Studying the Properties of Cellular Materials with GPU Acceleration

by

Pranav Madhikar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Chemistry

Waterloo, Ontario, Canada, 2015

Abstract

There has always been a great interest in cellular behaviour. From the molecular level, studying the chemistry of the reactions that occur in cell, and the physical interactions between those molecules, to the scale of the cell itself and its behaviour in response to various phenomena. Suffice it to say, that cellular behaviour is highly complex and, therefore, it is difficult to predict how cells will behave or to even describe their behaviour in detail. Traditionally cell biology has been done solely in the laboratory. That has always yielded interesting results and science. There are some aspects of phenomena that, due to cost, time, or other factors, need to be studied computationally. Especially if these stimuli occur on very short or long time scales. Therefore, a number of models have been proposed in order to study cell behaviour.

Unfortunately, these methods can only be used in certain situations and circumstances. These methods can, and do, produce interesting and valid results. Yet there is not really any model available that can be used to model more than one or two kinds of cell behaviour. For example, methods that can show cell sorting do not necessarily show packing. Furthermore, many of the models in the literature represent cells as collections of points, or polygons, so cellular interactions at interfaces cannot be studied efficiently. The goal of the work presented here was to develop a three dimensional model of cells using Molecular Dynamics. Cells are represented as spherical meshes of mass points. And these mass points are placed in a force field that emulates cellular interactions such as adhesion, repulsion, and friction. The results of this work indicates that the model developed can reproduce qualitatively valid cellular behaviour. And the model can be extended to include other effects.

It must also be recognized that Molecular Dynamics (MD) is very expensive computationally. Especially in the case of this model as many mass points are needed in the cellular mesh to ensure adequate spatial resolution. Higher performance is always needed either to study larger systems or to iterate on smaller systems more quickly. The most obvious way to alleviate this problem is too use high performance hardware. It will be shown that this performance is most accessible, after some effort, with Graphics Processing Unit (GPU) acceleration. The model developed in this work will be implemented with GPU acceleration. The code generated in this way is quite fast.

Acknowledgements

Firstly, I must thank my supervisor, Mikko Karttunen, for accepting me into his group. Mikko has been a tremendous source of guidance, wisdom, and expertise for which I will be eternally grateful. It was he who suggested this project to me, helped me develop it and pioneered its inception. I thank him for the support and hope that one day I will be as skilled as he is.

It will quickly become clear throughout this thesis my work could only begin after the hard work and ingenuity of my supervisor Mikko Karttunen, Jan Åström, a collaborator of his in Finland, and Anna Mkrtchyan, an alum of our group. Mikko and Jan first laid down the basis for the model described in this thesis in 2006, later Anna completed their work in 2D in 2014. Only after their effort was I given the opportunity bring their model into the third dimension. And for that I will be always grateful. Jan has continuously been helpful during this project, I thank him for that as well.

The programming work that I have done for this thesis was only possible after the work of Jan Westerholm at the Åbo Akademi University. I am still very much a novice when it comes to programming, and much more so when in comes to CUDA. My work would not have been possible without Jan W.'s efforts and support. I thank him for his work.

I would also like to thank the members of my committee Professor Marcel Nooijen and Professor Pierre-Nicolas Roy for their time and consideration.

Dedication

Newton is often attributed with saying that the reason he saw so far was because he stood on the shoulders of giants that came before him. He spoke of the great scientists that came before him. But I think this can be interpreted in a different way. My parents were the ones who always supported me and encouraged me. I owe all of my success to their relentless hard work and boundless love.

To my parents, Dr. Prabhakar Madhikar and Rukmini Madhikar, for being my giants. And to my brother, Prateek Madhikar, for being my best friend.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ALU	Arithmetic Logic Unit. 45
API	Application Program Interface. 40, 49–51
BPTI	Bovine Pancreatic Trypsin Inhibitor. 14
CAM	Cell Adhesion Molecules. 13
CPM	Cellular Potts Model. 26–28
CPU	Central Processing Unit. 2, 3, 15, 19, 36–41, 44, 54
CUDA	The Compute Unified Device Architecture. 3, 40, 41, 43, 45–47, 49, 50
DOD	Delaunay Object Dynamics. 24–26, 29
DRAM	Dynamic Random-Access Memory. 45

ECM

Extracellular Matrix. 93

EXPResSO

Extensible Simulation Package for Research on Soft matter. 14

GCC

the Gnu Compiler Collection. 41, 43, 48

GFLOP/S

Giga FLoating point OPerations per Second. 39

GPGPU

General Purpose GPU. 41

GPU

Graphics Processing Unit. iii, vii, xiii, 3, 15, 19, 36, 37, 39–41, 43, 44, 53, 77, 78, 92, 93

GROMACS

GROningen MAchine for Chemical Simulations. 14

HOOMD-blue

Highly Optimized Object-Oriented Molecular Dynamics. 14

HPC

High Performance Computing. 37

HSA

Heterogeneous Systems Architecture. 40

ICC

the Intel C Compiler. 41, 43

LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator. 14

MD

Molecular Dynamics. iii, 2, 3, 14–16, 19, 33, 36, 37, 41, 48, 50, 92, 93

MI

Mitotic Index. 35, 78, 84

MIC

Multiple Integrated Core. 93

MMTK

Molecular Modelling ToolKit. 14

MOSFET

Metal-Oxide-Semiconductor Field-Effect Transistor. 37

NAMD

NAnoscale Molecuar Dynamics. 14

OOP

Object Oriented Programming. 46

OpenCL

Open Compute Language. 3, 40, 41, 43, 45, 49, 50

P^{3}M

Particle Particle Particle Mesh. 18

PBC

Periodic Boundary Conditions. 17

PME

Particle Mesh Ewald. 18

SIMD

Single Instruction Multiple Data. 49

SM

Stream Multiprocessor. 44, 45, 53

SP

Stream Processor. 45

SPME

Smooth Particle Mesh Ewald. 18

TDP

Thermal Design Power. 37

VDW

Van der Waals. 16

Chapter 1

Introduction

The human body is said to contain roughly 100 trillion cells of varying types and functions [1, 2]. The number of cells is very great, but they all come from a single fertilized cell that contains all of the information needed to create the entire body. The development of all of these kinds of cells from the single to the embryo stage is not understood completely[3–6].

The cell is the fundamental building block of all living organisms, both for single celled organisms such as bacteria and multi-cellular organisms such as animals and plants. In fact, most of the base functions of life occur at the cellular level and rely upon the cells' ability to evolve, regenerate, and replicate. The cell embodies all of Koshland's seven pillars of life[7].

Koshland's Seven Principles of Life

1. **Program**
   The ability to efficiently store, retrieve, and apply the information to apply the remaining pillars of life.

2. **Improvisation**
   The ability the adapt the Program to changes in the environment. Such as evolving immunity to a virus.

3. **Compartmentalization**
   The capacity for certain parts of a living organism to specialize and increase its

productivity for certain activity. For example, separate digestive and immune systems would be more effective than a single system that does both.

4. **Energy**
   The capability of a life form to obtain, regulate, and store energy efficiently.

5. **Regeneration**
   Any life form that expects to survive must be able to heal in the face of injuries, but must also be able to remove waste as a result of energy gathering and storage.

6. **Adaptability**
   Life should be able to react to protect itself when the process of improvisation is too slow. Seeking medicine to treat disease, for example.

7. **Seclusion**
   The different chemical process of life are extremely complex, it is therefore necessary to have special systems that regulate each process separately to avoid confusion and errors.

This author submits that cell behaviour can be roughly divided into three categories: cell division, migration or motion, and function. Cell division and migration apply to most cells in a general way. Cell function, though, is highly specific to the cell type itself (e.g. neurons and liver cells). The work described in this thesis will focus on cell division mostly. The modelling of the development of cell function and migration is out of the scope of this thesis.

The goal of the work presented here is to describe an approach to modelling cell behaviour that encompasses as much of the behavioural spectrum of cells as possible. This will be done with Molecular Dynamics (MD). MD has already been used extensively to study the behaviour of complex biological behaviours such as proteins [8], lipids [9, 10], carbohydrates. However, the problem with MD is its high cost when it comes to modelling complex systems. It is not uncommon for runtimes of many weeks or months to complete a single simulation.

In order to reduce simulation run times, it is normally necessary to run these MD simulations on very large systems with Central Processing Units (CPUs). However, this is very expensive as these systems are very expensive to maintain and are also not very environmentally friendly as they require large amounts of power. Hence, it is desirable to use other methods of accelerating simulations on simple workstations or PCs.

A good way of accelerating these simulations is to use Graphics Processing Unit (GPU) acceleration technologies such as the The Compute Unified Device Architecture (CUDA) or the Open Compute Language (OpenCL). These technologies leverage the hardware capabilities of GPUs, or in the case of OpenCL other processor types as well, to greatly increase the run rate of simulation code. In some cases a speed up of up to one or two orders of magnitude is possible [11–13]. However, in reality, the performance of GPUs is about 2–3 times (in best case scenarios approximately 10 times) CPU performance, which is still significant [12, 13]. The modelling technique described here will be accelerated on GPUs with CUDA by combining the CPU and GPU to maximize performance.

The simulation method described in this thesis is based on a simple model of cells as closed loops of mass points that was introduced in 2006 by Karttunen and Åström [14]. Later, Mkrtchyan, Karttunen, and Åström added to this model and were able to reproduce some cell behaviour [15]. So far, the model was always two-dimensional. An extension in to the third spatial dimension was the next logical step. The work presented in this thesis is towards developing such a 3D model.

Structure of this Thesis

Each chapter will focus on one aspect of the work presented in this thesis. First, in Chapter 2, some information regarding the behaviour of cells is presented. That chapter will hopefully give readers an adequate primer on cell behaviour to be able to understand the rest of the project. A detailed knowledge of cell biology is not required, though it may be helpful.

Chapter 3 will described some modelling techniques used to model cell behaviour. Quite a few computational models of cell behaviour. Unfortunately, many of them have certain limitations in how they represent cells and, as a result, can only be applied in particular situations. The latter part of Chapter 3 describes the 2D model by Mkrtchyan et al. [15] that the work in this thesis is based upon.

Next, we switch gears slightly and look at GPU acceleration in Chapter 4. GPUs can be vastly superior to CPUs in certain applications. It turns out that certain parts of MD are perfect candidates for GPU acceleration. The reason to pursue GPU acceleration will be described. Then, some basic ideas of multi-threading on GPUs will be shown and some examples given.

In Chapter 5 the methodology of the 3D model of cells will be described in greater detail. That chapter will be ideal for readers who are most interested in the implementation. The basics of the simulation algorithm will be described as well.

Chapter 6 will show some interesting results that were created with the 3D code. As the 3D model is an extension of the 2D model, the first step of validation is comparing between the 3D and 2D model. The 2D model is particularly amenable to modelling epithelia (planar tissue) [15], so the comparison is done there.

Finally, Chapter 7 describes some conclusions that can be derived from the work done for this thesis. In addition, the plan for this project in the future will be discussed. The simulation code is, while adequate for replicating the 2D code, can still be improved in performance, and many more features can be added to be able to study more complex forms of cell behaviour.

Chapter 2

Background Information

2.1 Cell Structure

Cell structure can vary greatly from cell to cell [1, 16]. This section will show the structures of some simple cells. There are two kinds of organisms, prokaryotes and eukaryotes. Bacteria and archaea belong to prokaryote family. All animals, plants, and fungi belong to the eukaryote family.

Figure 2.1 shows a typical bacterium or prokaryote. This type of cell tends to have diameters of 1-10 µm[17]. Notice that there is no compartmentalization evident in this type of cell. All genetic information is stored in a nucleoid region with coiled DNA and cell functions happen more-or-less uniformly throughout the cell. All of this is enclosed in a plasma membrane which is encapsulated by a harder capsule. These cells can have flagella that exert control on the cell's motion.

Figure 2.2 shows a typical animal cell. These cells tend to be much larger than prokaryotes at 10-100 µm. In some cases these cells may have flagella, however most animal cells do not control their motion in this manner. Animal cells tend to be either stationary or moved passively such as blood cells. There are, nevertheless, some types of animal cells that move by selectively expanding the body, anchoring a part of their body, then contracting the free part of the body.

The animal cell is enclosed by a cell membrane that gives the structural integrity of the cell and controls what can move into and out of the cell. There is the plasma membrane that encloses the whole cell and other membranes that constitute the boundaries of the organelles such as the nuclear envelope of the nucleus [16, 18].

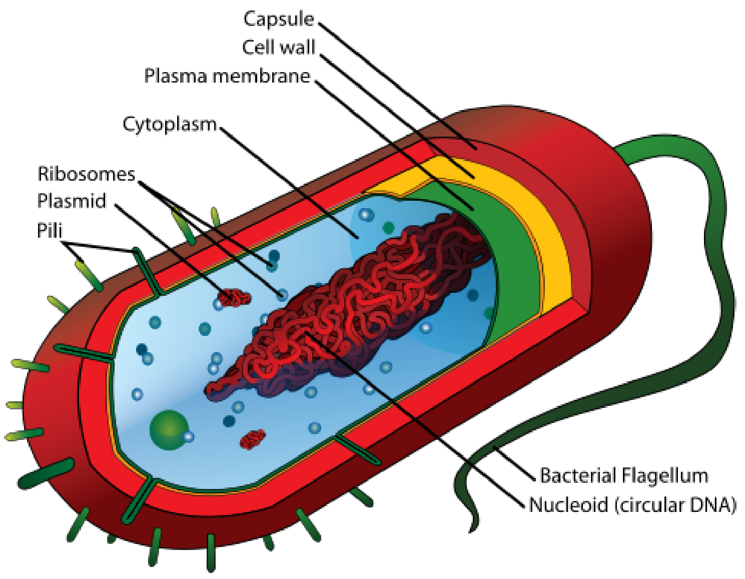


Figure 2.1: The structure of a typical bacterial cell which is a prokaryotic organism. Image taken from the public domain. By Mariana Ruiz Villarreal, LadyofHats [Public domain], via Wikimedia Commons.

The cell membrane is an extremely complex component of cells [16, 18]. It is made of a mixture of various lipids and proteins. These components also includes apparatus that connect the cell to its neighbouring cells or tissue. It is composed of a bilayer of amphiphilic molecules named lipids. Proteins are a major component of lipid membranes and give rise to most of the functionality of the membrane such as transport through the membrane, cell-cell communication, and anchoring to the surrounding tissue.

The Cytoskeleton

The structural strength of the cell comes from protein filaments that span throughout the cell as a mesh of interconnected fibres known as the cytoskeleton. This structure not only gives the cell its shape and mechanical properties, but is also essential in the transport of vesicles and organelles during cell division. The cytoskeleton is also essential for governing cell motion [1, 2]. The proteins that form the cytoskeleton can be categorized into three classes.

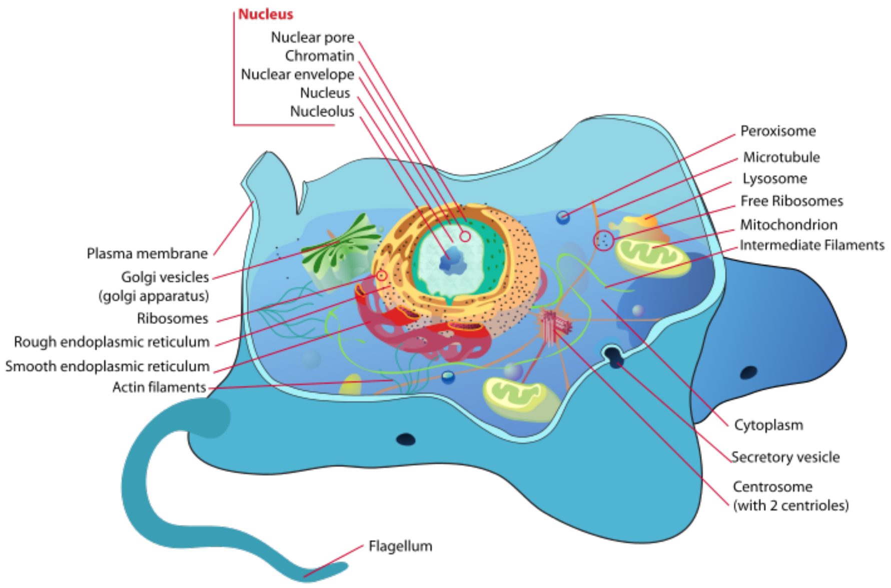


Figure 2.2: A typical animal cell. This eukaryotic cell has a much more complex structure. Unlike the prokaryotic cell, this one has specialized organelles with more complex structure. Also unlike eukaryotes, the DNA is kept inside the nucleus. Note that most animal cells do not have flagella. By LadyofHats (Mariana Ruiz) [Public domain], via Wikimedia Commons.

1. **G-actin and related microfilaments**
   These filaments combine to support the cell membrane. G-actin (Globular-actin) is a single chain of 375 amino acids with molecular mass of approximately 42 kDa. G-actin polymerizes into a chain named the F-actin filament which has a thickness of roughly 8 nm. These polymers are somewhat flexible with a persistence length of 3-17 µm [19]. These filaments can cross-link to create two dimensional networks. These networks tend to be roughly uniform with a constant average distance between cross linked proteins. Figure 2.3 shows a sketch of such as mesh.

2. **Intermediate filaments**
   Intermediate filaments are thicker than F-actin filaments and have a more complex hierarchical structure. In animal cells that lack cell walls, these filaments given the cell its mechanical strength [20]. Each intermediate filament is a cylindrical arrangement of protofilaments which are themselves composed of pairs of helical polymers.

3. **Microtubules**
   Microtubules are the thickest class of structural proteins. These are structures made of α-tubulin and β-tubulin. This unit is 8 nm in length and has a molecular mass of about 100 kDa. The overall microtubule can have a mass of about 160 kDa/nm.

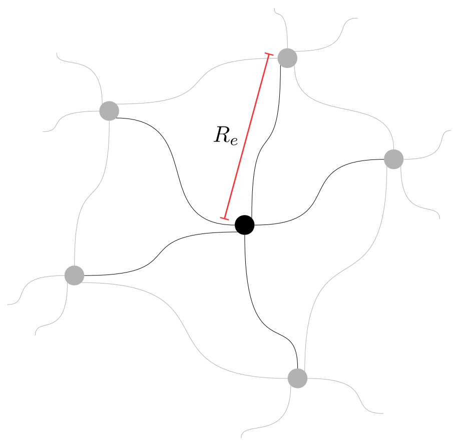


Figure 2.3: This sketch shows the mesh like structures generated through the cross-linking of F-actin filaments. This gives a good approximation for the structure of cell cortex.

Obviously, cells are incredibly complex things. Simulating a full cell would be extremely difficult, if not completely impossible. The way such an object reacts to stimuli would be impossible to simulate regardless of how much computational power is available. Simplifications are needed to be able to simulate such systems. Fortunately, as we will see in later chapters, the behaviour of the cell can be approximated. Thus cell behaviour can be studied *in silico* and compared to experimental results.

2.2 Cell Division

Cell division itself is a highly complex process that differs between cells of different organisms[21, 22] and is controlled by many different conditions [23–26]. Furthermore, inconsistencies can introduce many variations that further complicate the behaviour of cellular systems. These variations may have useful effects such as morphogenesis [27–30] and detrimental effects such as cancers or other diseases [31–33] when they occur erroneously.

Walther Flemming first observed cells in mitosis in the 1880s [34], and cell division been an intense topic of research ever since. Much work has been invested into studying the molecular mechanics of mitosis including what regulates it and which proteins participate in it [5]. He coined the term "mitosis" which comes from the Greek work for thread, *mito*, after noting the thread like structures of dividing cells. The process of mitosis is a multi-step process that depends on many factors within the cell and its environment. First the cell grows if there are enough nutrients in the system. Once it then reaches appropriate size, the internal mechanisms of cell division are started. This includes the replication of DNA and the creation of proteins that structurally partition the cell into two daughter cells. Cell mechanics ensure that mitotic cells have a roughly spherical shape[35, 36], they define the division plane[5, 37, 38], and even govern the changes in the cell shape by manipulating the cell cortex. The cells proliferate to create the various organs in the human body.

