to the front are accomplished.

The simplest approach is to take the front to represent a smooth transition between the different fluids. Since the interface is accounted for by singular source terms in the governing equations, this corresponds to approximating the singular functions, by smooth distributions on the fixed grid. The origin of this method can be found in many particle-in-cell methods for fluid flows and plasma simulations. Since the role of front is exclusively the advection of the marker function, this approach is very much like a front capturing method with a perfect advection scheme. While information must be passed between the front and the grid, the smoothing of the interface alleviates the need for any modification of the solution method for the equations near the interface. For finite Reynolds number multiphase flows, Tryggvason and Unverdi (1992) developed a method where the front is used both to update the marker function and to include surface tension. Their method that has been used to study a wide variety of multiphase flows and the same technique can be used in simulating other moving boundary problems including the cell motility problem.

3.1.2.1 Tracking the front

The key variables in the implementation of unstructured two-dimensional front are shown in the Figure 2.2. The front consists of points that are connected by elements. For each point, the only information stored is its coordinates. The elements, on the other hand, contain most of the front information. Each element knows about the points that it
is connected to, as well as the local structure of the front, including its neighboring elements. The elements also contain information about the physical properties associated with the interface, such as the surface tension, change in the value of the marker function used to identify the different fluids across the front, and any other quantities that are needed for a particular simulation. The elements are given a direction that defines the "outside" and the "inside" of the interface. For a given interface, all the elements must obviously have the same direction.

![Diagram of elements and corners](image)

Figure 2.2 Parameters involved in Front tracking method

Since we need to add and delete front objects (points and elements) in the course of a simulation, it is easiest to store the objects in a linked list. In a linked list each object contains a pointer to the next object in the list in addition to its intrinsic properties such as coordinates for the points and information about the front structure for the elements. The order within the arrays is completely arbitrary but one object must be designated as the first object in the front. The total number of objects in the front is stored and any operation involving the front is done by starting with the first object. The pointer to the next object in the linked list is then used to move to the next object and so on, until all the
objects have been visited. The unused objects in the array are also linked and a pointer to the first unused object is stored. The use of a linked list makes the addition and removal of objects particularly simple and while it is not absolutely necessary to maintain a pointer to the previous object also (a doubly linked list), deleting objects is simpler if we do. While the exact form of the data structure is not critical for the success of Front tracking computations, it can make considerable difference in the ease of implementation of the method. The data structure for unstructured grids is, of course, well established but the need to dynamically upgrade the front adds considerable complexity.

3.1.2.2 Restructuring the front

In general, an interface will stretch and deform as a result of the motion of interface. When marker points are used to track it, stretching results in an increased separation of the points and eventually it is necessary to insert new points to resolve the interface adequately. When the interface is compressed, the points are crowded together and although it is, in principle, not necessary to remove points, in practice it is generally better to do so to get rid of wiggles that may appear due to crowded points. These wiggles appear due to inaccuracies in computing parameters when using the values fall below the truncation values of numerical values. Figure 3.2 shows schematically a simple restructuring for a two-dimensional unstructured front. In figure 2.3(a), a large element is split by adding a point and in figure 2.3(b), an element is removed. The new point can be put at the mid point between the end points of the element that is being split or removed, but this generally results in poor mass conservation and artificial pressure perturbations for high surface tension. It is therefore better to account for the curvature of the element and use polynomial interpolation to determine the location of the new point.
But when sufficiently small element lengths are maintained consistently throughout the simulation, the elements can be assumed to be straight lines formed by joining the points.

![Diagram](image)

(a) Adding an element by splitting of large element

(b) Deleting front element

Figure 2.3 Adding and deleting of front elements

For unstructured grids, where points are connected by elements, the size of an element determines whether the grid must be refined or coarsened. We delete elements with length less than a certain fixed length which specifies the minimum length of
elements. As elements are formed by connecting successive points, when deleting the
element we actually delete a point which is the end point for the element to be deleted. In
the next iteration, if the element length is more than the maximum value for element
length, the element is split into two elements.

3.2 Numerical Implementation

Computational methodology for computing different parameters involved in the
problem and implementation of the above mentioned numerical methods are explained
more in detail here after in this section.