The cell cycle is the series of events that lead to duplication and replication of a cell. Figure 2.4 shows a typical cell cycle. The goal of the cell cycle is to produce two daughter cells that are accurate copies of the parent. There is a continuous growth cycle with accompanying increase in cell mass and volume, and a discontinuous division cycle in which DNA is replicated and distributed to the daughter cells.

While bearing in mind figure 2.4, we can then see what the detailed steps involved in cell division are. The cell cycle is divided into three phases[16, 39]: M-Phase, Interphase, and cytokinesis.

1. **Interphase**
   This is the part of the cell cycle in between subsequent mitosis events. The cells grow and replicate their DNA in this phase which can further be divided into 3 steps:

   (a) G_1: The first gap phase
   This is the longest and most variable portion of the cell cycle. When cells enter this portion of the cell cycle, they are normally half the size of their parent cell. They grow to maturity during this phase. At the beginning of this phase, all of the mechanisms that govern replication and division are halted until the restriction point. The restriction point is when the cell checks the supply of nutrients before it starts the remaining steps before mitosis. Once it is determined that there is sufficient nutrient supply, the cycle continues.

   (b) G_0: Differentiation and growth control
   After the organism has developed sufficiently, most cells differentiate into a new G_o state and no longer divides. The cell is still highly active doing things other than division and can be motile as well. This state is not permanent, the cell can reenter G_1 phase to divide again.

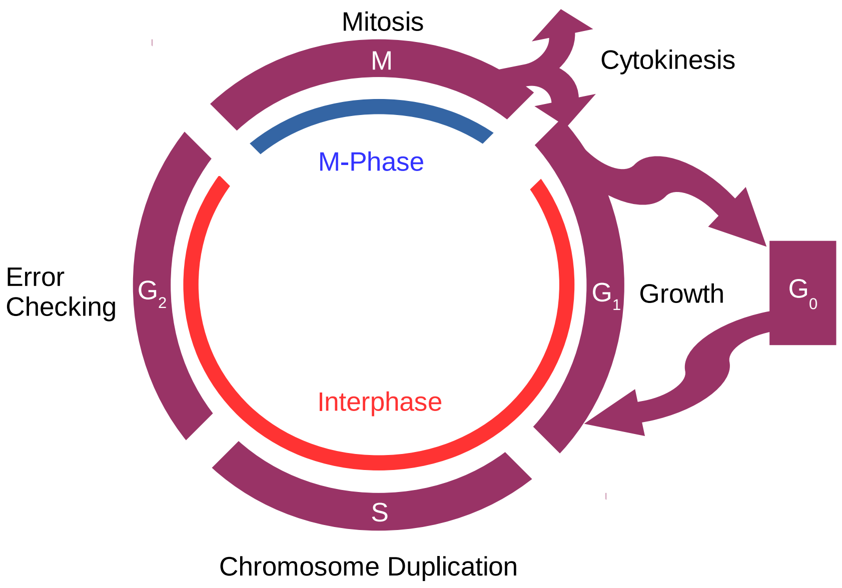


Figure 2.4: The different phases of a eukaryotic cell. During the G_1 phase the cell gains mass and volume. Chromosome duplication occurs during the S phase. The DNA error checking mechanism is started during the G_2 phase. Finally, the cell is divided during the M phase. Adapted from Figure 40–2 in [16].

(c) S: Chromosome replication
Due to complexity of eukaryotic DNA, replication must be done in a highly controlled manner. Replication happens from multiple points of origin. The duplicate copies of DNA copied at each point of origin are name "sister chromatids" and remain linked until they are separated during mitosis.

(d) G_2: The second gap phase
This is a relatively brief period during which the replicated DNA is checked for damage or errors. Enzymatic activity specific to mitosis accumulates and triggers mitosis once a threshold is reached.

2. **M-Phase**
Once the DNA has been replicated, the cell segregates them on opposite ends of a cellular scaffold named the mitotic spindle. After the segregation, cytokinesis, the

process of cleaving the cell into two daughter cells begins. This phase proceeds in five steps.

(a) Prophase: Formation of the mitotic spindle
A change in the properties of the cytoskeleton cause it to separate and form two poles of the mitotic spindle.

(b) Prometaphase: Nuclear breakdown
The nucleus breaks down and the sister chromatids and begin to move towards the centre of cell roughly half-way between the two cell poles.

(c) Metaphase: Mid-point of the process
At this point the chromosomes are roughly halfway between the two cell poles. They form a collection of chromatids named the metaphase plate.

(d) Anaphase: Sister chromatid seperation
The chromatids separate and begin to move opposite spindle poles and the poles themselves begin to move apart. The cell cortex is also activated to begin cleaving the cell into two.

(e) Telophase: Nuclear envelope reformation
The separated chromatids are now enveloped in new nuclear membranes.

3. **Cytokinesis**
Once all of the genetic information and organelles have moved into the regions of the parent cell, a contractile ring of protein fibers (actin and myosin) forms about the equator of the parent cell. The ring contracts, forms a cleavage furrow and pinches the cell into two daughter cells. Figure 2.5 shows the onset of cytokinesis. The contractile ring continues to contract until the two halves of the parent cell are "pinched off" of each other. This process is a delicate balance between the internal pressure of the cell (aka the turgor pressure in plants and fungi) [16, 40, 41], the plastic nature of the cell membrane, and the forces of the cytoskeleton. It is theorized that similar mechanisms contribute to cell migration[37].

2.2.1 The Division Plane

The selection of the division plane is another interesting aspect of cell division. The division plane of the cell is simply the plane in which the contractile ring lies. Or in other words, it is the plane at which the parent cell is pinched off to make daughter cells. The orientation of cell the division plane is required to generate complex multi-cellular organisms[42].

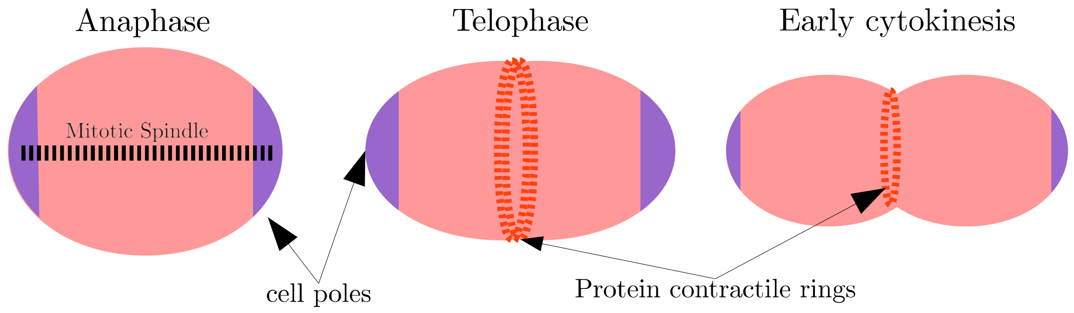


Figure 2.5: After the chromatids have been correctly separated a the end of anaphase, a contractile ring of actin forms roughly halfway between the two cell poles during telophase. This concludes the activities of M-Phase. Cytokinesis begins with the protein ring contracting and ends when the two halves of the parent cell are pinched off of each other.

Orientation of the division plane can have significant bearing on development and cell differentiation[43].

The selection of the cell division plane can vary from cell type to cell type. The shape of the cell can affect division plane [42] in addition other factors such as the alignment of the molecular spindle [44]. There may be more than one force that act on the centering of the molecular spindle.

Given all of these factors that can affect the division plane orientation, we can look at what different division planes are possible and effects they may have. The first type of division plane is selected by Hertwig's rule, also known as the “long axis rule” [45]. Hertwig concluded: “The two pole of the division figure come to lie in the direction of protoplasmic mass”. That is the mitotic spindle lies along the longest axis of a cell. Then, the division plane is chosen roughly mid-way, perpendicular to this axis.

In addition to reproduction where the cell is divided in to cell of equivalent size, it is possible to have asymmetric division[30, 38]. In this case the one of the daughter cells is significantly larger than the other[32, 35, 46, 47]. The division plane may depend on the alignment of the mitotic spindle, if the spindle is misaligned with respect to the axes of the cell, then the division occurs such that one daughter is smaller than the other. However this is not always the case, sometimes mitosis begins in a symmetric way, then at some point after the formation of the cleavage furrow, one half the cell expands and the other contracts. A consequence of asymmetric cell division is that the daughters may receive different kinds of intrinsic cell-fate determinants which play a role in differentiation[47]. Some kinds of stem cells are known to differentiate from reproducing through asymmetric cell division[48], and producing new cell types. We will see later that in the 3D model developed in this thesis, the division plane is set randomly and symmetrically (see Section 5.3 and figure 2.6). Other division planes are planned for the future. In the 2D model on which the 3D model is based, introduced by Mkrtchyan et al. [15, 49], all three division schemes were studied. So the model presented in Chapter 5 is amenable to modelling different division planes.

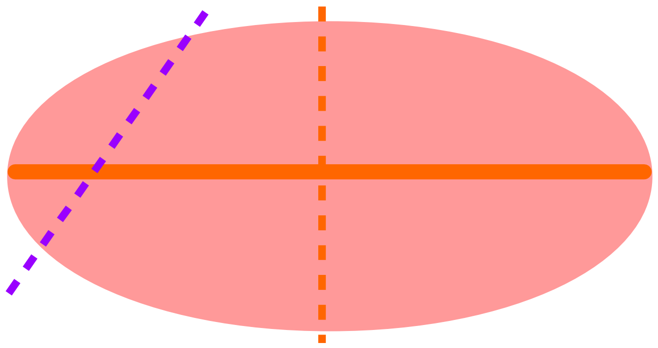


Figure 2.6: Division planes showing symmetric and asymmetric division. The bold orange line is the mitotic spindle. The orange dashed line shows the symmetric division plane. The purple dashed line shows an example of an asymmetric plane line.

2.2.2 Inter-Cellular Adhesion

The adhesion forces between cells play an important role in their interactions and behaviour [50–55]. Cells may adhere to substrates, the extracellular medium, or other cells. The complex structures of cell membranes make it difficult to describe accurately. There may be two sources for the adhesion interaction between cells: van der Waals forces, and site specific adhesion mediated by Cell Adhesion Molecules (CAM) [50]. Variation in cell adhesion is known to have effects on the geometry and function of tissues [54, 56]. At longer ranges, the adhesion is caused by van der Waals interactions and electrostatic interactions between lipids with charged head groups. As membranes of different cells come into close proximity with each other, steric hindrance and thermal undulations of the membranes cause repulsion.

Stronger adhesion interactions occur through specific interaction sites between CAMs. CAMs are molecules embedded in the cell cortex that interact with CAMs of other cells or the extracellular [50]. These CAMs may be spread evenly over the surface of cells or at specific junctions that induce some control over the structuring of tissue [50].

Due to the complex nature of cell behaviour, especially cell reproduction, there is a need for theoretical frameworks to study cells and provide predictions. Quite a few models have been proposed to meet this need[15, 34, 57–64]. Some of these are described in Chapter 3.

2.3 Molecular Dynamics

Molecular Dynamics (MD), is a very powerful and highly used method to model matter at the molecular level. With this method, we can simulate systems in states of equilibrium and non-equilibrium [65]. Over the past several years, many software packages have emerged that can run MD simulations of very large systems efficiently. These include packages such as the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [66], DLPOLY [67], the GROningen MAchine for Chemical Simulations (GROMACS) [68], the NAnoscale Molecuar Dynamics (NAMD) [69], the Extensible Simulation Package for Research on Soft matter (EXPResSO) [70], the Molecular Modelling ToolKit (MMTK) [71], and the Highly Optimized Object-Oriented Molecular Dynamics (HOOMD-blue) [65]. Most of these packages support multithreading on multi-core CPUs and parrallelization across multiple computational nodes with multiple CPUs. Some of these packages such as GROMACS and HOOMD-blue support acceleration on GPUs as well.

All of the packages above are fundamentally alike in that they use the same algorithms that have were first developed in the 1950–1970 in theoretical physics; these methods have been evolving ever since. When first developed, these methods were primarily used to study simple atoms[72–75].

One of the very first MD simulations was run by Alder and Wainwright in 1957[72] that used a hard sphere model where all atoms interacted through perfect collisions and studied their phase transition. Later, Rahman applied a continuous potential that made the atoms behave more realistically [73]. Rahman showed that this method could reproduce the properties of Argon at 94.4 K. The very first protein was first simulated with MD in 1977 by Karplus al. [76]. The Karplus group simulated the dynamics of Bovine Pancreatic Trypsin Inhibitor (BPTI). These simulations were all conducted with less than 1000 atoms, and since then the number of atoms that are routinely simulated has grown rapidly to the point that simulations with 10^4–10^6 atoms have become commonplace[8].

Furthermore, the very first simulations could only simulate on the order of 10's of ps at a time due to limitations of computational power. Nowadays it is common to run simulations up to the µs range with simulated domain sizes on the order of $1\,\mu m^3$ thanks to the vast improvements in computational power and methodological techniques. There has been a great deal of work in studying the properties of, among other systems, lipid aggregates[9, 10, 77], lipid monolayers[78, 79], bilayer pore formation [80], protein binding [81, 82], and enzyme activity [83].

Despite these vast improvements, there is still a never ending demand for larger simulations that are run for longer. Therefore, there is a great deal of interest for parrallelization of MD codes over multiple Central Processing Units (CPUs) and Graphics Processing Units (GPUs). This demand coupled with the fact that access to supercomputers with many CPUs is difficult and expensive has increased the interest in GPU acceleration which are cheaper to use and easier to operate. The technology that has been developed to run MD over multiple CPUs can be reused to run on multiple GPUs if needed.

The aim of MD simulations is to compute macroscopic behaviour form microscopic interactions[84]. When the macroscopic behaviour can be reproduced computationally by simulating microscopic systems, the simulations can be used to study the macroscopic systems computationally. MD furnishes the scientist with the capacity to study real and theoretical system as well.

2.3.1 Methodology

A simulation method, henceforth called the model, needs to be both correct as far as its results go and also needs to be tractable. That is, it must use minimal resources (memory and/or processing power). The most expensive part of any model is the complexity of the interactions between the particles in a system described by a potential energy equation. These equations are approximations of the fundamental laws of physics that govern real atoms. The approximations are made due to some assumptions that simplify the interactions somewhat.

With elementary mechanics, the force on a particle i is defined as function of the potential energy that a particle feels, U as shown in (2.1).

$$\mathbf{F}_i = -\nabla U \tag{2.1}$$

If U is defined correctly, the force on any particle can be calculated. When defining U, it must be first decided what interactions will be modelled. A valid simulation is completely dependent upon the accuracy of the potential energy landscape of the system. Simultaneously, the potential energy functions must remain tractable to not overwhelm computational resources. U is broken down into a sum of different potential functions U_{int} depending on the type of interaction,

$$U = \sum_{\text{int}} U_{int}. \tag{2.2}$$

Typically when modelling molecules, there will be non-bonded interactions between particles. If the particles are molecules, then the interactions between the atoms of the molecule will have what are known as bonded interactions. The bonded interactions are approximated by harmonic potentials (Eq. (2.3)), harmonic bending potentials (Eq. (2.4)), and torsional interactions (Eq. (2.5)) that can be of the Ryckaert-Bellemans [85, 86] type. Bonded interactions have quite a short range, not extending beyond three to four times the bond length. They are also highly anisotropic and require a bond to exist to be valid. No chemical reactions are modelled in a strictly MD simulation so these potentials depend entirely on the definition of the molecules' bond lengths, angles, and dihedrals which are part of the initial conditions of the system,

$$U_S(r_{ij}) = \frac{1}{2}\left(r_{ij} - r_{ij}^o\right)^2 \tag{2.3}$$

$$U_B(\theta_{ijk}) = \frac{1}{2}\left(\theta_{ijk} - \theta_{ijk}^o\right)^2 \tag{2.4}$$

$$U_T(\phi_{ijkl}) = \sum_{n=0}^{3} S_n(\cos(\phi_{ijkl} - \pi))^n. \tag{2.5}$$

where r_{ij} is the bond length between particles i and j, θ and ϕ are the bond and dihedral angles. The variables denoted by a "naught" are the equilibrium values of the same. The constants K_S, K_B, and S_n are constants that can be approximated empirically and depend on the system being modelled.

Inter-molecular interactions include Van der Waals (VDW) interactions and electrostatic interactions. These type of interactions extend well beyond bonds are also isotropic. They are approximated more completely than bonded interactions. VDW potential is approximated by the Lennard-Jones potential function (Eq. (2.6)).

$$U_{ij}^{LJ} = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) \tag{2.6}$$

The electrostatic potential is a much more complicated problem to tackle. VDW and bonded interactions have a much shorter range when compared to electrostatic potentials.

Therefore, a much larger number of particles (close to infinity) have to be simulated in order for the simulation to be realistic. This is, of course, impossible to do. This issue also affects other non-bonded interactions for the same reason of wanting to simulate systems comparable in size to experimental systems with many moles of particles in a reaction vessel.

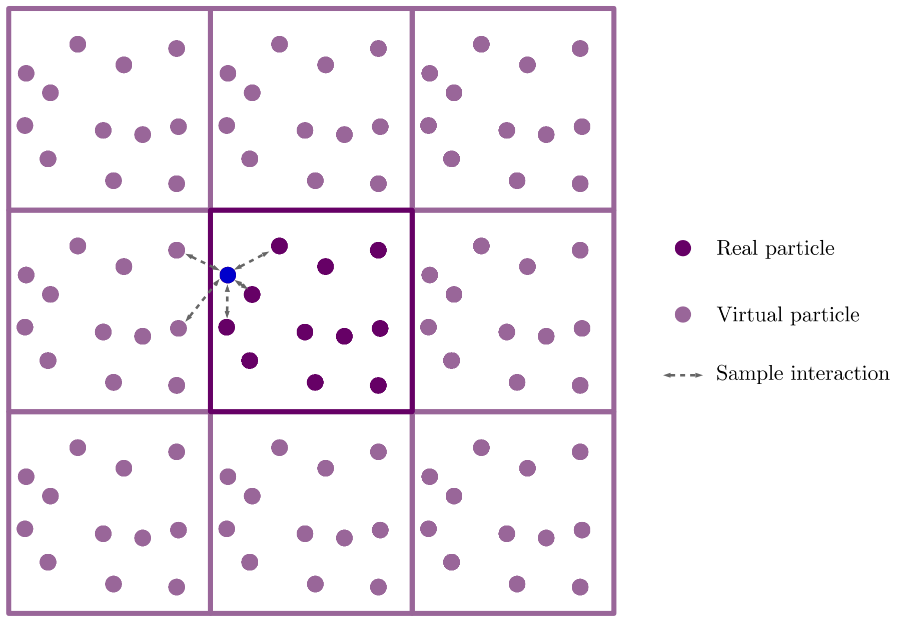


Figure 2.7: Sketch of periodic boundary conditions. A much larger system is created by periodic images of the simulated system. The virtual particles interact with the particles near the boundary of the simulation box. As particles exit the simulation box, a copy replaces it on the opposite side. Particles can interact with other particles and virtual particles depending on the distance between them.

These problem of system size is alleviated with Periodic Boundary Conditions (PBC), see figure 2.7. PBC make the simulation of smaller microscopic systems resemble larger macroscopic systems more closely. PBC introduce a periodicity into the system that is only valid for crystalline systems. This artifact introduced by PBC is removed by the minimum image convention[87].