3.2.1 Computation of parameters involved

3.2.1.1 Solving the PDE for parameter $b$:

The value of parameter ‘$b$’ representing bundle strength inside domain is defined
in the mathematical model by the partial differential equation:

$$\frac{\partial b}{\partial t} = \nabla^2 \left( \frac{b^3}{2} \right) - \gamma b \quad (3.1)$$

Using the FTCS (forward in time and centered in space) numerical scheme for
solving the PDE inside the domain (where the marker function is having positive value
over the grid points,

$$b_{i,j}^{n+1} = b_{i,j}^n + \frac{\Delta t}{2h^2} \left\{ \left( b_{i+1,j}^n \right)^2 - 2 \left( b_{i,j}^n \right)^2 + \left( b_{i-1,j}^n \right)^2 \right\} + \left( b_{i,j+1}^n \right)^2 - 2 \left( b_{i,j}^n \right)^2 + \left( b_{i,j-1}^n \right)^2 - \gamma b_{i,j}^n \} \quad (3.2)$$

The values of $b$ at grid points outside the domain (shown in figure 3.1) are not
defined in the problem and as we need them for implementing the above numerical
scheme for solving $b$ at grid points inside the domain and near to the interface, we do
extrapolation using a linear extrapolating function for computing the values of $b$ at grid points near the interface and lying outside the domain.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.1.jpg}
\caption{Solving of PDE for bundle strength at grid points near the interface}
\end{figure}

3.2.1.2 Stability:

To implement numerical simulation based on this model, two equations need to be solved as we proceed through time iterations for advecting the domain. First one is the PDE which gives the solution of $b$ inside the domain and the second is the advection equation which moves the domain with the computed velocity field. The time step is computed considering the numerical stability limitations of these two equations. To have overall stability, the time step implemented should be the minimum of these two time steps.

We get $\Delta t_1 = \frac{h^2}{4b_0}$ and $\Delta t_2 = \frac{h}{U}$ from the stability conditions mentioned below:
1) Stability condition for solving the PDE, \[ \frac{b_0 \Delta t_1}{h^2} \leq \frac{1}{4} \]

2) CFL condition for the advection equation, \[ \frac{U \Delta t_2}{h} \leq \frac{1}{2} \]

So \( \Delta t_1, \Delta t_2 \) are determined from above conditions and the minimum of these two should be the time step \( \Delta t \) for the simulation.

![Flow chart showing the numerical procedure for implementing Level set method](image)

Figure 3.2: Flow chart showing the numerical procedure for implementing Level set method

### 3.2.2 Level set Method

To solve this problem numerically, a marker function is defined as a distance function. The domain is represented on the grid using the marker function where it has
positive values and the values are negative outside the domain. After initializing the
marker function, we then compute the solution for $b$ by solving the governing partial
differential equation (PDE) inside the domain.

The governing parameters like curvature field, normal angle and velocity fields
are calculated over the entire grid. The marker function is advected with velocity field
numerically using higher order scheme Runge-Kutta / ENO method. A Flow chart of the
algorithm is presented in Figure 3.2.

3.2.2.1 Computing Marker Function by the Level set Method

A marker function is defined as distance function with values of marker function
negative outside and positive inside. The value of marker function at any grid point is
equal to its normal distance from nearest boundary point on the interface. In the Level set
method, the boundary of the cell domain is represented by the zero contour of marker
function. To determine if the grid point falls inside or outside of the domain, we make use
of the winding number. Mathematically, the winding number is given by the equation:

$$N = \frac{1}{2\pi i} \int \frac{dz}{z - z_0}$$

where $z$ is a complex number of form $z = x + iy$.

$N$ has value 1 if the grid point lies inside the domain and value 0 if outside. We make use
of this signed distance function in computing several other parameters. The initial
distance function computed is shown in Figure 3.3 for a circular shaped domain. As we
advect the distance function, it gets smeared off and it no longer remains to be a distance
function as we move in time. To regain the exact distance function, we need to reinitialize the marker function.

### 3.2.2.2 Finding gradient of ‘$b$’

To compute the gradient of $b$ at grid points near and inside the boundary, we make use of the extrapolated $b$ values at ghost points outside the domain. The gradient is then computed using the forward difference numerical scheme illustrated here:

$$
|\nabla b|_{i,j} = \sqrt{\left(\frac{b_{i+1,j} - b_{i,j}}{h}\right)^2 + \left(\frac{b_{i,j+1} - b_{i,j}}{h}\right)^2}
$$

(3.3)

### 3.2.3 Computing Curvature

The curvature at each grid point is computed using the normal vector, from $\kappa = \nabla \cdot \mathbf{n}$

where, $\mathbf{n} \rightarrow (n_x, n_y)$ and is computed from marker function using

$$
(n_x)_{i,j} = \frac{\psi_{i+1,j} - \psi_{i,j}}{h}
$$

(3.4)
\[(n_y)_{i,j} = \frac{\psi_{i,j+1} - \psi_{i,j}}{h} \quad (3.5)\]

Now, the curvature field is found using:
\[
\kappa_{i,j} = \frac{(n_x)_{i+1,j} - (n_x)_{i,j}}{h} + \frac{(n_y)_{i,j+1} - (n_y)_{i,j}}{h} \quad (3.6)
\]

### 3.2.2.4 Computing the Perimeter of the domain

The velocity given by the mathematical model requires the parameter \( S \) which is perimeter of domain. From the Divergence theorem we have:
\[
\int_{\Omega} (\nabla \cdot \vec{F}) dV = \int_{\partial \Omega} (\vec{F} \cdot \vec{n}) dS \quad (3.7)
\]

Making use of this theorem we can compute the perimeter of a 2D domain as shown
\[
\int_{\Omega} (\nabla \cdot \vec{n}) dV = \int (\vec{n} \cdot \vec{n}) dS = \int_{\partial \Omega} dS \quad (3.8)
\]

### 3.2.2.5 Computing Velocity

The velocity field is a function of several domain parameters including \(|\nabla b|\). In this work we assumed that the values of coefficients as value one which simplifies the velocity expression as
\[
(V_n)_{y} = \left(\frac{\beta}{S^2} - \frac{V_d}{2}\right) + \left(\frac{\beta}{S^2} + \frac{V_d}{2}\right) \cos \phi_{y} - \kappa_{y} - |\nabla b|_{y}
\]

The normal angle and curvature are to be calculated at each grid point to compute velocity. The velocity can then be computed at each grid point and then the velocity components can be resolved.

\[
(u_x)_{y} = (V_n)_{y} \cos \phi_{y}
\]
\[
(u_y)_{y} = (V_n)_{y} \sin \phi_{y}
\quad (3.9)
\]
3.2.2.6 Advection the domain

The advection equation is

\[
\frac{\partial \psi}{\partial t} + \mathbf{u} \cdot \nabla \psi = 0
\]  

(3.10)

We advect the domain using higher order accurate ENO method (second order accurate in space). Following the method introduced by Sussman [10], the implementation of advection using ENO method is illustrated below:

Re-arranging the advection equation:

\[
\frac{\partial \psi}{\partial t} = L \psi
\]  

(3.11)

Assuming that \( \psi_{i,j}^n \) and \( \mathbf{u}_{i,j}^n \) are valid discrete values defined at \( t = t^n, x = x_i \), and \( y = y_j \), we advance the solution to \( t = t^{n+1} \) by first finding a predicted value for the Level set function

\[
\psi_{i,j}^* = \psi_{i,j}^n + \Delta t L \psi^n
\]  

(3.12)

and then correcting the solution by

\[
\psi_{i,j}^{n+1} = \psi_{i,j}^n + \frac{\Delta t}{2} \left( L \psi^n + L \psi^* \right)
\]  

(3.13)

The operator \( L \psi \) is discretized by

\[
L \psi = -\left( u_x \right)_{i,j} \psi_{i,j}^{i+1} - \psi_{i,j}^{i-1} - \left( u_y \right)_{i,j} \psi_{i,j}^{j+1} - \psi_{i,j}^{j-1} 
\]  

(3.14)