With this setup, the electrostatic potential energy function can be formulated in a tractable way. Due to their long range, electrostatics have to be modelled using more complex methods. To consider all of space surrounding a particle, Ewald sums are used where the electrostatic energy is calculated in Fourier space[88–90]. There are three popular algorithms that implement Ewald sums: Particle Mesh Ewald (PME)[88], Smooth Particle Mesh Ewald (SPME)[91], and Particle Particle Particle Mesh (P^3M) [92]. Eq. (2.7) defines the total electrostatic energy in a system of N Coulomb point charges.

$$E = \frac{1}{2}\sum_{i,j=1}^{N}\sum_{n\epsilon Z^3}\frac{q_i q_j}{|\mathbf{r}_{ij}+\mathbf{n}L|'} \tag{2.7}$$

where r_{ij} is the vector between particles i and j, $L = diag(l_x, l_y, l_z)$ is a diagonal matrix with sidelengths lx, l_y,l_z, $\mathbf{n}$ indexes the surrounding periodic cells, and the $'$ indicates that the calculation is ignored for $i = j$ and $\mathbf{n} = \{\mathbf{0}, \mathbf{0}, \mathbf{0}\}$.

Equation (2.7) decays very gently over r, so a direct calculation is not feasible. The PME, SPME, and P^3M are some efforts to solve this problem. Readers are referred to [90, 93, 94] for more details about electrostatic force-field calculations. Due to the complexity of these potential functions, the simulations may require a large amount of computer resources—even more is needed for simulating macromolecules. This problem is solved by coarse-graining [95].

The simulations are often times run with the molecules being described at the atomic level. This can be prohibitively expensive. Coarse-graining is built upon the assumption that the internal dynamics interatomic interactions are of less importance than intermolecular interactions of large macromolecules [95]. Groups of atoms are grouped into beads that then interact with each other. There are a large number of coarse-graining methods[96–100] that have been implemented in the aforementioned software packages.

Now that the interparticle interactions are defined, the next step is to calculate and update the particle positions and velocities. As part of the initial conditions, the particle positions are chosen randomly and given random distributions. The positions are taken form a uniform distribution and velocities are chosen from the Maxwell-Boltzmann[84] distribution, shown in Eq. (2.8), to reach equillibrium quickly,

$$p(v_i) = \sqrt{\frac{m_i}{2\pi k_B T}}\exp(-\frac{m_i v_i^2}{2k_B T}). \tag{2.8}$$

Equation (2.1) can be rewritten into a differential form,

$$m\frac{d^2\mathbf{x_i}}{dt^2} = -\nabla U(\mathbf{x_i}) \tag{2.9}$$

which will have to be solved numerically. The velocity Verlet [101] algorithm is used thanks to its symplectic nature symmetry under time reversal [101, 102]. Equations (2.11) and (2.10) define the math behind velocity Verlet,

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \frac{\mathbf{F}_i(t)}{m_i}\cdot \Delta t^2 + O(\Delta t^4) \tag{2.10}$$

$$\mathbf{v}_i(t) = \frac{\mathbf{r}_i(t+\Delta t) - \mathbf{r}_i(t-\Delta t)}{2\Delta t} + O(\Delta t^2). \tag{2.11}$$

Notice that in all of the equations described above, it seems that they can be calculated simultaneously for each particle. The calculation of, say, the force on a certain particle does not depend on the forces on the other particles in the system. This can be taken advantage of any the calculation may be parrallelized to be carried out simultaneously on multiple cores and CPUs. Due to network latency, unfortunately, there are a diminishing returns associated with increasing the number of CPUs. The number of CPUs used has to be balanced with the overall performance (measured in steps/day or ns/day) of the simulation. The latency problem can be avoided by using GPUs. GPUs can run many threads simultaneously, and deal with intercommunication without high latency networking. Many of the MD software packages mentioned before already support acceleration on GPUs.

Finally, note that the MD done for this thesis is slightly different in that it does not model the interaction between atoms and molecules. Essentially, only the aspect of integration used in MD will be used for modelling cell dynamics. This is done by defining custom, and much simpler, force fields that operate on mass points that are much more massive than single molecules. This is described in detail in Chapter 5.

Chapter 3

Some Cell Modelling Techniques

As discussed in previous chapters, cell behaviour is extremely complex and depends on many factors. The behaviour is interesting because its responses to chemical and physical stimuli is what leads to development of complex multi-cellular living organisms. Much has been learned through the traditional experimental nature of cell biology. Much progress has been done since Flemming first described mitosis in the late 19th century [34].

Cell biologists have learned a great deal of regarding the operation of the internal components of the cell. Such as the role of mitochondria in energy production and cellular metabolism, the photosynthesis function of chrolophyl in plant cell, the conduction of electrical impulses through axons in nerve cells, etc [16, 18]. The vital behaviour of groups of cells is also a vibrant topic of inquiry [103–105]. Not only for scientific curiosity, but also to more practical ends such as understanding the mechanisms of disease or decay due to old age. Naturally, there is a need for multidisciplinary research activities to full understand such phenomena.

Any complex phenomenon that is difficult to describe analytically, i.e. anything that is even a little more realistic than spherical cows in a frictionless vacuum, is an attractive topic for the development of computational models that allow us to run simulations that can accurately simulate that behaviour. The models that arise from such analysis can then be used as either a basis for experimental design, as an augmentation to already available experimental evidence, or even as a cheap sandbox to create strange environments and stimuli that are difficult to create in the lab. Thus, many computational techniques have been developed to study cells. This chapter attempts to summarize some of these techniques.

When considering cell behaviour, it is possible to have multiple types of models that can be used. The field of mathematical biology encompasses a number of so–called continuum models which are ones in which the behaviour of cells is modelled using mathematical equations that link certain behaviours to parameters related to the cell type or environment. This method produces a set of coupled differential equations that can be solved and their solutions compared with experimental data. The problem with these techniques is they sacrifice the vast majority of the properties of the cells themselves in order to elucidate some macroscopic "emergent" property. This is not always possible as inter-cellular interactions cannot always be approximated fully [106].

The other type of models are those that consist of cells that are treated as individual cells. In most cases, the cell is modelled as a closed shape that interacts with its surroundings in a particular way. The nature of this interaction, of course, depends on the model itself. The advantage with these methods is that the individual behaviour of the cells themselves can be studied and observed computationally. Naturally, the cost of computations is significantly higher with these type of models. Despite the increased granularity, or resolution, of these types of models, many approximations have to be made regarding cell structure itself. Additionally, these models will abstract away all of the minute intermolecular interactions that intercellular interactions are based on. It is also possible to create hybrid models that are based on both of these principles with discrete elements being affected by continuous functionals [107].

The model that is presented in this thesis is based on previous work by Karttunen and Åström, who designed the basic principles in 2006 [14], and the work of Mkrtchyan et al. [15, 49] that was used to study epithelial packing in 2014. The 2D model is described in Section 3.3

3.1 Mathematical Biology

The first kind of models that can be used to study cell behaviour are those that will be referred to as either mathematical models, these are the kind that simply reduce cell behaviour into mathematical formulations that apply to on specific aspect of cell behaviour. This techniques are placed under the umbrella of Mathematical Biology and can be thought of as continuum models.

In this class of techniques, differential equations are derived from principles observed experimentally or from first principles —biological first principles. Unfortunately, there is a difficulty in defining principles such as the rules of equilibrium or conservation that govern cell behaviour[108]. This is due to the non-linear nature of the interactions between cells, i.e. the outcome of interactions of a cell with its neighbours is not always a linear sum of each individual interaction with each individual neighbour [109].

Therefore, it is rarely biologically sound to try to show some macroscopic properties from first principles [106]. Nonetheless, these methods can be very useful to study systems that are at least temporarily in the valid domain of applicability of a specific model.

In general, models of this type have been developed to study many various cellular phenomena [108, 110, 111] such as: cell-cycle control, cell death, cell differentiation, cell aging and renewal, and the same for cancer cells. Or sub-cellular phenomena such as: DNA control (Transcription, Replication, Repair), or endocytosis[112]. Readers interested in other aspects of this field are referred to Refs. [106, 108, 110, 113–115].

3.1.1 Models of Cell Population Dynamics

As an example of mathematical biology, consider the study of population dynamics. The progression of cell growth over time has always been an important area of study. Firstly, measuring the population of a culture and observing its trend are fairly simple procedures, compared to other more complex experiments, so it can be done relatively easily. The trends in the number of cells in a culture over time can easily be observed, and be plotted. This type of graph will be called the population curve or population trend line hereafter. The cells of different organisms grow at different rates and with different kinds of trends. Anomalies in population trends can be linked to illness [116]. The population curve can also differ greatly within the same species depending in what stage of life that organism is in (development, reproduction, death, etc.). Growth curves with remarkably similar properties can be found in the progression of many quantities such as the height of a human, or population of most organisms. These growth curves tend to be sigmoidal and can even describe quantities such as dose-mortality relations [117].

While making some simple assumptions, one can very easily derive the population curve of an ideal system where nothing limits cell growth at all. At one instance in time, some fraction r of cells will divide, so the rate of change of the number of cells N will be given by (3.1). Which can trivially be solved to give (3.2) where N_o is the initial number of cells.

$$\frac{dN}{dt} = rN \tag{3.1}$$

$$N(t) = N_o e^{rt} \tag{3.2}$$

The function in (3.2) is called an intrinsic growth trend, it describes the population growth under ideal conditions where each individual has complete access to nutrients and does not perish. Unfortunately, or fortunately depending on perspective, this is not the case. The conditions for growth are far from ideal, one can even say that it the world is hostile of growth. There are many factors that limit growth rate such as competition with other cells for nutrients, the limited lifetime of cells with an associated death rate, the threat of disease and/or predators, etc. A more complex trend is normally observed. This is where we shift to looking at sigmoidal functions.There have been number of functions [116–118] that are known to follow this behaviour:

- Intrinsic Growth [119]
$$N = N_o e^{kt}, \tag{3.3}$$
- The logistic function [117]
$$N = \frac{N_{max}}{1 + be^{-kt}}, \tag{3.4}$$
- The Gompertz function [120]
$$N = N_{max} e^{-be^{-kt}}. \tag{3.5}$$

These models are not only valid to study the cell number over time, but can also be applied to other observables such as average size or weight[116, 118]. Typically in such systems, growth starts at population of zero (or some small value close to zero) at $t = 0$, accelerates to a maximum (μ_m) after a lag time (λ), finally the growth rate drops to zero again asymptotically at a maximum population A [117].

In 1981, Schnute generalized all of the growth models shown above, including some other more complex ones, into special cases of a universal growth model[116]. Zwietering et al. [117] did a study that compared various special cases of the Schnute model [116] and showed that they can produce good fits to the population trends of various bacteria, where the parameters μ_m, λ, and A were found with nonlinear fitting.

3.1.2 Continuum Models of Cell Behaviour

Local interaction functions are used to develop these type of models[51, 121–123]. From a mathematical perspective, these models are not much different from the models in mathematical biology. The microscopic interactions of cells are abstracted into functional forms which may describe parameters such as density, growth rate, death rate, inter-cellular interaction strength [122], or interaction strength with the medium [109]. Unfortunately, this kind of modelling cannot take into account of all the minute interactions between cell membranes, and therefore are not always ideal for simulating cell behaviour.

Somewhat like in many-particle physics, all of the individual cells are modelled with some continuous parameter like density. Then, spatiotemporal equations of motions are derived that describe the dynamics of a continuum of cells [115, 124]. This method yields the dynamics of the total population as whole and individual behaviour is neglected. Continuum models have been used to study a variety of phenomena such as tissue deformability [51], tumour growth [121], viscosity of tissues [122], and anisotropic tissue growth [123]. While the results of computational studies have proven favourable, their limitations remain. Continuum models cannot take the mechanical interactions between cells at the cellular scale of size. Methods that can model individual cells are needed to fully understand cell behaviour.

3.2 Discrete Cell Models

Now we take a look at some models that implement cells as individual entities that interact with their surroundings, much like real cells. These models can be more computationally expensive, however they allow us to study different aspects of cell behaviour at a lower level without forgoing any of the small interactions between cells. It is possible to have models that simulate cells as either two dimensional or three dimensional objects. Two dimensional models are cheaper computationally though they do not possess the ability to model the full breadth of cell behaviour directly as they are missing the third dimension. Three dimensional models can capture cell behaviour more fully, but are computationally very expensive. More generally, cell models that simulate single cells can also be named agent-based methods.

3.2.1 Delaunay Object Dynamics

Delaunay Object Dynamics (DOD) is a three dimensional technique where each cell is modelled as a three dimensional, elastic, and adhesive Voronoi cells [64, 124–126]. A cell is described as a three dimensional polygon that is constructed with Delaunay triangulation. Readers interested in the details of Delaunay triangulation are referred to [125, 127, 128]. Each face and edge of each cell is then modelled with damped Newtonian Mechanics [125].

The Delaunay triangulation used in this method is slightly varied form regular Delaunay triangulation and is termed weighted Delaunay triangulation [64, 129]. To put it simply, Delaunay triangulation is method of triangulation for a set of points that obeys the Delaunay Condition [125, 130]. The Delaunay condition is that the circle circumscribing any triangulation must not contain any other points. Figure 3.1 shows a Delaunay triangulation of a random set of points.

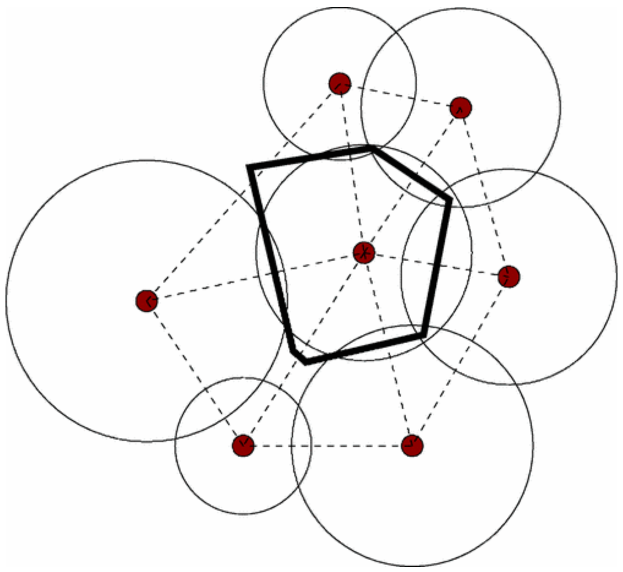


Figure 3.1: Here the Delaunay triangulation of two dimensional cells is shown. The dashed edges belong to Delaunay simplices, the solid line is the Voronoi region corresponding to the middle cell. The overlapping circles represent the weighting of the triangulation. Reprinted with permission [124] © 2005 American Physical Society.

The Delaunay triangulation describes the topology of the surroundings of the cell and how it is positioned in space with respect to its neighbours. This way, the triangulation describes the distances between cells very well. The dual graph of Delaunay triangulation is the Voronoi tessellation of the same set of points [131]. The Voronoi regions, shown in Figure 3.1 describe the shape, contact surfaces, and sizes of the cells in the system [64]. Cell division, death, or flux changes the triangulation by adding, removing, or changing the positions of points in the system. As the points change the Delaunay and Voronoi graphs change accordingly.

The forces between cells are caused by interactions occurring on the surfaces of the Voronoi cells are modelled along the Delaunay triangulation simplices [64, 124, 126]. The DOD force-field contains terms for active forces that are generated by the exertion of cytoskeleton on the cells' surroundings, and passive forces that are a result of the cells' interactions with their neighbours. The passive forces model cell elasticity and adhesion. Lastly, the forces are damped by a drag forces on all cells that embody the interaction of the medium with the cells.

DOD has been used to study the proliferation, death, and behaviour of lymphoid cells [64], tumour cell reaction to changes in nutrient levels [124]. DOD is also generally applicable to tissue organization [64], and cell migration [132].

3.2.2 The Cellular Potts Model

Physicists have been for a very long time modelling and analyzing problems which exist at many different scales of space and time, so called multi-scale problems. The mechanisms that control tissue organization depend on the local intercellular interactions between each cell and its neighbours. The Cellular Potts Model (CPM) is a lattice based model that that can be used to understand the factors at the cellular scale that affect tissue organization [60, 133].

For many decades, physicists have been developing models that apply to very difficult multi-scale problems. It is often possible to find some already existing model used for some unrelated science in the wild that can be applied to a new problem at hand. The CPM is one such model that was developed by Glazier and Graner[134, 135]. The model originally developed in the field of solid-state physics to study ferromagnetism. This model is also called the Glazier-Graner-Hogeweg[60] model.

The CPM is based on the Potts model that was developed by R.B Potts under the tutelage of C. Domb in the early 1950s[136]. Potts proposed this model as a part of his PhD thesis. It is a generalization of the Ising model[137] where instead of considering two possible states, such as for example atomic spin states -1 and +1, the Potts model considers any number of states. See the review by Wu[138] for a more detailed description of the Potts model.

Much like the Potts model, the CPM is a lattice model at heart. That means that each cell no longer is its own entity, but collection of grid points, which may be pixels or voxels. These collections grid points are then treated individually. Graner and Glazier first studied cell sorting using this method [135], but it is possible to study other phenomena such as cell migration[62], and morphogenesis[139]. The lattice grid may be cubic or hexagonal. The basic idea in this type of modelling is too minimize the energy under certain imposed fluctuations. The cell is modelled as a more-or-less deformable object and its shape is affected by both internal and external stimuli. The parameters of the model can be mapped to the physical and biological properties of real cells.

Figure 3.2 shows an example of a two dimensional square lattice that was used by Voss-Böhme [60] in their analysis of CPMs. The system is divided into a grid lattice, which can be in 2D or 3D, and each lattice cite is assigned an index depending on what type of site it is. The 0 index is typically reserved for the medium, and the remaining sites are enumerated to lie in their own cells. This defines cells as internally structure-less boundaries that can have complex shapes. The cell interactions are described as effective energies and elastic constraints. Time is propagated as changes in the configuration by minimizing these interaction energies. The configurations are updated by with a modified Metropolis Monte-Carlo algorithm [62, 140, 141] in which sites are updated randomly and accepts changes with a Metropolis-Boltzmann probability [62, 136].

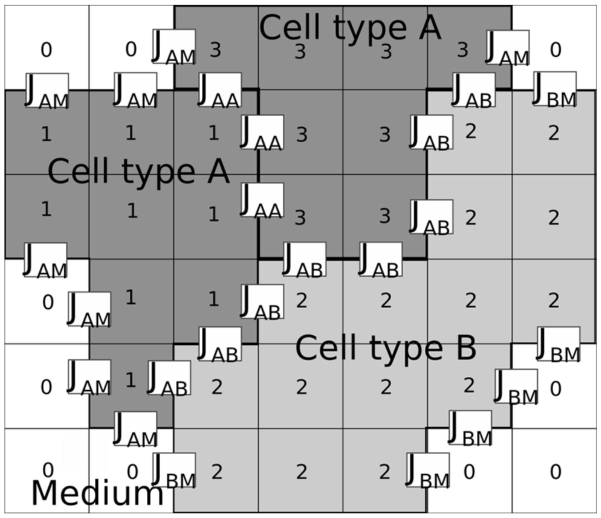


Figure 3.2: The CPM is a lattice model where each cell is considered to be a collection of grid points. The cells have indices 1,2,3 and the medium is with index 0. Domains of the same index greater than 0 define the shape of each cell. There are three interaction terms: J_{AM} which is for the interaction between cell type A and the medium, J_{AB} which is for the interaction between the A cells and B cells, J_{AA} between the cells of the A type, and J_{BM} for the interaction between B and the medium. J_{BB} is also valid, however is not needed in this figure. Taken from [60], CC license.

Let the state of the system be denoted by $\sigma(x, y)$ where (x, y) is the location being considered. J is a single parameter than accounts for adhesion and cortical tension which is measured in cost of energy per unit of membrane length between two sites of different type. The volume of the cell (or area in 2D) is constrained to a reference of v_o with compressibility κ^{-1}. ΔE is the energy difference between the two states. Equation (3.6) shows the probability distribution for accepting or rejecting updates,

$$P(\Delta E) = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0. \end{cases} \tag{3.6}$$

Where T represents the average fluctuation in the boundary of each cell. Normally, one Monte-Carlo step (MCS) is an attempt to update each lattice site. Equations (3.7), (3.8), (3.9), (3.10) show how the energy of σ may be defined. E_{chem} is the contribution to the energy coming from motile forces that point along the direction of cell polarity n_i with field strength μ_i.