Where the value of \( \psi \) at the cell boundaries is found by
\[
\psi_{i+,j} = \begin{cases} 
\psi_{i,j} + \frac{1}{2} M(D_x \psi_{i,j}, D_y \psi_{i,j}), & 1/2\left((u_x)_{i+,j} + (u_y)_{i+,j}\right) > 0 \\
\psi_{i+,j} - \frac{1}{2} M(D_x \psi_{i+,j}, D_y \psi_{i+,j}), & 1/2\left((u_x)_{i+,j} + (u_y)_{i+,j}\right) < 0 
\end{cases} 
\]

\[M \text{ is a switch defined by}
M(a,b) = \begin{cases} 
a, & |a| < |b| \\
 b, & |b| \leq |a|
\end{cases} \tag{3.16}
\]

and the differences are found by
\[
D_x \psi_{i,j} = \psi_{i+,j} - \psi_{i,j} \\
D_y \psi_{i,j} = \psi_{i,j} - \psi_{i-,j} \tag{3.17}
\]

The equation for \( \psi_{i,j+1/2} \) is similar.

### 3.2.2.7 Re-initialization

In the re-initialization process \( \psi \) is converted into a new Level set function \( \psi_d \) in which \( \psi \) and \( \psi_d \) share the same zero Level set, and \( \psi_d \) satisfies Eq. (3.18).

\[
|\nabla \psi_d| = 1 \tag{3.18}
\]

for values \( x \) which \( \alpha \) cells of the zero Level set,$$
|\psi_d| < \alpha h
$$

A Level set function that satisfies Eq. (3.19) is called a distance function because \( \psi_d \) is the signed normal distance to the zero Level set of \( \psi_d \).

Sussman, Smereka and Osher (1994) introduced an iterative approach to re-initialize \( \psi \). The advantage of an iterative approach is that if the Level set function \( \psi \) is already close to a distance function, then only a few iterations are necessary to turn \( \psi \)
into the valid distance function $\psi_d$. Following Sussman, Smereka and Osher [10], the re-initialization step is achieved by solving the following partial differential equation,

$$\frac{\partial \psi_d}{\partial \tau} = \text{sign}(\psi)(1 - |\nabla \psi_d|),$$

with initial conditions,

$$\psi_d(x,0) = \psi(x),$$

where

$$\text{sign}(\psi) = \begin{cases} 
-1, & \psi < 0 \\
0, & \psi = 0 \\
1, & \psi > 0 
\end{cases}$$

and $\tau$ is an artificial time. The steady solutions of Eq. (3.19) are distance functions.

The distance function is restored by implementing the equation

$$\frac{\partial \psi}{\partial t} = \text{sign}(\psi)(1 - |\nabla \psi|).$$

The equation is again re-written in form

$$\frac{\partial \psi}{\partial t} = L \psi$$

where

$$L \psi = \text{sign}(\psi)(1 - \sqrt{\frac{\tilde{D}_x}{h}})^2 + \left(\frac{\tilde{D}_y}{h}\right)^2$$

The iterations are carried over the fictitious time with time step, $\tau = \frac{h}{2}$

### 3.2.3 Front tracking method

The Front tracking leads naturally to a very efficient numerical method. In this method, advecting the interface is grid independent in the sense that it needs no grid,
except for specifying the piecewise constant initial data. The initial data can be specified on an arbitrary spaced grid and therefore represented very accurately. Here in this work, the method is applied to several initial shapes starting from simple circular shaped domain to more complicated ones. The initial coordinates of front points are generated using a MATLAB program and this data is saved in data files. These files are input to the FORTRAN code which implements the necessary computations to obtain data for simulations. The code computes all the parameters needed at the grid points and the front point coordinates at each time step and stores them in separate files for post processing and visualization of simulation results.

3.2.3.1 Marker Function

The marker function in the Front tracking method initially is a step function which means that the value of the marker function is equal to one inside the domain and zero outside the domain. To determine which of the grid points fall inside the domain, we again use the winding number theory employed for Level set method at each grid point.

With the marker function which is defined as step function, we cannot identify where exactly does the domain boundary exists between the grid points. All we know is the information of the grid points between which the boundary exists. To determine the exact location of the boundary, we need to modify the marker function near the boundary. The marker function needs to be smoothly varying curve near the interface starting with value of one to zero.
3.2.3.2 Computing $|\nabla b|$ at front points

Once the Eq. (3.1) is solved numerically inside the domain as mentioned in section 3.1, the gradient is found at front points by taking $b$ values from two points along the line which is normal to domain and passing through the front point. First point is taken to be the front point itself where we know the value of $b$ is $b0$ from the boundary condition. The second point location is chosen at a distance $h$ (equal to the distance of grid spacing) and inside of domain on the normal line passing through the front point. The value of $b$ at this second chosen point is determined using area weighted interpolation making use of the $b$ values of cell corners of the cell in which this point falls.