The energy of any given pixel of a cell is given by

$$E = E_{adh} + E_{vol} + E_{chem}, \tag{3.7}$$

where

$$E_{adh} = \sum_{k,l} J_{kl} \left(1 - \delta_{k,l}\right) \tag{3.8}$$

is an approximation of the intercellular interaction between cells, J is related to membrane tension and differs depending on what two cells are being considered (J_{AB}, J_{AM}, J_{AA} in Figure 3.2), k, l index the neighbouring pixels. $\delta_{k,l}$ is 1 if the pixels k, l belong to the same cell, and 0 otherwise.

$$E_{vol} = \sum_{i} \frac{1}{2}\kappa(v_i - v_o)^2 \tag{3.9}$$

is the energy needed to deform a cell, where v_0 is the cell volume (area in 2D) at equilibrium, κ^{-1} is the cells compressibility, and v_i is the instantaneous cell volume.

$$E_{chem} = \sum_{i} -\mu_i \mathbf{n}_i \cdot \mathbf{r}_i, \tag{3.10}$$

is the energy resulting from motile forces along polarization vector $\mathbf{n}_i$, with magnitude μ_i. $\mathbf{r}_i$ is the location of the cell's centre of mass.

CPMs have been used to study a wide variety of cellular phenomena, especially cell sorting [134, 135, 142, 143], cell migration [62, 144], and chemotaxis [145, 146](cell motion in reaction to chemical gradient).

However, it is clear that the interactions between cells in CPM models do not take into account any mechanical interactions between cells. While the resultant motion of cells due to energetically unfavourable conditions can be correct, the mechanical interactions of cells play a vital role in their behaviour [54, 56].

3.2.3 Topological Models

This class of models are based on the work of Matella and Fletterick that they did in the 1980s[147–149]. These models are inherently two dimensional and rely on graphs of adjacent cells, much like DOD (see Section 3.2.1). In topological models, the cells are triangulated such that they always meet at corners with two neighbouring cells [149] at trivalent junctions. The resultant arrangement is intentional because it was observed that cells in planar systems favoured hexagonal packing[15, 49, 56, 61, 150–152]. The dual map of the trivalent junction describes the adjacency of cells.

The dynamics of the cells are simulated by manipulating the map structure to introduce cell division, growth, movement, adhesion, differentiation and death [149]. Cell division is simulated by introducing new edges that divide a parent cell into fairly symmetric daughters. Cell growth is simulated by systematically increasing the size of cells. Cell movement is done by exchanges in adjacent cells. Differentiation is done by subtly changing parameters of some of the cells so that their behaviour is slightly different. And finally cell death is simulated by removing some edges of the dying cell until it finally disappears.

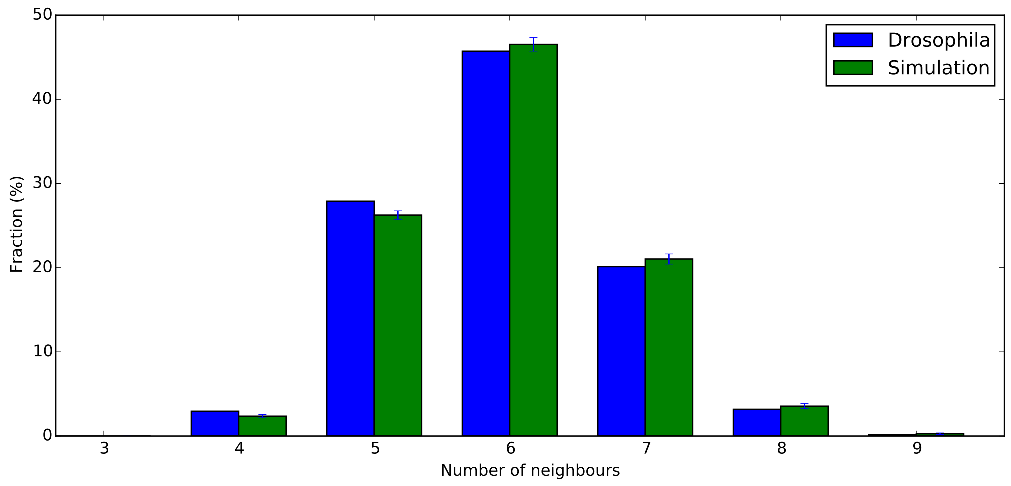


Figure 3.3: This graph shows the packing distribution of a topological simulation done by Patel et al., data taken from [61]. The results are of orthogonal equal split division.

Once again, the limitation of this technique is that topological models cannot account for cellular geometry, the mechanical interactions between adjacent cell membranes, and the interactions of cells with the medium. Topological model simulations are known to be good approaches to study two dimensional structures such as epithelia [61, 151, 153, 154].

However due to the assumptions made about the packing of cells, these models cannot be applied to simulate three dimensional tissue, or tissue that is not necessarily hexagonal.

Consider the results shown in Figure 3.4 and Figure 3.3. The topological simulation that was run by Patel et al. [61] produced the correct packing distribution of epithelial cells. But in Figure 3.4, we clearly see that the mechanics at the cellular level are not necessarily correct.

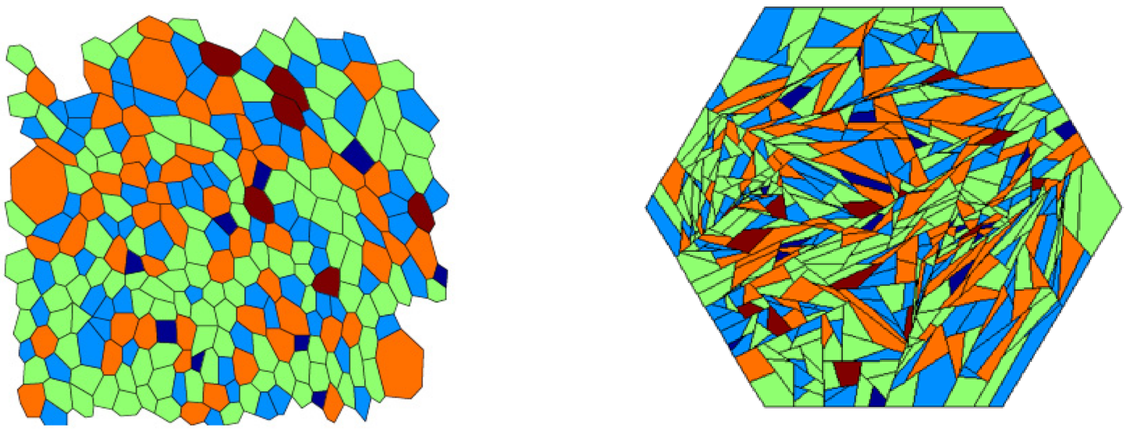


Figure 3.4: Left: The approximate polygonal topology of the epithelium the *Drosophila melanogaster* (fruit fly) wing disc as measured by Patel et al. [61]. Right: The topological model that was used to simulate the wing disc, simulated by Patel et al. [61]. Dark blue cells have four neighbours, blue have five, green have six, orange have seven, and maroon have eight. Even though Patel et al. ended up with the correct distribution of cell grouping, the model cells themselves do not arrange themselves like real cells. © 2009 Patel et al., CC license.

3.2.4 Vertex Models

Vertex models are a made of sub-cellular particles that are bound in tightly bound clouds, developed by Newman[63]. Each cell is represented by a cloud of mass mounts that are held together with strong intracellular Morse potentials [63, 155], weak Morse potentials are used for inter-cellular interactions (Eq. (3.11)),

$$V(r) = V_o \exp(-\frac{r^2}{\zeta_1^2}) - U_o \exp(-\frac{r^2}{\zeta_2^2}), \tag{3.11}$$

where V_o, U_o, ζ_1, ζ_2 determine the strength of adhesion and repulsion. This model can be used to study cell reproduction in two and three dimensions.

3.3 Two Dimensional Cell Dynamics

The model that is described in this work is a three dimensional version of one that was first designed by Karttunen and Åström in 2006 [14]. Later, in 2014, Mkrtchyan, Åström, and Karttunen expanded upon the model by adding modes of cell growth and division[15, 49]. Since the model is two dimensional, epithelial cell packing seemed like an obvious target for model validation. They saw that their model could accurately reproduce the packing of the *Drosophila* wing disc[15, 49].

To summarize, the new model is a single-cell based mechanical model with which accounts for cell cortex contractility, and cell-cell adhesion[15, 49]. Each cell is a closed loop of mass points, and the mass points interact with each other in a force field that accounts for bonding interactions between neighbouring mass points in the same cell, adhesive interactions with neighbouring cells, and intercellular friction. Each cell is assigned an internal pressure which controls the cells' growth. Mitotic cells are known to grow by internal pressure [35, 37], which makes the driving force behind growth biologically sound. Furthermore, the physical properties of tissue, and the effects of division on the same, can be controlled at the cellular level. The system allows for spontaneous cell rearrangements and movement without the need for stochastic laws.

3.3.1 Intracellular forces

A cell is represented by a loop of springs connected at mass points. This structure was originally suggested by Åström and Karttunen in their work on cell aggregation in confined spaces[14]. Tension forces operate on each mass point through spring interactions with neighbouring mass points. Figure 3.5 shows the tension and pressure forces operating on each mass point.

Spring forces of adjacent mass points balance the internal pressure force. With this, the forces acting on a mass point can be denoted as

$$\mathbf{F}_i^{cell} = \boldsymbol{\sigma}_i \boldsymbol{\eta}_i - \boldsymbol{\sigma}_{i+1} \boldsymbol{\eta}_{i+1} + \frac{Pl}{2} \left(\boldsymbol{\nu}_i + \boldsymbol{\nu}_{i+1} \right), \tag{3.12}$$

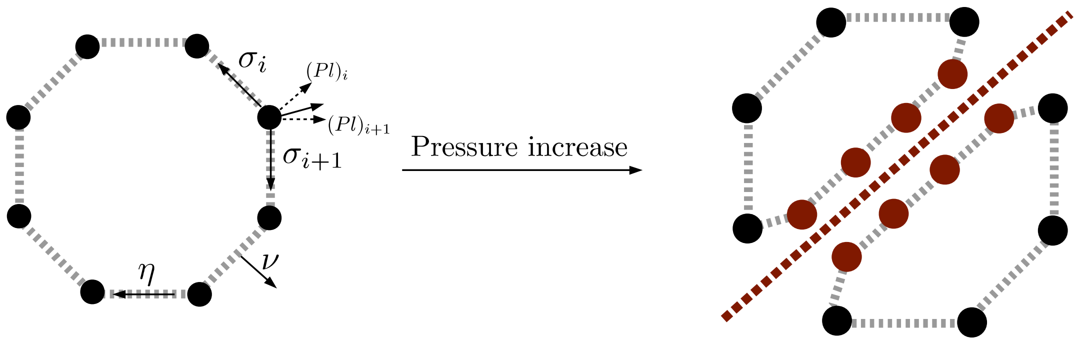


Figure 3.5: **The two dimensional model that was introduced by Mkrtchyan et al. [15, 49]. Cells grow by gradual increase in their internal pressure. When a cell is divided, new mass points are added along the division line. The division line can be symmetric random, orthogonal, or asymmetric.**

where η, ν are the tangential and normal vectors respectively, and σ is the tension force defined as $\sigma_i = K^{spr}(l - l_o)$, l and l_o being the equilibrium and instantaneous spring lengths respectively. All of the springs are given the same spring constant $K_i^{spr} \equiv K^{spr}$, but this condition can be lifted.

3.3.2 Intercellular Forces

The force-field also defines inter-cellular forces in a two dimensional tissue. Mass points of different cells experience repulsion (Eq. (3.13)), adhesion (Eq. (3.14)), and intercellular friction (Eq. (3.15)). For all of the following equations, i indexes the mass point being considered and j indexes mass points *belonging to other cells*,

$$\mathbf{F}_{ij}^{rep} = \begin{cases} -K^{rep}(R_c^{rep} - R_{ij})\hat{\mathbf{R}}_{ij} & \text{if } R_{ij} < R_c^{rep} \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

The mass points repulse if they are within R_c^{rep} of each other. The repulsion force should balance the pressure force that pushes all mass points outwards towards the mass points of other cells, so the repulsion spring is set as $K^{rep} \approx K^{spr} Pl$.

Intercellular adhesion maintains tissue integrity. Real cells adhere to each other through adhesive molecules[156, 157], this behaviour is emulated by this model by including attractive intercellular forces. Each mass point acts as a site for adhesive interaction. All sites are assumed to have the same adhesion strength $K_{ij}^{adh} = K^{adh}$. The adhesion force is defined in Equation (3.14) as

$$\mathbf{F}_{ij}^{adh} = \begin{cases} K_{ij}^{adh}\left(R_c^{adh} - R_{ij}\right)\hat{\mathbf{R}}_{ij} & \text{if } R_{ij} < R_c^{adh} \\ 0 & \text{otherwise,} \end{cases} \tag{3.14}$$

so that mass points within the attraction range, R_c^{adh}, attract each other with strength K^{adh}.

During tissue formation cells can experience local rearrangements or large scale migrations. These actions depend the cells impend the motion of each other, therefore, the cell movement can be controlled be controlling the level of viscous dampening between cells. If i and j are two cells moving past each other, then the friction between them depends on their relative velocity $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. $\mathbf{F}^{fric}$ is then calculated with the tangential component of $\mathbf{v}_{ij}$, $\mathbf{v}_{ij}^{\tau}$. The effects of a viscous medium is also added with a dampening coefficient c that acts uniformly on all mass points,

$$\mathbf{F}_{ij}^{fric} = -\gamma_i \mathbf{v}_{ij}^{\tau}. \tag{3.15}$$

With all of its components defined, Equation (3.16) shows the full force acting on any particle. This force is then used to simulate the particles in an MD simulation:

$$m\ddot{\mathbf{r}} = \mathbf{F}_i^{cell} + \sum_j \mathbf{F}_{ij}^{rep} + \sum_j \mathbf{F}_{ij}^{adh} + \sum_j \mathbf{F}_{ij}^{fric} - c\mathbf{v}_i \tag{3.16}$$

In isolation, many cells prefer a roughly spherical shape [35]. Though in tissues where cells interact strongly with neighbours, cells can take a more polygonal shape [151]. In two dimensional tissue such as epithelia, experiments have shown that cells pack together with a particular topology. Lewis[152] showed that epithelial cells pack as mostly hexagonal cells, with lower fractions of pentagonal and heptagonal cells. Gibson et al. [151] later showed that this distribution of cell packing is conserved among different species, which suggests a common mechanism to the emergence of this packing. It is important for any models of 2D tissue to also contain similar distribution of cell polygon types. Comparison between the polygon distributions of experiment and simulation are shown in Figure 3.6.

The model introduced by Mkrtchyan et al. [15] is also capable of modelling the effects of different kinds of cell division on cellular proliferation.

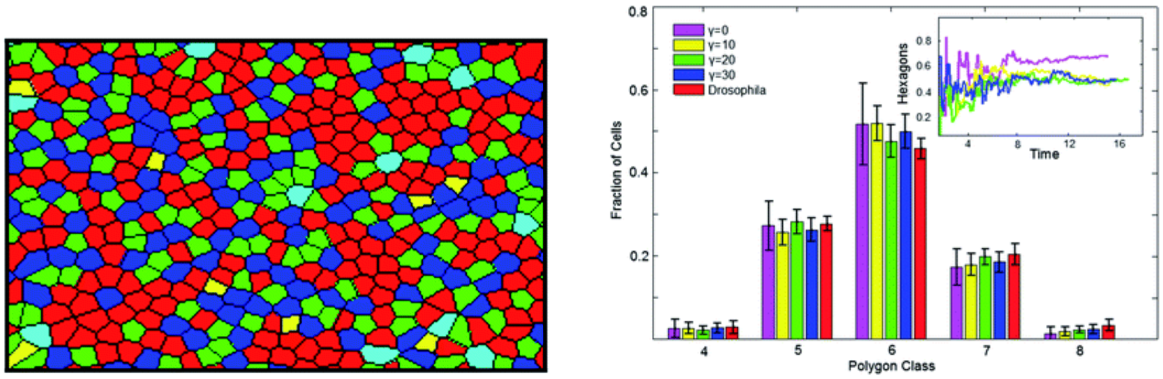


Figure 3.6: Left: The packing of epithelial cells at the end of the simulation. Pentagons are green, hexagons are red, and heptagons are blue. Right: Bar chart showing the packing distribution compared to *Drosophila* packing at different γs. Right Inset: The percentage of hexagons over time. Adapted form Mkrtchyan et al. [15, 49] with permission of The Royal Society of Chemistry.

The model that is developed in this thesis is similar to the 2D model introduced by Mkrtchyan et al. earlier[15]. The 3D model is described in detail in Chapter 5. The principles remain the same, except they are applied to a 2D mesh of mass points on the surface of a sphere instead of on a loop of mass points.

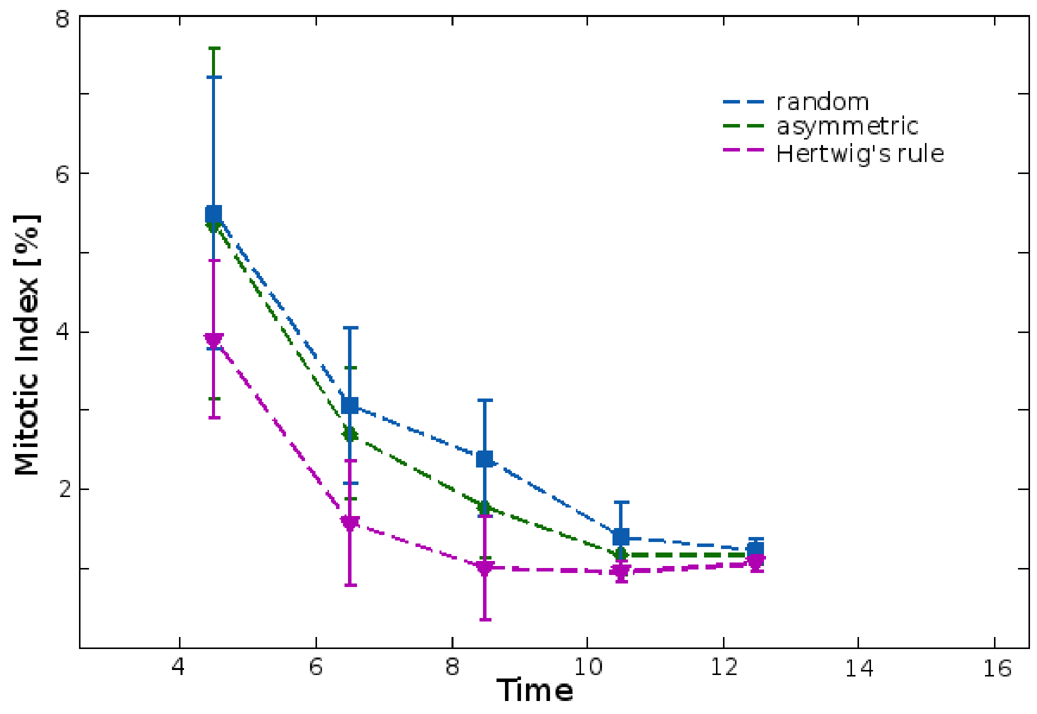


Figure 3.7: Mitotic index development over time for 2D simulated cells. Asymmetric and symmetric division show similar Mitotic Index (MI) time evolution. Hertwig's rule division [45] decays faster. In all three cases, the mitotic index trend is comparable to experimental trends[158], see Section 6.1. Figure from [159], manuscript to be submitted.