3.2.3.3 Computing Curvature, $K$

Curvature is found along the front points from values of normal and tangent vectors. The normal and tangent vectors at front points are computed as below from x, y co-ordinates of front points.

Computing normal:

$$
(n_x)_i = \frac{y_{i+1} - y_{i-1}}{\Delta S}
$$

(3.22)

$$
(n_y)_i = \frac{- (x_{i+1} - x_{i-1})}{\Delta S}
$$

(3.23)

Computing tangent:

$$
(T_x)_i = (n_y)_i
$$

(3.24)

$$
(T_y)_i = -(n_x)_i
$$

(3.25)

Computing curvature from normal and tangent:
\[ \kappa = n \cdot \frac{\partial T}{\partial S} \]

(3.26)

In discrete form:
\[ \kappa_i = \left( n_x \right)_i \frac{(T_x)_{i+1} - (T_x)_i}{\Delta S_i} + \left( n_y \right)_i \frac{(T_y)_{i+1} - (T_y)_i}{\Delta S_i} \]  \hspace{1cm} (3.27)

3.2.3.4 Computing Perimeter, \( S \)

In Front tracking method, the perimeter can directly be computed by summing up lengths of all the elements looping over the front points.
\[ S = \sum \Delta S_i \]  \hspace{1cm} (3.28)

3.2.3.5 Computing normal angle, \( \phi \)

The normal angle at front points, \( \phi \) is the angle measured from the x-axis to the normal line of domain passing through the front point is computed using,
\[ \phi_i = \frac{\left( n_y \right)_i}{\left( n_x \right)_i} \]  \hspace{1cm} (3.29)

3.2.3.6 Computing Velocity at front points, \( u \)

After computing all the parameters involved in velocity equation, velocity at front points are computed simply substituting all the above parameters into the velocity equation.
\[ \left( V_u \right)_i = \left( \frac{\beta}{S^2} + \frac{V_d}{2} \right) + \left( \frac{\beta}{S^2} - \frac{V_d}{2} \right) \cos \phi_i - \kappa_i - \left| \nabla b \right|_i \]  \hspace{1cm} (3.30)

Components of velocity \( u \), in x, y directions:
\[ u_x = V_u \cos \phi \]
\[ u_y = V_u \sin \phi \]  \hspace{1cm} (3.31)
3.2.3.7 Moving the Interface

The interface is directly advected by moving the front points with computed velocities at each time step. Elements joining the new front points represent the new position of the interface. x, y coordinates of front point are moved as described below:

\[
x_{i}^{n+1} = x_{i}^{n} + (u_{x})_{i} \Delta t
\]

\[
y_{i}^{n+1} = y_{i}^{n} + (u_{y})_{i} \Delta t
\]

(3.32)

3.2.3.8 Restructuring front

As mentioned in Chapter 2, it becomes necessary to restructure the front to maintain accuracy. The element length is used as criterion for decision making about the element to be modified as part of restructuring the front. Maximum and minimum values of element lengths are chosen to be 0.2h and 0.5h, where h is grid spacing distance based on observation for best results. In the numerical experiments, it is observed that setting the minimum length permitted to a smaller value than 0.2h resulted in the formation of wiggles and on the other hand if the maximum length of element is set to more then 0.5h the front shape is observed to be not smooth enough to capture the curved shape of the interface. Restructuring for a three-dimensional unstructured front is complicated one and for implementing it, one needs to employ linked lists in the computer code for numerical simulations. But for two-dimensional case, restructuring can be done in a simple manner if the front points are stored in an array in the exact order as they appear on the front. Connected elements which actually represent the front are simply formed by joining the front points. For example, the first element is formed by joining the first point and second point and second point is formed by joining second and third points. The last element is formed by joining the end point to the first point.
The restructuring which involve deleting of elements can be done simply by deleting the front point from the array of point coordinates and moving all the next values to one step down so that the array consists of continuous values from the starting point to the end point. Deleting a front point makes the number of front points and elements count decrease by one. Similarly splitting of an element is implemented by adding a new point (at middle of the element which is to be split) in the point coordinate array at the correct location. This is done in two steps, first step involves identifying the index value of array where the new point must be stored (to reflect the exact sequence as on the front) and then moving the array values one step ahead starting from index of end point to the marked or identified index value where the new point should be inserted. This moving of array elements will make room for the new point at the exact location. Now the mid point coordinates are calculated from neighboring points (preceeding and succeeding points) and those values are inserted in the array as new point coordinates. Therefore splitting of element makes the count of front points and elements to increase by one.
4. Results