Chapter 4

Programming on GPUs

So far, we have seen why we need to use simulations to model the behaviour of phenomena that are too complex to study theoretically, yet still important and interesting. Experimentally, it is not always possible to study all the details of biological systems. This could be due to a shortage of funds, inadequate access to instruments, unavailability of appropriate personnel, time constraints, time and length scales that are difficult to study experimentally, or unavailability of biological samples. There is always a great demand for simulation software that can simulate biological systems, both existing and theoretical. And so, many simulation techniques have been developed that can be used to model cell behaviour, which is the focus of this work.

Some software packages exist already. And where they do not, the algorithms that are used in them can implemented. However, most of these packages are designed to run on CPUs. For example, most Molecular Dynamics (MD) code was initially developed to run on CPUs with the possibility for using multi-threading or running on many hundreds of Central Processing Units (CPUs). It is certainly possible to run the simulation code on many hundreds of CPUs, but it is not always favourable. In fact, when compared to running Graphics Processing Unit (GPU) code in certain cases, it may become difficult to advocate CPU code over GPU code. Given the inadequate computational power of the average personal workstation, users are forced to get access to supercomputers which have their own set of problems. GPUs can be used to replace CPUs in some cases. This chapter introduces the advantages of running simulations on GPUs and summarizes their use and operation.

4.1 GPU versus CPU

CPUs are created with a great mount of complex control circuitry. This is to maximize performance and flexibility. However, this dramatically higher mount of complexity leads to high power consumption and heat dissipation. The Thermal Design Power (TDP) is generally higher with higher clock speeds. Higher performance has always been the driving force behind processor hardware design. Over the past few decades, processor clock speeds increased steadily and came to a stop a few years ago as can be seen in figure 4.1. The saturation in clock speeds can be attributed to the fact that even though performance is the ultimate goal, it is constrained by power consumption and heat dissipation. This constraint has become more stringent in the past few years because the heat produced per transistor has not reduced by much, whereas the number of transistors per chip have been increasing exponentially. The heat produced by chips with high clock speeds has become unmanageable [160, 161]. Furthermore, the thickness of the oxide layer in most Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) designs is now at an irreducible minimum, so that even if the heat problem is solved, transistor switching speed still cannot be reduced by much [160, 161]. The heat dissipation problem, transistor switching speed problem, and the demand for high processing capacity are reconciled by multicore chips where there are more cores with lower clock speeds [160, 161].

Some problems with relying on CPUs for High Performance Computing (HPC) are outlined below. Many problems are a consequence of being forced to use compute super-computers with thousands CPUs.

Problems related to CPUs:

1. **Higher run times**
   As will be shown further below, CPUs are plainly slower than GPUs for running MD simulations. The force calculating step in MD is such that the forces on any one particle are independent of the forces on the other particles in a system. Therefore, it is possible to calculate the forces on multiple particles simultaneously. This fact is taken advantage of fully on single GPUs which are capable of performing many more such calculations simultaneously. Many ($10 - 10^3$) of CPUs have to be used simultaneously for the same effect.

   However, there is one important caveat here, the speed of the CPUs is limited by the interconnect between the different CPU nodes. Interconnects always have a delay on the order of 1–10 ms, CPUs operate in the ns range. The latency forces CPUs

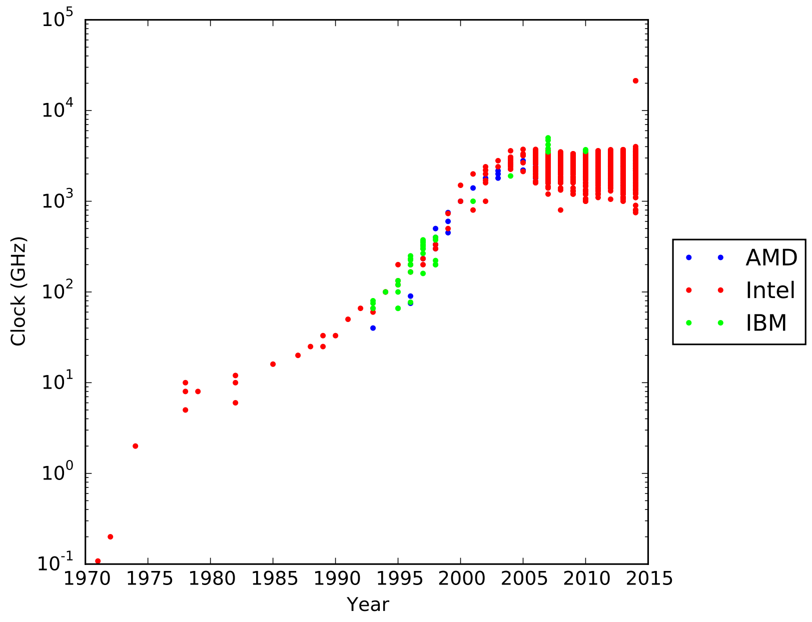


Figure 4.1: CPU Clock speed trend over the past few decades. Apart from a single anomaly, it seems that clock speeds have reached a maximum. Data taken from cpudb.stanford.edu.

to wait relatively long times when many CPUs are communicating data to process. The latency problem places a limit on how many CPUs can be used efficiently on a single problem.

All of these issues add up to long simulation run times. Simulation run times are the limiting factors to how fast a researcher can iterate on the initial conditions of their simulations. If run time is minimized, productivity will be maximized.

2. **Lack of access to supercomputers**
   Computers with many thousands of CPUs are rare. Those that do exist were built for a high cost, and are expensive and difficult to maintain. Obtaining access to such compute servers is not easy and even after getting access, the server has to be shared with other users.

3. **Complexity in programming**

   (a) **Inadequate software**

The software installed on the servers is not always appropriate and users rarely have full control on what software they use. Sometimes,

(b) **Queue time**
The super computers often have long queue times due to the number of users clamouring for compute devices CPUs.

(c) **Down time**
These systems are prone to failures and require periodic regular maintenance due to their complexity.

4. **Power consumption**
GPUs are known to consume less power than CPUs per Giga FLoating point OPerations per Second (GFLOP/S) [161, 162], see Figure 4.2. This means that using GPUs saves considerable power and can be considered less harmful to the environment—greener.

This slow down in clock speeds has been accompanied by an increase in the number of cores per CPU, which has forced developers to parallelize software to run on multiple cores to maximize usage. That, and improving algorithm efficiency, will be the most reliable way to ensure faster run times in the foreseeable future. Running codes on highly parallel GPUs should be the most obvious next step in this direction.

A single GPU can be used to replace large numbers of CPUs thanks to the enormous compute capacity they have. Simulations can be run on a single workstation that costs much less and is easier to maintain. Though, for fairness, these problems are present and just as frustrating when problems are so large as to need many GPUs. With GPUs the need for large compute clusters is shifted to truly enormous problems that would not even be tractable with many CPUs. So by using GPUs we can either reduce costs vastly, or start studying vastly more complex and/or large systems. Which are both highly sought after improvements in any case.

So far, it has been implicated that there are no disadvantages for using GPUs. This is not true. There are many problems with using GPUs [11–13]. The GPU was designed to solve a very specific kind of problem [11–13]. This was the problem of applying complex transformations to images efficiently. When personal computers were first being developed, CPUs were not powerful enough to adequately manipulate images. This necessitated the invention of specialized processor that would only process images. As a result, the GPU has a very particular architecture. Therefore, a GPU can only be used under certain circumstances.

Issues with using GPUs:

1. **Dependence on CPUs**
   GPUs cannot be used as general purpose processors due to the highly specialized nature of GPU hardware design. Consequently any programs that utilize GPUs must be managed by a CPU, see figure 4.7. This may change in the near future with processors that borrow architectural principles from both CPUs, named Heterogeneous Systems Architecture (HSA). However, even with HSA devices, complexities will still exist and glsgpu acceleration techniques will still be needed for them.

2. **Complexity**
   The highly parallelized hardware architecture of GPUs exposes programs to highly esoteric constraints on algorithm design and validation. It is very likely that seemingly simple algorithms would have to be heavily redisigned to maximize performance gains. If code algorithms are not written correctly, then using GPUs may actually yield worse performance than CPUs due to the slower clock frequencies of GPUs.

3. **Interfacing**
   Targeting the GPU for executing simulation code is non-trivial. Until fairly recently, this was not even possible as there were no Application Program Interfaces (APIs) available to do floating point math normally. So all physical problems had to be recast into graphics processing problems which was quite non-intuitive. This problem has been solved somewhat with the introduction of some libraries such as The Compute Unified Device Architecture (CUDA) and Open Compute Language (OpenCL). These libraries have made interfacing with GPUs simpler.

4. **Problem dependence**
   The specialized nature of GPU architecture makes the device only good at solving certain types of problems [11, 13]. Some criteria for choosing problems to accelerate on GPUs are:

   (a) **Large computational requirements**
   The GPU only outperforms the CPU when its hardware is being used to the fullest. Small calculations will generally be more efficiently computed on the CPU

   (b) **Substantial parallelism**
   It follows from issue 4a that the calculation must be parallelizable. Problems with a lot of recursion most probably will not benefit form GPU acceleration (e.g.

computing the Fibonacci sequence or the factorial of a number). Fortunately MD can be parallelized efficiently.

(c) **Latency independence**
The calculation being done is not affected by latency. Because of issues 1 and 3, there is a large overhead in transferring data to and from the GPU. This is really only a problem for graphics applications and is included here for completeness. It does not affect MD code very much as, typically, initial conditions have to be moved to the GPU only once. Data is read from the GPU only to output it. There are also ways of hiding this latency when running calculations, but that is out of the scope of this thesis.

Until a few years ago, direct floating point math was not possible on GPUs. The effort to use GPUs for general computations is called General Purpose GPU (GPGPU) [12] computing. GPGPU has become significantly easier with the advent of new hardware which supports floating point arithmetic (needed for scientific computing), and new software libraries and standards that make interfacing with GPUs more efficient are simple. Software such as OpenCL and CUDA.

With all of the problems with CPUs and GPUs in mind, there still are certain problems particularly well suited to execution on GPUs[11, 13, 65, 163, 164]. It has been shown that MD simulations are definitely well suited to GPU acceleration[11, 65, 165–170], they have large computational requirements, benefit from parallelism, and can be independent of latency. When applied correctly, GPUs will always outperform CPUs by at least 2–3 times, and in some cases by a few orders of magnitude. See figure 4.2 for some comparison of performance information.

Stone et al. made a simple comparison, shown in table 4.1. Some energy evaluation kernels were written in c and in CUDA with varying levels of optimization. C code was compiled with the Gnu Compiler Collection (GCC) and the Intel C Compiler (ICC), ICC is known to yield higher performance binaries, especially on Intel devices. A large portion of the speed up on GPUs can be attributed to the significantly faster memory access speeds on GPUs thanks to the highly optimized hardware design and faster GPU memory clocks. Anderson et al. [65] found that many aspects of MD, including Lennard-Jones force calculation, neighbour list generation, can be sped up significantly on GPUs [65].

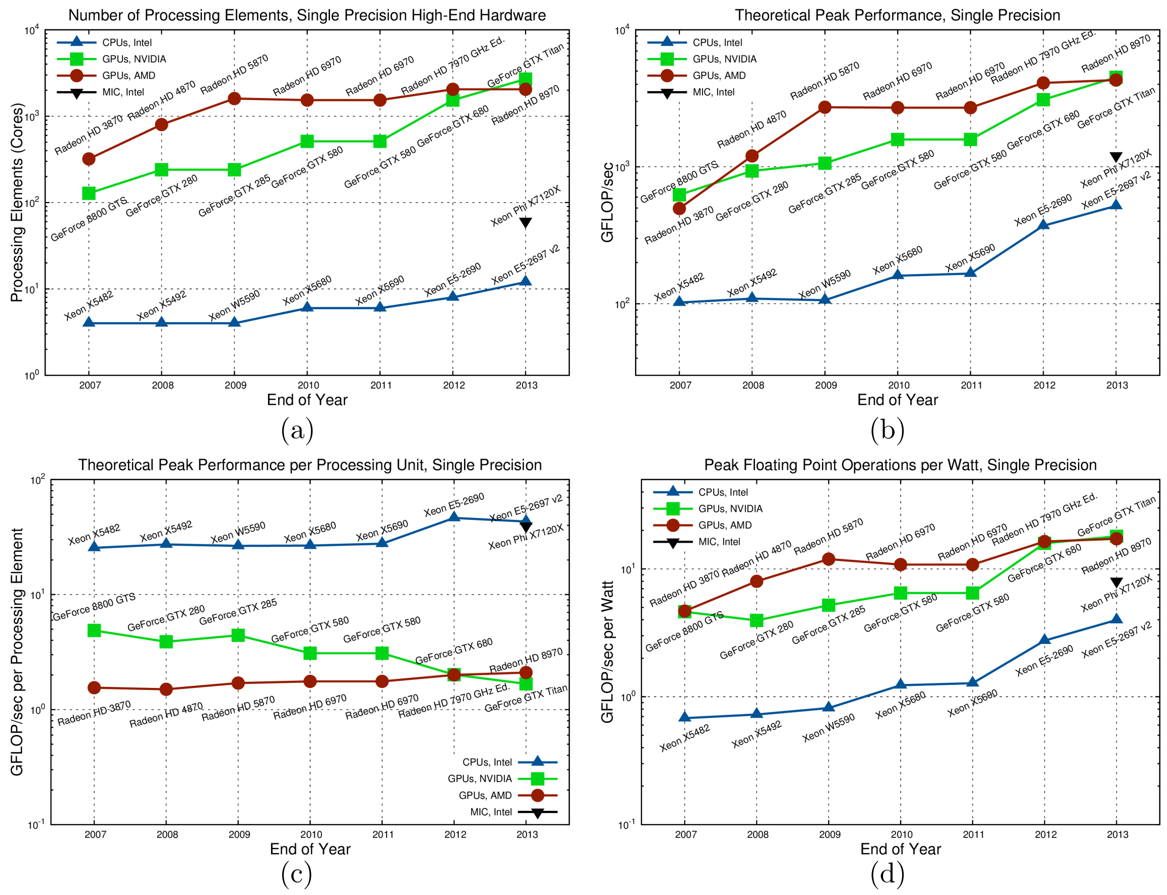


Figure 4.2: Comparison of performance data of some high end processors. AMD CPUs are not included as the trends are largely the same. AMD APUs are not included because they cannot be directly compared to GPUs and CPUs. (a) The number of processing elements per processor; GPUs have considerably mode processing elements than most CPUs. (b) The theoretical peak performance of each processor; GPUs outperform CPUs easily. (c) The performance of the processing elementsCPUs in general have higher performing cores, but GPUs have many more. Enough to outperform CPUs. GPUs have many more cores. (d) Inspite of their higher performance, GPUs are the more energy efficient alternative. Credit to Mark Rupp[162], CC license.

With all of this in mind, it may be concluded that an efficient solution for MD should contain parts that use both CPUs and GPUs. The goal of the work in this thesis is to produce such a program that can simulate cellular behaviour with a model based on MD principles, and accelerate the appropriate parts (force calculations, velocity and position

Table 4.1: Direct Coulomb summation performance comparison. Performance was compared on the CPU with compiled with GCC and ICC and on the GPU with code compiled with CUDA at different levels of optimization. Clearly, GPUs are superior for energy evaluations even with minimal optimization. Data taken from Stone et al. [11].

Kernel	Performance Normalized to Intel C	Billion Evaluations/sec	GFLOPS
CPU-GCC-SSE	0.052	0.046	0.28
CPU-ICC-SSE	1	0.89	5.3
CUDA-Simple	16.6	14.8	178
CUDA-Unroll8x	38.9	34.6	268
CUDA-Unroll8y	40.9	36.4	191
CUDA-Unroll8clx	44.4	39.5	291

updates) on GPUs. This code, in conjunction with the force-field described in Chapter 5, will be used to model cell behaviour.

If following sections GPU architecture and programming will be introduced. The terms used will be CUDA terms for the sake of simplicity, but the concepts are common between CUDA and most other libraries. The remainder of this chapter may seem like it only applies to GPUs but much of it will be a general introduction to programming on any parallel device. Thanks to libraries like OpenCL (which was not used for the program used in this thesis), the principles will remain the same regardless of device.

4.2 GPU Architecture

In this section a brief overview of GPU architecture will be given. A detailed analysis of the design specifications is out of the scope of this thesis as it differs from GPU to GPU depending on brand and model type. A cursory understanding of GPU design should be sufficient to proceed. Readers looking for more detailed descriptions are referred to [12, 13].

Simply speaking, the entire goal with GPU hardware design is to supply a greater number of computing cores to do the calculations. Until recently, GPU hardware was only suitable for image transformations. Newer GPUs are capable of doing floating point math at rates much faster than CPUs. Figure 4.3 shows the complexity of some NVIDIA GPUs, other types of GPUs are not much different conceptually. There are many levels of hierarchy and caching that make the design complicated. A simpler depiction of CPUs and GPUs is presented in Figure 4.4.

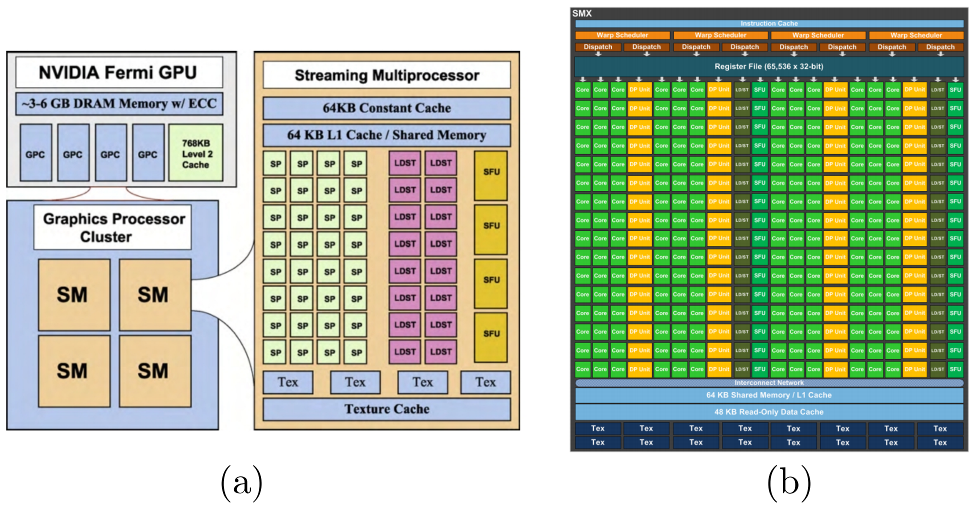
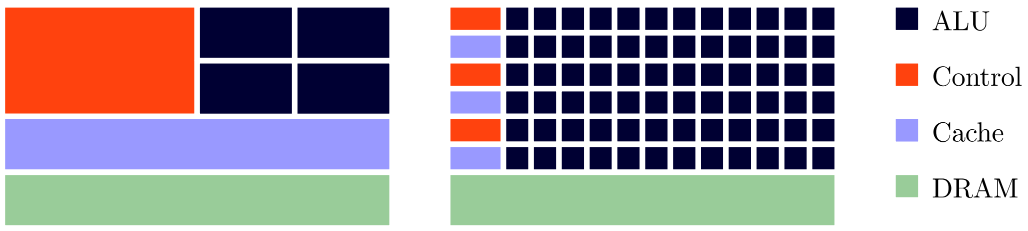


Figure 4.3: Core placement and categorization on two kinds of NVIDIA GPUs. (a) Typical Fermi^{TM} NVIDIA GPU design, the cores are placed within Stream Multiprocessors (SMs) which are themselves inside placed into Graphics Processor Clusters. There are multiple levels of memory (caches) but the most relevant to GPU developers is the DRAM memory. (b) This picture shows a more detailed representation of a Kepler^{TM} type gpu. There are multiple control circuits including circuits that manage the memory accessing and the caching of data in the different levels of cache.

The designs shown in Figures 4.3 and 4.3 are far too complicated to considered exactly when coding, especially when targeting more than one kind of GPU. The structure of the GPU is abstracted into a more tractable form. For the sake of generality, some general terms are defined here. Newer CPUs are being designed with many cores incorporated into them, so it is beneficial to use some common terms that are equally applicable to multicore CPUs, GPUs, and any other parallelized computational devices. The managing device will be named the “Host” and the computational device the “Device”. A host may have any number of devices. A PC may contain one host (the main CPU), and one or more GPUs devices. A large compute server may contain hundreds or thousands of devices. For the remainder of this section, device and GPU are used interchangeably.

As shown in Figure 4.5, each device can be divided into a collections of Compute Units (CU), the compute units access global memory (G) as a group simultaneously. These compute units are nothing but a collection of Processing Elements (PE) (cores) with common Local Storage (LS), each processing element contains within another cache of private



(a) CPU (b) GPU

Figure 4.4: Schematic of CPU and GPU architecture. (a) The CPU contains much more control circuitry and more complex caching and fewer Arithmetic Logic Units (ALUs) which are basically the same as processing cores. Dynamic Random-Access Memory (DRAM) is accessed through the motherboard (a little slower than GPU DRAM) but is shown here for comparison. (b) The GPU is a little more simple but has many more ALUs. This is an important image to bear in mind when targeting the GPU in code. All data is stored in the large DRAM segment (also called global storage) and each read or write to global memory is slow and expensive.

memory (P). G is slow to access and has the highest capacity (approximately 1–10GB), LS is faster with moderate with about half a MB, and P is the fastest with a few KB's of storage. All communication with the host must be done through G memory as it is the only memory accessible to all CUs and the host, LS is only accessible to the PEs in a single CU, and P is only accessible by a single PE. The three levels of memory may be separated again into Constant cache, texture cache, etc for GPUs or L2, L1 cache, etc. for CPUs. But this sub-hierarchy of caches is not as important from the programming aspect normally. They do, however, come into play when optimizing code.

In NVIDIA GPUs and CUDA, a compute unit is named a Stream Multiprocessor SM and processing element is called a Stream Processor (SP). There are also other terms apply to NVIDIA specifically, but do not differ much in their meaning. AMD devices use the same terminology used in figure 4.5, which are OpenCL terms. Figure 4.3 shows the architecture of a fermi type GPU. Typically, each SM contains 32 cores. This collection of cores is mapped to a "warp" of threads which play an important role in memory access, see Section 4.3.3. Each core contains one optimized core floating point calculations and one for integer calculations. Thread hierarchy is based in this hierarchy in the hardware. Each thread block is run on a single SM, see Section 4.3.2.

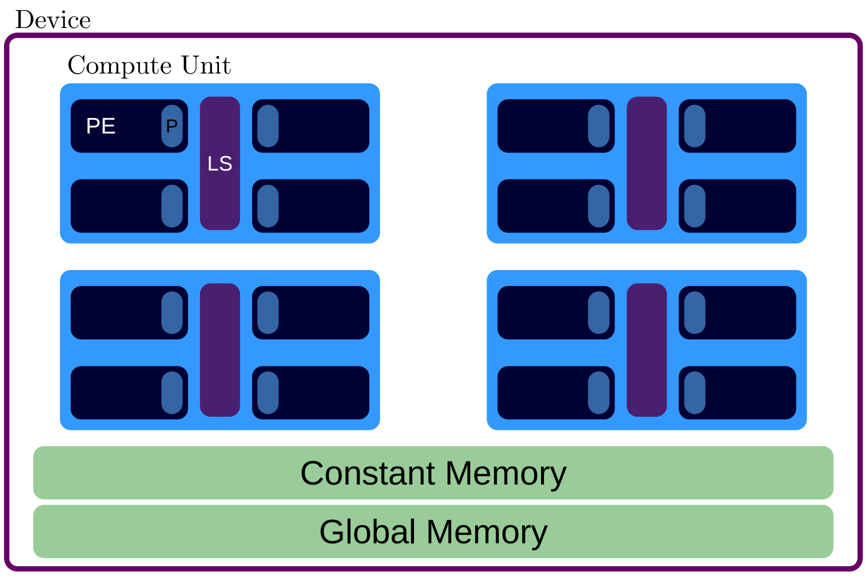


Figure 4.5: From the programmers perspective, any parallel device can be approximated with this structure. Note that the number of elements shown here is just an example, actual devices can have any number of CUs and PEs.

4.3 Programming Perspective

The hardware abstractions being out of the way, the programming perspective will be illustrated in this section. The GPU architecture is important when aiming for the highest performing code. In table 4.1, Stone et al. [11] showed that GPU code without optimization is still very fast. So, when coding, at least initially, a detailed picture of hardware is not needed as much as when optimizing. From the programmers perspective, the GPU is not much different from a CPU with a very large number of cores.

A unit of computational work is named a kernel [171]. Kernels contain a module of code that does a certain task. It can be thought of as a function or subroutine that runs on the compute device (GPU). Listing 1 shows some sample code with the same kernel implemented in C and in CUDA. The kernel can be executed in a serial manner on the CPU or in parallel on the GPU.

CUDA code follows the same standards of C/C++ with some some extensions and differences. The differences from C/C++ include things like lack of recursion [171, 172], no Object Oriented Programming (OOP). CUDA also introduces some new data types and built-in functions to ease programming. There are also some libraries that make programming with CUDA almost indistinguishable from programming in C++ such as Thrust [173] and boost.compute (still in testing, `kylelutz.github.io/compute/`). Some accelerated versions of standard libraries: cuFFT, cuBLAS, CUDA Math Library, etc. There is no need to use C/C++, CUDA supports Python, FORTRAN, C#, and Java.

As an example, consider the task of adding two vectors, $\mathbf{c} = \mathbf{a} + \mathbf{b}$. CPU code is straight forward. This task can be done with a `for` loop sequentially. Since elements of a and b can be summed independently, the operation can be parallelized to execute on multiple cores on the GPU, see figure 4.6. Assuming the CPU and GPU cores have the same clock, the operation can be sped up by eight times.

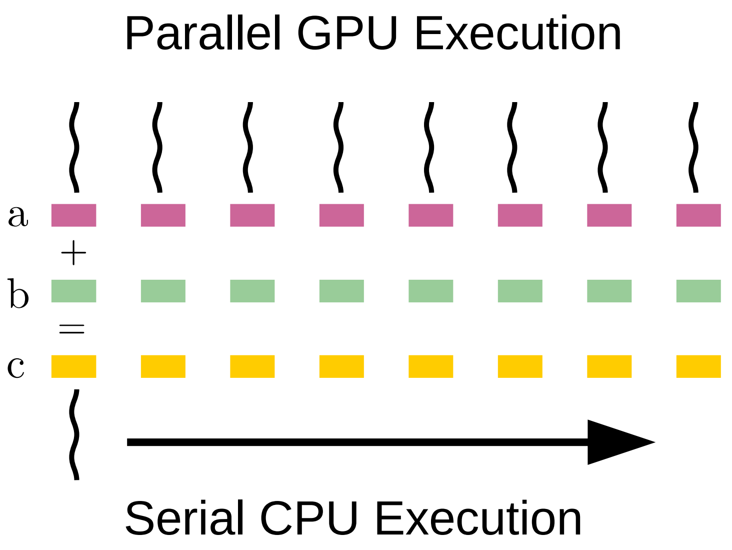


Figure 4.6: Diagram showing single threaded CPU execution and multithreaded GPU execution. Note that the segments representing a, b, and c reside on either host DRAM or device DRAM depending on what device executes them. From the programming perspective (refer to Section 4.3.2), the CPU has one 1D block of one thread and the GPU has one 1D block of 8 parallel threads.

Listing 1 shows CPU C code and GPU CUDA code. CUDA syntax is not very different from regular C syntax. The variable `m` specifies the block size which is the number of concurrent threads. If the size of a, b, and c is greater than the number of maximum possible concurrent threads `GPU_MAX_THREADS`, then `m = n/GPU_MAX_THREADS` and the calculation is looped `m` times, `n` threads running at a time. At first glance, this might seem to be quite a fatal limitation. A very large number of threads can be running on modern GPUs, about

2000 per stream multiprocessor. Another limitation is the block size (explained further in 4.3.2)

CUDA programs are compiled with the CUDA compiler (NVCC) which is a wrapper around GCC. CUDA kernels are written the same way as normal C/C++ function with the exception that they must return nothing (only return `void`) and are defined with the `__global__` keyword.

4.3.1 CUDA Execution Model

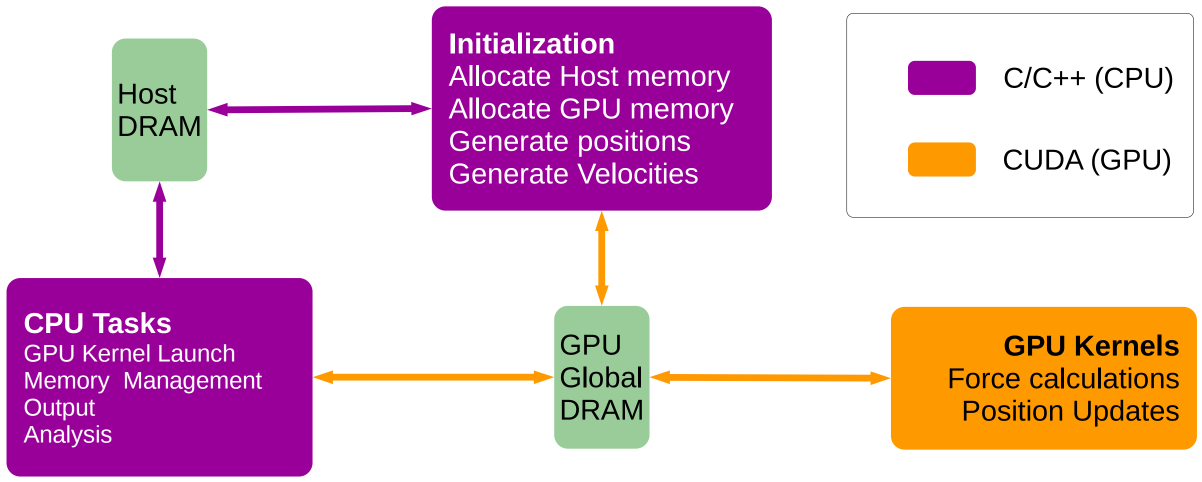


Figure 4.7: Sketch of CUDA's execution model. The program is started on the host. host code then manages allocating memory on host DRAM and GPU DRAM, initializing variables, moving data to GPU Global memory, and managing GPU kernel launches.

Figure 4.7 shows the execution model in a typical CUDA MD program. The execution of CUDA GPU code is handled by the host. The host allocates memory on GPUs and manages the transfer of the memory. The host also manages kernels; it sets the block size `n`, determines the number of blocks to run `m`, and passes the kernels the memory locations that they need (`d_a,d_b,d_c`). CUDA benefits from the fact that kernel calls look very much like normal C/C++ calls, so getting set up in the beginning is straight forward.

At the beginning of the program the host has to search the system for an onboard CUDA capable device and assign in for execution. Managing the simulation over multiple GPUs is possible and also desirable when simulating very large systems. The host has to manage the distribution of the work across the GPUs. GPUs can be thought of as calculation machines that must be managed manually.

All data is communicated to the GPU through the GPU's on board of global memory. Regardless of platform, moving data from one region of memory to another region is very time consuming, especially if the memory is in different locations. Memory transfer must be minimized in order to maximize performance because there is a highly latency in communication with GPUs. Thankfully, the communication line with GPUs has a fairly large bandwidth, 100's of GBs/sec., so a lot of data can be moved at once. Once data has moved to the GPU, it should be kept there until it absolutely must be sent back to the host. Minimizing the need for data manipulation on the host entirely should always be one of the goals of any good algorithm. Thanks to this execution model, the idle CPU time can be used to perform other tasks

Once on the GPU, the simulation can begin in earnest. Ideally, all of the resources of the GPU are being used at maximum capacity and all the host is doing is waiting for the GPU to finish. However, because there is no way for the GPU to access storage drives, or manage memory, the data has to be transferred back to the host every once in a while for output. In the case of the cell division code, data has to be moved to handle the cell division part.

At the end of the simulation, since the host manages GPU memory, it has to handle releasing no longer needed resources. Though, if the program is to end at this point, CUDA is intelligent enough to release that memory when the program ends. Nevertheless, it is good practise to keep careful track of memory because chasing memory leaks on the GPU can be quite frustrating.

4.3.2 Thread Hierarchy

A kernel is a task that applies to large amounts of data. With each kernel, a single block of instructions (a single function) is being applied to a large amount of data. This way of data processing is named Single Instruction Multiple Data (SIMD) in computer science and is the most obvious target for parallel computing. A thread is the concept of a kernel being executed once on some piece of data once. In other words, each kernel is executed as a, potentially large, collection of threads on large amounts of data. Now, the focus is shifted to thread management from the programmer's perspective. It is useful to know the workings of the hardware for optimization, but the hardware complexity is abstracted out in the CUDA and OpenCL API. So, at least in the beginning of a project, it is useful to focus on the basics of GPU programming. Readers interested in more detail are referred to [172]

The size of data that is needed for MD is very large, the position and velocity of, and the force on each atom in a system should be calculated simultaneously. If we take these tasks as one function, then this function is of one (discrete) variable, namely the index of each atom in some master data array containing the coordinates of all atoms. In this sense, this is a 'one dimensional' problem (see Figure 4.7). One can imagine such problems with higher dimensionality. There are many problems in science and engineering that are applied over 2D or 3D grids, for example, the temperature distribution on a metal plate. For conceptual compatibility and ease of programming, it is convenient to bundle concurrent threads in 1D, 2D, 3D, or N-dimensions.

This is recognized in the field of GPU acceleration, and the API is designed with this functionality. OpenCL kernels can be bundled in N dimensions, whereas CUDA is limited to three. Keep in mind that this dimensional bundling does not limit the problems that can be solved at all, one may choose to implement everything in one dimensional blocks if that is desirable. This bundling is only for convenience.

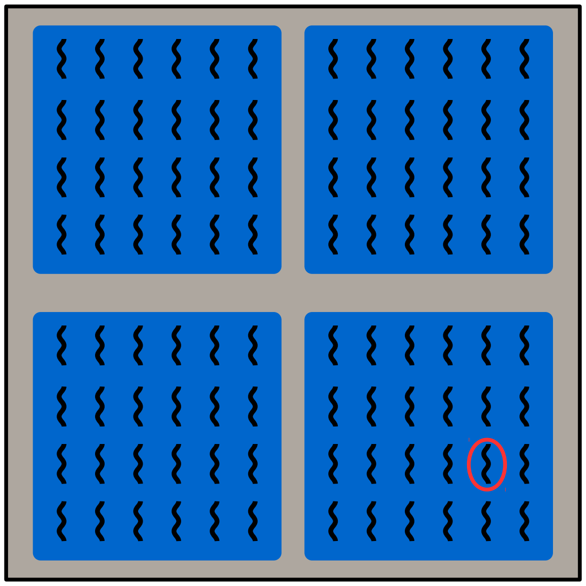


Figure 4.8: Here is a two dimensional set up of thread blocks is shown. In this figure, following the convention in Eq. 4.1, $t_1 = 6$, $t_2 = 4$, $t_3 = 1$ and $b_1 = b_2 = 2$, $b_3 = 1$. So that the thread index of the thread circled red is thread (4, 1, 0) in block (1, 0, 0). That thread also has a global index of (10, 1, 0).

In CUDA terms, all threads are bundled into blocks. In each block, threads can be indexed by a one-dimensional, two-dimensional, or three-dimensional index. Similarly, the blocks can be indexed by one-dimensional, two-dimensional, or three-dimensional index. The total number of threads launched by the host is given by

$$n_t = t_1 t_2 t_3 \cdots t_n \cdot b_1 b_2 b_3 \cdots b_m \tag{4.1}$$

where t is the number of threads in along dimensions $1, 2, 3$, and b is the number the blocks. Of course with CUDA, $n \leq 3, m \leq 3$. Depending on the hardware, there can be constraints on n_t and other strategies will be needed to solve that problem (multiple GPUs or looping over the work with a single GPU). In new CUDA GPUs, $t_{max} = t_1 t_2 t_3 = 1024$, so $n_{t,max} = 1024 b_{max}$. b_{max} depends on the hardware and implementation, it is difficult to determine a priori as it depends on the number of registers available per thread. b_{max} plays an important role in optimizing occupancy. Once again, this is just a limit on how many threads and blocks *can be launched at a single time*. If more are needed, then the calculation can be looped or split among GPUs.

device occupancy is defined as the ratio of the number of active threads per device divided and the maximum possible number of threads per device. Obviously, this should be kept as close to one as possible. The GPU hardware and drivers are designed to do this as much as possible. Maximizing device occupancy is difficult, detailed knowledge of GPU design, API, and thread hierarchy are required. Occupancy is affected by choice of block size, shared memory usage, private memory usage, and hardware limitations.

At first glance this splitting of threads into blocks may seem arbitrary, it actually serves two vital purposes [172]: (1) This indexing gives a precise identity to each thread and this identity can be accessed within a kernel, (2) block and thread breakdown informs memory breakdown and vice-versa. Each thread has private memory to itself, all of the threads have access to shared memory in a block, and all blocks have shared access to global memory. Figure 4.8 shows an example of two dimensional thread blocks. Tuples of indices are used to identify threads, they can have a local index which requires indexing blocks and threads, or global indices. It is necessary to calculate the global index because data spans all of threads. The CUDA code snippet below shows sample code with this indexing in action. Assume that there are three arrays, one for each of the Cartesian axes, that contain the position of all atoms. The positions of can be distinctly accessed with a unique index `i`. With this information, we launch the kernel `KernelExample` once for each atom. So the index of the thread can be converted to the index of corresponding atom.

```
__global__ KernelExample (float* Xdata, float* Ydata, float* Zdata){
  printf("BlockID = (\%d, \%d, \%d), ThreadID = (\%d, \%d, \%d)\n",
          blockIdx.x, blockIdx.y, blockIdx.z,
          threadIdx.x, threadIdx.y, threadIdx.z);
 // Below since Ydata spans all thread blocks, we get the global
 // address of a thread by multiplying by blockDim.y
 int i = blockIdx.y*blockDim.y + threadIdx.y;
 float y = Ydata[i];
}
```

Special reserved variables such as `threadIdx` store the index of each thread. These variables are used to access data in global memory, or in shared memory among threads in the same block.

4.3.3 Memory Hierarchy and Access

The placing of threads in separate blocks brings control to memory access and is an important factor in optimizing performance. All threads always have access to global memory. There is also shared memory that is made available to all threads of the same block. A block of threads will only every run on a single Stream Multiprocessor (PE in Figure 4.5). And then, each thread has private memory to itself. Private is the fastest to access, shared a little slower, and global memory is the slowest to access [12, 172]. Shared memory and private memory is temporary and has the same lifetime as its block or thread respectively, global memory persists over many kernel calls, as long as the host does not release it.

Not only is global memory slow to access, but access to global memory has to be done in certain ways to maximize speed. Memory access occurs 16 threads at a time regardless of how many threads are requested to access memory [172]. 32 threads are named a warp in CUDA terminology, so memory access occurs in half-warps. This means block sizes must be multiples of 2, as required by CUDA, and also multiples of 16 to maximize memory access efficiency. OpenCL does not require even block sizes [174], but still benefits greatly by making block sizes a multiple of 16 on NVIDIA devices.

```
__global__ void MemoryAccessKernel(float* array){
  int index = blockIdx.x * blockDim.x + threadIDx.x;
  float element = array[index];
}
MemoryAccessKernel<<<1, 1>>> (float* arrayWithOneElement);
MemoryAccessKernel<<<1, 16>> (float* arrayWith16Elements);
MemoryAccessKernel<<<1, 17>> (float* arrayWith31Elements);
```

The first two kernels will take the same amount of time to complete, while the last kernel call will take double the time of the first two kernels even though it is accessing almost the same number of memory locations as the second kernel call. Memory access latency is very often a major bottleneck in kernel execution. The following section contains some strategies that can be used to maximize memory access performance, and maximize performance in general. Following these strategies is beneficial, but should not be a priority until after the software is shown to be correct and without any bugs.