In this chapter, the results of numerical computations using Level set and Front tracking methods are presented. The methods are implemented for several test cases each with a different initial domain shape. With the velocity expression derived in the model, which involves curvature and gradient of \( b \) terms, as we move forward in time during simulation process, any initial irregular shape of domain changes to a circular shaped domain. The domain moves forward in space with velocity field computed over the entire grid in case of Level set method and at front points in case of Front tracking method.

Computations are done for all the test cases until steady state values are observed for parameters. It was observed that for this specific problem with the considered grid resolution and time step values, the steady state values reach after 40,000 time iterations which is around 0.5 units in time. A coarser grid, 90 \( \times \) 45 was considered initially for the computation process and later for more accurate results, all the test cases are implemented using a spatial resolution of a 180 \( \times \) 90 (uniform grid) is considered. The results obtained were verified to be the same when the coarser grid 90 \( \times \) 45 is used.

All of the computations are carried on for both the methods using Fortran code while Matlab is used for post processing of data for the purpose of simulation visualization and extracting plots. The data generated during the computations are stored in data files at each time step and these files are input to the Matlab during the post processing step. The solution of bundle strength, \( b \) obtained after solving the PDE by using an iterative method is shown in figure 4.1. The contour plot of curvature field for a circular shaped domain is presented in figure 4.2.
The values used for the constants during this simulation are:

\[ \gamma = 1.0 \, \text{s}^{-1} \]
\[ \beta = 5.0 \]
\[ b_0 = 20.0 \, \mu\text{m} \, / \, \mu\text{m}^2 \]
\[ V_d = \frac{2\beta}{\pi^2} \]
\[ h = 0.06333 \, \mu\text{m} \]
\[ nx = 180 \]
\[ ny = 90 \]
\[ X - length = 6.0 \, \mu\text{m} \]
\[ Y - length = 3.0 \, \mu\text{m} \]
\[ \Delta t = 4 \times 10^{-5} \, \text{s}^{-1} \]

Figure 4.1 Solution of PDE, \( b \) in a circular shaped domain
The simulation results shows that the domain reduces to circular shape as it moves forward along with time and this is found to be true for all the test cases studied in this work. The normal vectors are computed over the entire grid using the Level set function when implementing the Level set method for this problem and this is shown in figure 4.3 for a circle.

![Curvature field](image)

Figure 4.2 Curvature field, $\kappa$ computed using Level set function for circular shaped domain

The evolution of a domain in time using Level set method is captured at four different times and presented in figure 4.4 (a-d) for four different test cases considered in this work to show how the domain shapes change as they move forward. The circle with radius 0.5 units is considered and the major and minor axes for ellipse are 0.6 and 0.4
units respectively. To solve this problem implementing Level set method, the parameters involved needs to be computed over the entire grid at each time step. For different initial

\begin{figure}
\centering
\includegraphics[width=\textwidth]{initial_domain_shape_circle.png}
\caption{a) Initial domain shape: circle}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{initial_domain_shape_ellipse.png}
\caption{b) Initial domain shape: Ellipse}
\end{figure}
c) Initial domain shape: Peanut shape

d) Initial domain shape: petals shape

Figure 4.4 Domain shapes with evolution of time
domains, the curvature values captured at different points (points where exact boundary intersects the fixed grid) on the domain boundary are plotted in Figure 4.5.

For all the cases, the values of curvature obtained here are observed to be close to 2.5 units along the boundary of domain which gives an idea about the final shape of the domain. As curvature is inversely proportional to radius, a constant value of curvature on the boundary indicates constant radius (circular shaped) domain. The ideal curvature computed along a circle should a straight line when plotted as shown in Figure 4.10 which is just a referred here for indicating constant curvature associated with constant radius. One can note that the curvature value here is different as this plot has different radius (initial radius for circle shaped domain case).

Figure 4.5 Plot of final curvature values along the boundary for petals shaped domain
Area of domain in all test cases using Level set method is observed to be a value of 0.514 sq. units as shown in Figure 4.6. The perimeter is shown in the Figure 4.7.
which shows the results for the considered cases. The plot shows the convergence of the semi-perimeter values to steady value of 1.5 units after some time. The velocities of domains are plotted and are shown in Figure 4.8. They all converge to steady state value of around 3.6 units per unit time.

![Velocity of domain vs Time](image)

**Figure 4.8 Plot of velocity of domain vs time**

The results using Front tracking method for two test cases are presented here. In Front tracking method, the parameters involved are computed only at front points rather than over the entire grid which makes the computation process much faster compared to Level set method. In Front tracking method, maintaining the front is very important and is observed that the domain shape is best represented by the front when distance between
front points is chosen between 0.2 to 0.7 times the grid spacing. If the distance between front points is too small, wiggles appear during the front advection. On the other hand when the distance is too large, front is not smooth enough to represent a curved shape of the domain. In Figure 4.9, computed normal vectors are plotted for a circular shaped domain using Front tracking method at initial time, t=0. These normal vectors are computed using the coordinates information of the front points and is explained in section 3.2.3.3.

![Normals computed at front points for circular shaped domain](image.png)

Figure 4.9 Plot of normals computed at front points for a circular shaped domain-Front tracking Method

To plot the evolution of domain in time, we need information of front point locations at each time step. This information computed during the course of computation process is stored in files for the purpose of post-processing and simulation. As time advances, the domain shape changes its shape until its curvature becomes a constant
value all over the domain boundary and it moves forward in space. The plot of domain shapes at four different times are captured along with their locations and are plotted in

Figure 4.10 Curvature values computed at front points for circular shaped domains-Front tracking Method

Figure 4.11 Evolution of circular shaped domain in time-Front tracking Method
Figures 4.11 and 4.13 for circle and ellipse shaped initial domains. The exact initial shapes of domains are shown in Figures 4.10 and 4.12 over which the normal vectors computed are plotted.

![Figure 4.12 Normals at front points for ellipse shaped domain-FrontTracking Method](image)

The plot of normal angles computed at front points at the end of the simulation is shown in Figure 4.14. The starting front point is chosen at leading edge of domain and the angle is computed moving along the other front points in anti clockwise direction along the domain.
Figure 4.13 Evolution of elliptical shaped domain in time-Front tracking Method

4.14 The plot of normal angle computed along the boundary of domain
The position of domain is computed for two cases at each time step to compare Level set and Front tracking methods. The result plot is shown in figure 4.15. The centroid is of

![Figure 4.15](image)

**Figure 4.15** Plot of position of domains vs time

the Level set function inside the domain which is same as centroid of the domain, is computed for locating the domain position in the case of Level set method. Computing domain position in the Front tracking method is straightforward as we have the coordinates of the front points.
5. Conclusion

Summary

Simulation results prove that the model predicts the cell motility problem correctly and the popular numerical techniques which are widely used in computational fluid dynamics are also applicable to numerically simulate the cell motility problem efficiently. Two numerical methods employed here have their own pros and cons. Front tracking method works very fine even when two interfaces come close to each other. On the other hand Level set method is robust and very easy to implement for complicated shapes. The disadvantage with Level set method is that it cannot identify the two interfaces separately when they come very close, to a distance, less than the grid spacing. The disadvantage with Front tracking method is that it becomes complicated with restructuring of the front for maintaining the accuracy.

Future Work

The numerical methods used in this work are versatile and can be applied to almost all kind of boundary dominated problems in different fields. The Front tracking method implemented in this work can be applied more accurately using advanced techniques taking into consideration different aspects like mass conservation of the domain, using higher order non-linear interpolations in computing parameters involved and using curved lines for connecting front points. Modified versions of Level set method can be considered which are being developed by researchers for implementing these kind of boundary dominated problems to work accurately even when boundaries come very close to each other.
6. References