4.3.4 Some strategies to enhance GPU performance

Generally speaking, kernels can be divided into three types: latency bound, compute bound, and memory bound.

(1) **Optimizing latency bound kernels**
Latency bound kernels depend on their execution by the host. They are optimized by adjusting the execution parameters that were set by host code. Latency problems can occur when not threads are not broken down into blocks optimally.

  (a) **Maximize occupancy**
  The GPU has built-in circuitry that launches units of work on each SM, if the work given is not enough to saturate this process, then performance is wasted/lost. This requirement is difficult to meet and should be left to the end of the project. Maximizing occupancy helps greatly to shifting the slowest step to memory access.

    (i) **Saturate the GPU**
    Adjust block size, private memory, and shared memory usage such that there is always an abundance of work to be done by the hardware.

    (ii) **Concurrent kernel execution**
    The host normally waits until after the kernel has finished executing to resume. If adjusting the launch parameters does not help, many kernels can be launched concurrently.

(2) **Optimizing compute bound kernels**
Compute bound kernels are kernels that spend most of their time doing calculations, this type of kernel can be optimized with algorithmic improvements. Sometimes, simple changes in the way calculations are carried out can result in significant performance gains.

(a) **Uniform execution path**
The GPU executes instructions uniformly for an entire warp. In order to maximize usage, all threads should have the same execution path (run the same code). This problem only occurs if the divergence is among threads of the same warp (same block).

(i) **Avoid thread branching**
Some tasks require divergence depending on the data being processed. In those cases, it may be beneficial to decide what data needs to be processed on the host, and launch kernels that work with only that data. The code snippet below shows a bad example of thread branching—the first 6 threads are idle with this execution path.

```
if (threadIdx.x > 5)
DoSomething();
else
DoNothing();
```

(b) **Loop unrolling**
There is a performance benefit to type out each iteration of `for` loop. This increases the amount of work done for each thread so efficiency is improved significantly.

(c) **Intrinsic/Fast Math**
If exact matching with CPU is not required, then there are intrinsic optimized versions of common math functions that should can be used for performance. For example, `__sinf()` vs. `sinf()`. Readers are referred to CUDA/OpenCL documentation [174, 175] for more details regarding intrinsic functions.

(3) **Optimizing memory access**
Memory bound kernels are kernels that spend time accessing memory. These kernels can optimized by correctly requesting memory reads (or writes).

(a) **Coalesced memory access**
As noted before, memory access is done on a per warp, in some cases per half-warp depending on GPU, basis. Furthermore, there is a minimum transaction size for each memory read. On Kepler GPUs it is four warps (128 threads) [176], since memory transactions are always done n warps at a time, it is good practise to assume minimum transaction of 32 threads. Additionally, It is always preferable to access contiguous blocks of memory at a time. That means that threads in a single warp should be accessing adjacent regions of memory always. If this is done properly, memory access can be "coalesced" into a single operation.

(i) **Align memory access in a block**
Thread index should be aligned with element index to maximize performance. For example, thread 0 accesses element 0, thread 1 accesses element 1, etc. Threads can also be aligned with an offset, which should be a multiple of minimum transaction size (32 bytes). Thread 0 accesses element 8, etc.

```
// example of aligned memory reads
float a = bigArray[threadIdx.x];
// with an offset
float b = bigArray[threadIdx.x + 8];
```

Below is an example of inefficient access. Every third element is being accessed.

```
float a = bigArray[threadIdx.x*3];
```

Finally, we have an example of misalignment, this occurs when the offset is not a multiple of the minimum transaction size of memory.

```
float a = bigArray[threadIdx.x+13];
```

(ii) **Avoid broadcasting**
Broadcasting is the access of multiple threads to the same, memory location. Instead constant memory should be used. Not only is many threads reading from the same memory location inefficient but writing to it can lead to difficult to debug race conditions.

(iii) **Avoid concurrent thread writes**
The opposite of broadcasting is when multiple concurrent threads (i.e. threads in the same block) attempt to update a single memory location. This leads to undefined behaviour and is named a race condition. Race conditions do not necessarily lead to lower performance but can lead to incorrect results and complicated bugs.

```
/****** CPU code segment start ********/
void add (float* h_a, float* h_b, float* h_c, int n){
  for (int i = 0; i < n; i++){
    h_c[i] = h_a[i] + h_b[i];
  }
}
add(h_a, h_b, h_c, n);
output(h_c);
/******** CPU code segment end ********/

/****** GPU code segment start ********/
__global__ void add(float* d_a, float* d_b, float* d_c){
  i = blockIdx.x*blockDim.x +  threadIdx.x;
  /* blockIdx.x iterates from 0 to m-1
     blockDim.x equals n
     threadIdx.x iterates from 0 to n-1
  */
  d_c[i] = d_a[i] + d_b[i];
}
HostToGpuCopy(h_a, d_a);
HostToGpuCopy(h_b, d_b);
add<<<m, n>>>(a, b, c);
/* m = 1 if n <= Max number of threads (can be 512 or 1024)*/
GpuToHostCopy(d_c, h_c);
output(h_c);
/******** GPU code segment end ********/
```

Listing 1: Adding two vectors is not very different when comparing GPU and CPU code. The number of threads to launch is specified by host code and device code contains special variables with the thread index information.

Chapter 5

Methods and Implementation

As described earlier, modelling cell behaviour could be highly challenging due to highly complex nature of most cells. An average cell contains many multitudes of different molecules, large and small, that all come with their own particular complex behaviours and interactions. These complex interactions are, of course, all highly interesting on their own, but they unfortunately make studying the cell as a whole an intractable problem.

Fortunately, cell-cell interactions can be approximated through a variety of approximations, see Chapter 3. However, these approximations tend to be ad hoc in the sense that they can only model limited behaviours of cells in certain regimes.

We will approach building this model from two angles. The first being the theoretical basis, physics and mathematics, for the model. And, naturally, the second will be actual implementation in C++. C++ was chosen for out simulation because of the variety of tools and libraries available for acceleration. We will be using CUDA for our acceleration needs.

The cells are modelled with Molecular Dynamics techniques, where each cell is comprised of a spherical membrane made of a fixed, though optionally can be varied, number of particles in a force field. The force field itself contains terms for the surface tension of the cell membrane, the adhesion (or friction) between cells, the repulsion between cells, and a term for the drag that the cell experiences in its medium. This model is based on work by Mkrtchyan et al [15] in 2014, and Åström and Karttunen [14] in 2006.

The simulation is then optimized to run on GPUs. The optimization method will be outlined in some detail and some strategies that can be employed to further optimized this code will be discussed.

In Section 5.1, the model will be defined and described in detail. Section 5.2 describes the values that are given to the parameters that are part of the model, and Section 5.4 describes the implementation with CUDA and the design choices that were made to increase performance.

5.1 The Force-Field

First we will define the force-field of the particles in our simulation. There are two broad categories within the force-field that have to be defined. That is done in the following sections.

In Section 5.1.1, the assumptions that are made with this simulation method are described and justified. It is important to begin with a relatively simple model to begin, then more complexity and detail can be added if needed.

In Section 5.1.2, the structure of the model cell will described in detail, this includes the geometry of the particles that make up the cell and the physics of the cell membrane. To summarize, each cell is modelled using the C180 molecule [177–179] as a template, where the particles and bonds are treated as balls and springs.

Section 5.1.3, the interactions that govern the interactions between the particles within a cell are described. The forces defined here govern the integrity of the cell body and regulate its growth.

In Section 5.1.4, the methods used to model inter-cellular interactions will be described. These include the final elements of the MD force-field. We will see that interactions between cells is broken down into an adhesive force and a repulsive force. Another force is a dissipative one that is used to model the environment around the cells.

5.1.1 Assumptions in the Model

The ultimate goal of this work is to simulate the behaviour of cells as completely as possible. However, we first begin with a simpler model then, more complexity and detail can be added if needed. Hence, some assumptions are made to simplify the construction of the model.

1. **There is only one cell type.**
   All of the cells in the system will be identical in structure. They will be allowed to behave independently. Some of the remaining assumptions follow from this one.

2. **All cells grow at the same rate.**
   If able, all of the cells will grow and the same rate. To be more clear, the driving force behind cell growth, discussed in section 5.1.3, will be the same for all cells. Cells also grow at the same rate whatever their surface areas.

3. **External conditions do not affect the simulation.**
   External conditions such as pressure, temperature, humidity are ignored. These can be indirectly modelled by setting the parameters discussed in sections 5.1.3 and 5.1.4, and therefore can be easily introduced into the model at a later time. Another consequence of this assumption is that the availability of nutrients in the system, and the cells' access to those nutrients, are both ignored. However, these can be included back into the model.

4. **Cell death is ignored.**
   In reality, cells die either because they have depleted their lifetime, due to unfavourable conditions, or both [180, 181]. However, this fact is ignored. Though it is possible to add cell death to the model, it will not be explored in this thesis.

5. **The cells exist in an infinite space.**
   The cells will not be confined spatially, at least initially, in anyway. This not only means that the cells will not be hindered in any way, but also means that there are no periodic boundaries or any other boundaries of any time.

6. **All of the cells are spherical in their natural state.**
   This comes from the fact that animal cells favour a rounded structure due to their cortex contractility[35, 36].

7. **Cell-cell interactions are isotropic**
   Cells can interact differently depending on direction. But this is ignored for our model, the cells in our simulations interact uniformly with each other.

8. **Hertwig's rule can be ignored.**
   The cell division plane will always be random and through the centre of mass, see 5.3.

9. **Each daughter cell will be identical.**
   The model will not, initially, do any asymmetric cell division, see 5.3.

5.1.2 The Model Cell

The cell itself will be modelled as sphere made up of a number of particles with uniform mass. The mass of the particle in itself is an interesting parameter and will be discussed in more detail in Section 5.2. Figure 5.1 is a sketch of how such models are constructed.

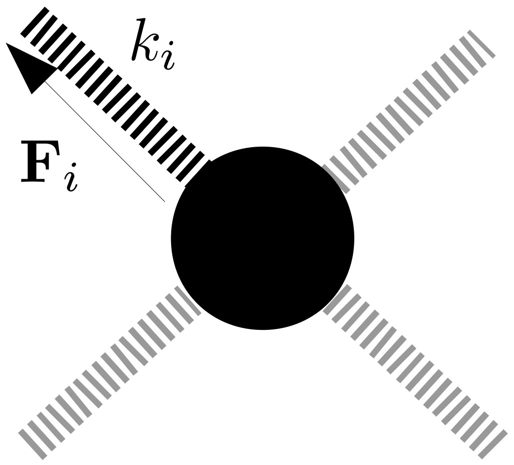


Figure 5.1: The ball and spring model. Figure shows a particle with an arbitrary number of bonds. This particle has N springs connected to it.

We can easily calculate the forces on the particle as shown in equation (5.1). Basically, we do a vector sum of the force caused by each spring,

$$\mathbf{F}^S = \sum_{i=1} \mathbf{F}_i = \sum_{i=1} -k_i \mathbf{x}_i, \tag{5.1}$$

where $\mathbf{x}_i$ denotes the displacement vector of the particle from the equilibrium position of with respect to spring i. The spring force becomes an important element of the force field used in our Molecular Dynamics simulation. Given this very basic modelling technique, we can build particle model for the whole cell. However, we must first come with a general method to represent the cell within our simulations.

A generalized spherical geometry is needed to build the cell itself. Coming up with this geometry is not a trivial task. Because to be usable for the simulation, it must meet certain criteria listed below.

Criteria for choosing the cell geometry:

Uniformity:
The geometry should be more or less uniform. In other words, the distances between particles should not change by very much over the surface of the sphere. This is to simplify the parametrization of the simulation in the future. Ideally, all of the particle pairs will have the same bond length[1]

Simplicity:
The geometry should be simple and well understood. The coordinates of the particles that make up our sphere should not be difficult to calculate. Ideally, an already known geometry should be used.

Versatility:
The geometry should accommodate the physical changes that are expected with cell behaviour. In other words, the geometry should allow for deformation and expansion without breaking down. Additionally, the geometry should be easily matched with basic cellular structure. i.e. spherical cell should be modelled by a spherical geometry, ellipsoidal with ellipsoidal geometry, etc.

Spatial Resolution:
The behaviours of cells can be highly complex as discussed in Section 2.1, so there should be enough points to accurately describe these behaviours. Conversely, while it is true that a high resolution is needed, it must not be so high that it overwhelms the available computational resources.

Taking all of the above criteria into account, a simple model for the cell can be created in 2D, however, this is not true for the 3D case. Mkrtchyan et al. showed a good sketch of the 2D model [15] which is reproduced in Figure 5.2. Figure 5.3 shows the 3D extension to this model.

[1] This way we can only worry about setting one bond length in our simulation. If in our geometry, there are many possible bond lengths for each bond, then they would each have to be set manually. This way we eliminate the degrees of freedom introduced by the geometry of the model cell. The effect of bond length can be studied in greater detail at a later time.

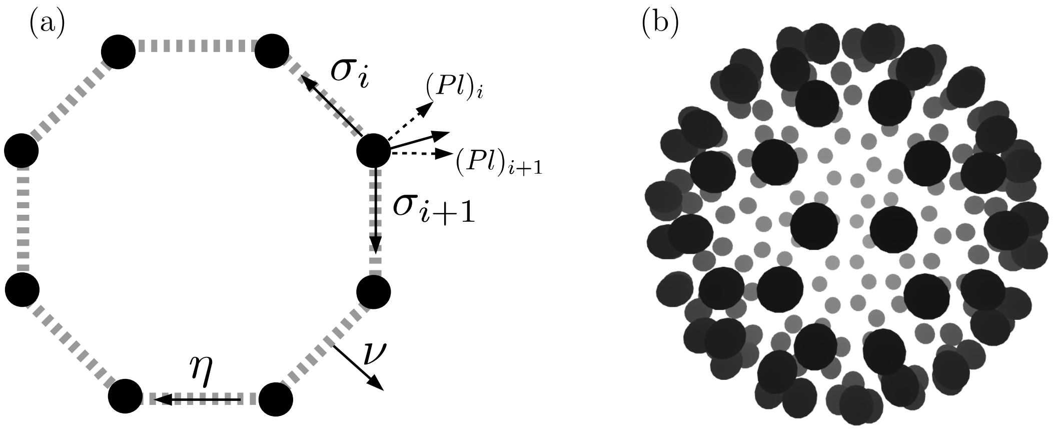


Figure 5.2: (a) The 2D model of the cell, all of the points are in the plane of the page, the various forces that act on each mass point are defined in Section 3.3 and [15]. (b) The 3D model, the number of particles that make up the membrane is variable in theory, though it is 180 in all of the simulations run for this thesis. The cells tend to stay spherical unless they interact with surrounding cells or other surfaces in the system such as walls.

The 3D model itself is not much more complicated than the 2D model mathematically or conceptually, the complication arises from the numerical calculations associated with the additional degrees of freedom introduced by modelling the system in 3D. Another note is that each particle is bonded to three neighbours in the 3D model, one more than the 2D case.

The 3D geometry of the model cell may seem familiar, this is because the geometry is borrowed from the C180 fullerene [177–179]. The geometry of this fullerene is well understood and fairly simple. One can quite easily find the coordinates of each atom from a variety of sources in the literature [36, 179, 182], another popular resource is M. Yoshida's library of fullerene geometry [183]. The 180 particle mesh is also of sufficient resolution to be able to model the cells, while keeping the load fairly low. A few thousand cells amounting to less than one million MD particles can be modelled comfortably [65]. If a higher resolution is needed, then this geometry can be changed to a larger fullerene. The particles are spaced more-or-less uniformly over the surface of the fullerene, so a uniform equilibrium bond length (equilibrium spring length) can be assumed. Lastly, the geometry is approximately spherical, this is to ensure that we start from a simple model. Most cells tend to favour a spherical shape[35, 36], and different kinds of fullerenes exist that can be used to model different types of cells, though that is left for later research.

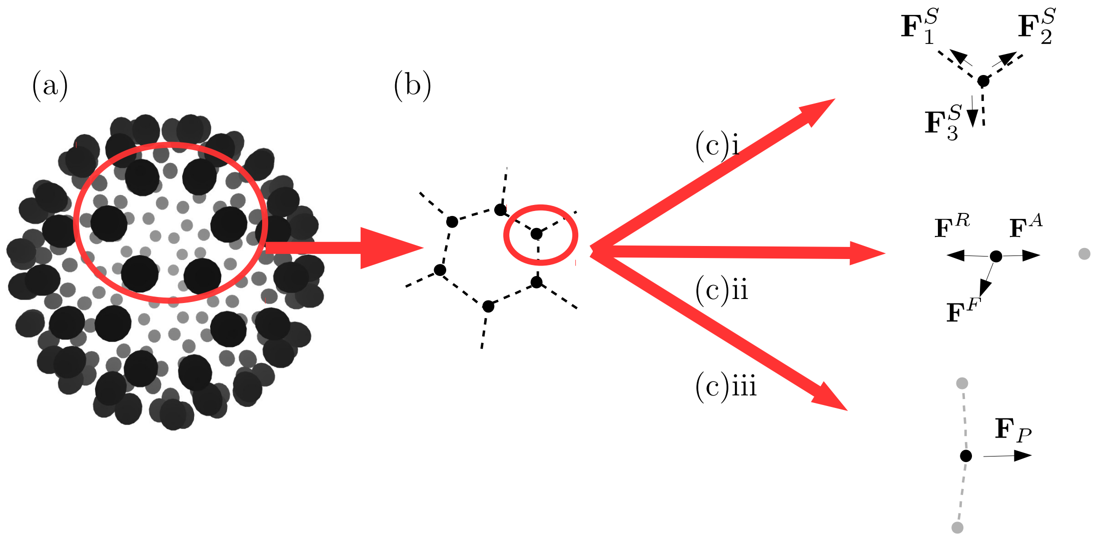


Figure 5.3: (a) The 3D geometry of the cell. (b) One isolated face of the cell, note that each cell is bonded to three neighbours. (c) The different types of interactions that each particle experience in the force-field. There are spring forces (i), Attraction, repulsion, and friction forces (ii), and a pressure force that governs the cell growth (iii).

Referring to Figure 5.3, we can now see a more detailed expression for the force on a single particle in the cell shown in equation (5.2),

$$\mathbf{F}_i = \sum_{j=1}^{3} \mathbf{F}_{ij}^S + +\mathbf{F}_i^P + \mathbf{F}_i^R + \mathbf{F}_i^A + \mathbf{F}_i^F, \tag{5.2}$$

where $\mathbf{F}_{ij}^S$ is the spring force associated with neighbouring particles within the same cell, $\mathbf{F}_i^P$ is the pressure force acting of the membrane of the cell, $\mathbf{F}_i^R$ is the repulsive force between the membranes of different cells, $\mathbf{F}_i^A$ is the adhesive force between membranes of different cells, $\mathbf{F}^F$ contains the terms for friction between cells that hinders cellular rearrangements [15] $\mathbf{F}^{F,e}$ and $\mathbf{F}^{F,m}$ the friction with extracellular medium.

With the basics of the model in mind, we can proceed to define the force-field in more detail. The force-field can be broken down into two basic categories. The interior of the cell and the exterior inter-cellular interactions.

5.1.3 The Cell Interior

There are two competing forces that act in the interior of the cell. The spring interaction between neighbouring particles, and the pressure force that will maintain the structure of the cell and also lead to the growth of the cell. The parameters shown below are in dimensionless units, the basis for this scheme will be shown in Section 5.2.

The Spring Interaction force.

In reference to equation (5.2), we need to define the $\mathbf{F}^S$ terms in the force-field. As already shown in equation (5.1).

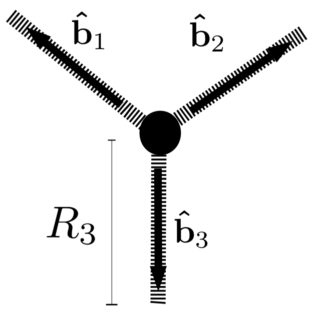


Figure 5.4: The unit vectors along each bond are shown. These are the vector terms used to then calculate the spring for contribution.

Using the description of the bond[2] vectors, we can represent the spring force explicitly. Now we need another index, j, to index the neighbouring particles.

$$\mathbf{B}_{ij} = \mathbf{r}_i - \mathbf{r}_j \tag{5.3}$$

$$R_{ij} = \|\mathbf{B}_{ij}\| \tag{5.4}$$

[2]Note that this is not bonding between atoms in a molecule, it is just the attraction between points on the mesh that forms the cell membrane. It comes from cortical actin filaments that give the cell its shape.

$$\hat{\mathbf{b}}_{ij} = \frac{\mathbf{B}_{ij}}{R_{ij}} \tag{5.5}$$

$$\mathbf{F}^S_{ij} = E\frac{A}{R^o_{ij}}\hat{\mathbf{b}}_{ij}(R_{ij} - R^o_{ij}) \tag{5.6}$$

Where E is the Young's Modulus of the spring and A its cross-sectional area, and in our case we set the cross-sections to be the same, so we can simply redefine $k^s = EA/R^o_{ij}$ and 5.6 would be equally valid. The new k^S variable is related to the cell cortex contractility, which depends on the structure of the cell membrane and will differ between different cells. Additionally, we can assume that the equillibrium bond length is the same for all particles. So we set $R^o_{ij} = R_o$, and

$$\mathbf{F}^S_i = \sum_{j=1}^{3} k^S \hat{\mathbf{b}}_{ij} \left(R_{ij} - R_o\right) - \gamma_{int}\mathbf{v}_{ij}. \tag{5.7}$$

The motion of the mass points in the spring interaction cannot be left to oscillate freely when interacting with each other. So an internal damping term, $-\gamma_{int}\mathbf{v}_{ij}$, is introduced to make sure that the mass points behave well where $\mathbf{v}_{ij}$ is the relative velocity of neighbouring mass points.

The Pressure Force

The internal pressure of the cell is the second element of the force-field that applies to the within the cells. The pressure force has is simpler by definition, the pressure is simply the ratio between a force and the area it is applied to. In this case, the area that it is applied is simply the surface area of the cell S. The internal pressure of each cell will apply a uniform force on each particle in our membrane mesh. So the magnitude of the pressure force on each particle will be equivalent. Cells are known to regulate their pressure before mitosis [35, 37],

$$F^P_i = P_n S_n, \tag{5.8}$$

where n indexes each individual cell. However, referring to our assumptions (5.1.1), we see that the cells will have the growth rate will be, independent of surface area. Hence our pressure can be redefined as:

$$F_i = P_i S, \tag{5.9}$$

where both P_i and S are constants. Thus we can regulate the growth rate by setting $P_i S$. It is also possible to regulate the population of the system by developing a relationship between $P_i S$ and the number of cells to study population dynamics in the future.

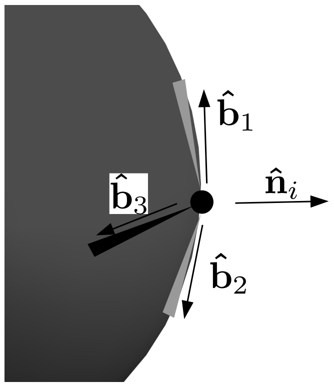


Figure 5.5: Diagram showing the normal vector to the surface of the cell pointing from the particle i. This normal vector gives us the direction of the pressure force.

From figure 5.5, we can then calculate the normal vector and use equation (5.9) for the pressure force on each particle as

$$\mathbf{n}_i \approx (\mathbf{b}_1 - \mathbf{b}_3) \times (\mathbf{b}_2 - \mathbf{b}_3) \tag{5.10}$$

and

$$\mathbf{F}_i^P = P_i S \hat{\mathbf{n}}_i \tag{5.11}$$

This formulation follows the assumptions that we made in 5.1.1, assumption 2, on page 59, states that the cell division shall not depend on the size of the cell. Therefore, PS is treated as its own variable that is not a function of cell radius r_{cell}.

At any given pressure, the springs in that are used to model the cell cortex will extend or contract to attain an equilibrium volume. There is a threshold volume V_D at which cells divide, with a corresponding pressure $P_D S$. The growth rate of a cell can be thought of as the rate at which the volume of the cell increases. That rate of volume increase is then set by the rate of increase of internal pressure $\Delta(PS)$. $\Delta(PS)$ is set to be the same for all cells, $P_i S = P_i S + \Delta(PS)$ increases at a constant rate until V_D is reached, then the $P_i S$ of each cell is optionally reset to a lower volume, and the process is repeated for each daughter cell.

5.1.4 Inter-cellular interactions

Cell-cell interaction is highly complex and varied [184–186]. Modelling such complex interactions is not a trivial task [14, 15, 62, 64]. Furthermore, the interaction type, strength, and directionality may vary greatly from cell to cell. However, we assumed that our cells will interact isotropically with each other. This simplifies things for us greatly, as our cell-cell force-field need only depend on intercellular distance. We can completely generalize the cell-cell interaction force into two parts, adhesion and repulsion. We have an additional term in the force-field for the viscous damping (or friction) force.

The attraction and repulsion forces are implemented as springs as well, with one difference. They both will have a cut-off distance. This means that the springs only react to compression or to extension, but not both. The repulsion and adhesion terms are both calculated for individual particles that are within repulsion and adhesion range of each other.

The repulsion force is needed to prevent the cells from penetrating one another, and the adhesion force is to simulate the interaction between different cells,

$$\mathbf{F}_i^R = \sum_j \mathbf{F}_{ij}^R \tag{5.12}$$

$$\mathbf{F}_{ij}^R = \begin{cases} -k^R \left(R_c^R - R_{ij}\right) \hat{\mathbf{r}}_{ij} & \text{if } R_{ij} < R_c^R \\ 0 & \text{if } R_{ij} \geq R_c^R \end{cases} \tag{5.13}$$

$$\mathbf{F}_i^A = \sum_j \mathbf{F}_{ij}^A \tag{5.14}$$

$$\mathbf{F}_{ij}^A = \begin{cases} k^A \left(R_c^A - R_{ij}\right) \hat{\mathbf{r}}_{ij} & \text{if } R_{ij} < R_c^A \\ 0 & \text{if } R_{ij} \geq R_c^A \end{cases} \tag{5.15}$$

where j indexes the particles near the particle i but not part of the same cell as i. Here four new parameters are introduced. R_c^R and R_c^A are the repulsion and adhesion cuttoff radii respectively. These have to be set manually in the simulation. Theoretically, the would be related to the interaction ranges of different cells. And k^A, k^R are the spring constants that set the strength of the adhesive and repulsive forces. The constants k^A, k^R can be related to the spring constant k^S so that they can be set in comparison to the spring force. $k^A = Ak^S$ and $k^R = Rk^S$, where A and R are the adhesive strength and repulsive strength, respectively.

The third term in the cell-cell interaction part of the force-field is the viscous damping. As cells move past each other, there should be friction between the two cell membranes. This friction would depend on the relative velocity of the cells. The relative velocity of two cells can reduced down to the relative velocity of particles in the membranes of the two cells $\mathbf{v}_{in}$,

$$\mathbf{v}_{in} = \mathbf{v}_i - \mathbf{v}_n \tag{5.16}$$

$$\mathbf{F}_i^{F,e} = \sum_n -\gamma_{ext} \mathbf{v}_{in}^{\tau} \tag{5.17}$$

$$\mathbf{v}_{in}^{\tau} = \mathbf{v}_{ij} - (\mathbf{v}_{ij} \cdot \hat{\mathbf{n}})\, \hat{\mathbf{n}}, \tag{5.18}$$

where $\mathbf{v}_{in}^{\tau}$ is the tangential component of the relative velocity, γ_{ext} is the damping the mass points experience with points of other cells.

Finally, we have another friction. This force arises from the friction of the cells with their environment. Basically, there is some liquid, such as water perhaps, that hinders the motion of cells. The velocity of the particles is used to calculate this force,

$$\mathbf{F}_i^{F,m} = -\gamma_m \mathbf{v}_i \tag{5.19}$$

Thus, we have defined all of the terms in equation 5.2. Now we proceed to give values to all of the parameters (constants) that appear in the force-field.

5.2 Parametrization

The conceptual design of the simulation has been described in previous sections of this chapter. Now, the numerical values of the various constants are described. This is necessary to be able to compare the results of simulations to experimental results.

The average volume of a human cell HeLa [187, 188] can be taken as $V_{avg} \approx 10^3\,\mu\text{m}^3$ [189, 190]. Using this, let's take the unit of volume in the simulation to be [V]$\approx 10^3\,\mu\text{m}^3$. Consequently, [L]$\approx$10 µm. Stewart et al. [35] measured the internal pressure of cells and found that it was roughly 0.1nN µm^{-2}, which can be rewritten as 10^{-8} N [L]$^{-2}$. With, the unit volume [V] given above we can approximate the total surface area of a cell with A≈5 [L]^2. The outward force acting on each mass point is then given by $F^P \approx 5 \times 10^{-8}$ N. Each cell has 200 mass points, so that the force on each mass point will be 1×10^{-9} N. Therefore the unit of force in the simulation will be set to [F]=10^{-9} N.

Stewart et al. [35] also estimated the Young's modulus of mitotic cells and found them to be E ≈10^{-9} N µm^{-2} = 100 [F] [L]$^{-2}$. E is related to k^s with $k^S = Y\frac{A_o}{R_o}$, where A_o is the cross-sectional area of the cortex interconnects (thickness of the springs) and R_o is the equilibrium bond length. A_o approximated by $A_o \approx 1$ [L]^2 assuming a diameter of 100 nm, and R_o with $R_o \approx 0.1$ [L] which is set by scaling the C180 structure to a volume of 1 [V]. Then, the result is k^S=1000 [F] [L]$^{-1}$.

The adhesion and repulsion spring constants are defined by comparing their strength to the spring forces. Mkrtchyan et al. [15] reasoned that the repulsion forces should be stronger than internal pressure forces. This follows form the fact that a pressure force of $(PS)_o$ is pushing outwards against neighbouring cells, so more rigid cells require stronger repulsion between them. With this in mind R is set to $(PS)_o$, so that the repulsion force is always a factor of PS stronger than the spring force ($k^R = Rk^S$); $PS \approx 100$ as the internal pressure will be on the order of 100. With the same reasoning, the adhesion strength A is set to 0.5 so that more rigid cells experience stronger adhesion, but still less than the attraction between points in the cell membranes ($k^A = Ak^S$). These values will be varied and their effects studied later.

HeLa cells have a mass of 10^{-12} kg on average [191]. Given that the model cells have about 200 mass points, the unit of mass can be set to [M]≈10^{-13} kg by setting the mass of each cell to 10 [M]. Setting the mass of the cell to 10 [M] is done by arguing that the mass of each mass point should be high enough to reduce motion as much as possible which is achieved by setting the mass of each mass point to M=0.05 [M], $\gamma_m \approx 10$, $\gamma_{ext} \approx 1.0$, and $\gamma_{int} \approx 100$. With this, the unit of time can be calculated with $[\text{T}] = \sqrt{[\text{M}][\text{Length}]/[\text{F}]} \approx 10^{-5}s$. Considering the fact the animal cells can take anywhere from an hour to a day[16], this unit is too small to set the time step in the simulation. Instead, the time step by measuring the number of time steps needed for a cell to divide, which is 1000 in the simulation's case. The time for a cell to go through one division is roughly $T^{div} = 1000\Delta t$. Δt is set to 10^{-4} [T] for algorithmic reasons (stability and run time).

The growth rate of cells is set in the simulation with the parameter $\Delta(PS)$, which is not going to be any one value. It will be varied to study what effects it has on the results for simulations. The internal pressure of each cell will be regulated with $\Delta(PS)$ increments between a minimum and maximum pressure. The minimum pressure is set to $(PS)_o = 10$

which keeps cellular volume at roughly 1 [V]. Maximum pressure is chosen considering the algorithm to ensure stability, it will generally not be greater than 100 [F] [L]$^{-2}$.

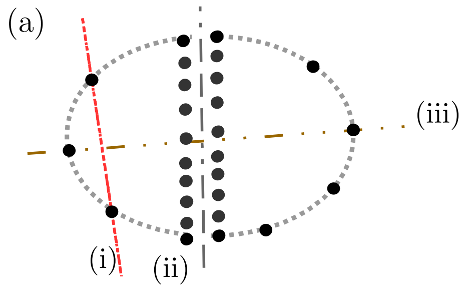
Table 5.1: Listing of all of the required parameters by accelerated cell dynamics simulations. Some of these will be varied to see what effects they have on cell behaviour.

Parameter Name	Variable	value
Particle mass	m	0.05
Number of mass points	N	180
Equilibrium spring length	R_o	0.1
Repulsion Range	R_c^R	0.2
Attraction Range	R_c^A	0.3
Spring Constant	k^S	1000.0
Adhesion Strength	A	0.5
Repulsion Strength	R	100.0
Intercellular Damping	γ_{ext}	1.0–10.0
Internal Damping	γ_{int}	100.0
Medium Damping	γ_m	0–20.0
Division Volume	V_D	2.9
Time Interval	ΔT	0.0001
Pressure Increment	$\Delta(PS)$	0.008
Maximum Pressure	$(PS)_1$	50.0–100.0
Minimum Pressure	$(PS)_o$	10.0
Threshold Pressure	$(PS)_{th}$	63.3

5.3 Modelling the Cell Division

Here the division of the cell will be discussed. In order for cell division to proceed, the mathematical basis for the division of the C180 molecule must be discussed first. The division does not occur in the model spontaneously, e.g. growth occurs spontaneously due to the pressure. Therefore, there needs to be a systematic way of dividing the appropriate cells, those that are over some minimum volume, into two daughter cells.Figure 5.6 shows a sketch of the different possible division methods.

Oscar Hertwig described a common way in which cells divide [30, 45], the long axis rule. The cells tend to divide along the longest axis of cells. Another type of division is asymmetric division where the two daughter cells are not identical [30, 192]. Finally, we



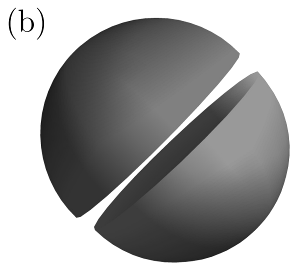


Figure 5.6: (a) The division schemes shown in 2D: (i) asymmetric Cell Division, (ii) Hertwig's rule [45], new generated points shown, (iii) random division through the centre of mass. (b) Random division through the centre of mass in 3D, the new points are not shown.

also have random division line selection through the centre of mass, this is the method that will be used in our model. Figure 5.6 (b) shows division by random division plane in 3D cells.

Division using Hertwig's rule only differs from the random division case when cells are not spherical. When they are spherical, as is the case in our model, division methods (ii) and (iii) are identical. However, cells can be deformed due to their interaction with the surroundings. In that case Hertwig's rule may be applied and it will differ from case (iii).

After the division plane is chosen, the C180 geometry is divided in two and reconstructed for the two daughter cells. This geometry is unfavourable for the cells and the force-field naturally brings the cell back into spherical shape over a short period of time.

This framework allows the modelling of all three division schemes, symmetric random, longest axis (Hertwig's), and asymmetric. So far, only random symmetric has been implemented, the other schemes will be implemented in the future.

5.4 Implementation with CUDA

Implementation of the accelerated MD code is explained in this section. As mentioned before, CUDA will be used to accelerate our simulation. Due to the complexities of GPU acceleration some techniques are needed to maximize the usage of the GPU. On top of that, we have to be careful to avoid the potential issues associated with this type of coding. If not done correctly, using the GPU can actually be detrimental to performance [12, 164].

A simple description of the simulation is given as a series of steps on page 74. The code was initially developed by Jan Westerholm from Åbo Akademi University, in Finland. Subsequently, other features were added to conform with the requirements for this project.

5.4.1 The Division of Labour

The reasoning behind GPU acceleration is to use the maximum amount of stream processors (or processing elements) simultaneously. This can be done with multiple levels of parallelization. The work of doing all of the calculations associated with the simulation needs to be consolidated into appropriate kernels, and the kernels have to be coordinated by the host. This is conceptually simple, but can be difficult to implement.

As mentioned in earlier sections, the force field can be divided into two parts: Inter and intra-cellular interactions. However, at the implementation level, both of these interactions are calculated at the same time, by the same level. The distribution of tasks is not related to the definition of our force-field. So, if needed, the two aspects of this work can be modified without affecting each other.

The simulation partitions the work of calculating the forces according the to some rules that are set by default in CUDA. CUDA assumes a multiple of 32 threads per block being run [171, 175] . Putting it simply, this means that the number of threads that can share memory must be a multiple of 16. The maximum number of threads per block depends on the Compute Capability of the GPU and is usually 512, or 1024, threads per block [171, 175].

As mentioned before, each cell is modelled by 180 mass points. To simplify the simulation algorithm, it is useful to let one cell be operated on by a single block of 180 threads, each thread operating on a single mass point. So that each block of threads will operate on 180 coordinates. However, recall that memory access should be coalesced. Since 180 is not a multiple 32, the data structure for each cell is padded with 12 zeros to make 192 threads per block. Note that this does not mean that calculations of only one cell can be done at a time, that would depend on the hardware structure of the GPU.

The code snippet below summarizes the points in this section.

```
int nBlocks = no_of_cells;
int thPerBlock = 192;
```

```
  // function call to simulate the cells
  SimulateCells <<nblocks, thPerBLock>> (mass_point_coordinates,
                                          simulation_parameters);
```

5.4.2 Description of the Code

A simple explanation of the simulation code follows. We focus on the most relevant parts of the code.

Initialization

First, we must initialize the simulation. During initialization, the data structure that will be used to store all relevant data, which is mostly coordinates of the membrane particles, will be created. Additionally, the limits of the hardware must be probed as those are needed to calculate the maximum size of our system.

A vital portion of the initialization phase, is setting the limits of the simulation. The simulation process requires a large amount of memory allocated on the host and the GPU to proceed. Additionally, due to the large amount of time required to allocated memory, the maximum amount of available memory must be allocated at the beginning and used without modification. This is an indirect way of saying that we must set a maximum for the number of cells in our system before we begin the simulation.

Simulation

The next step is to perform the simulation. All of the starting cells have been created in the previous phase of the simulation. This step is by far the most complex and the most time consuming.

Initially, the pressure of the cells must be set. This can be set to be a constant or to a growing value. There is a threshold pressure below which the cells will not grow, this threshold depends on the parameters set for the simulation.

The system must first be divided into lists of neighbouring cells. This is needed to calculate the dissipative force $\mathbf{F}^V$. A data structure containing the corresponding neighbours for each cell is populated.

Another crucial step is to update the positions of the particles. Three phases of the particle positions are stored. The previous time step positions, the current time step positions, and the next time step positions. In this step, the previous time step data is updated to current time step data, the current time step updated to the next time step data. The next time step data is effectively discarded. Then we calculate the forces and the future time step positions with the current time step and past time step positions,

The velocity Verlet, shown in Equation (5.20), method is used to calculate the next time step coordinates and the positions.

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r_i}(t-\Delta t) + \frac{\mathbf{F}_i(t)}{m_i}\cdot \Delta t^2 \tag{5.20}$$

$$\mathbf{v}_i(t+\Delta t) = \frac{1}{\Delta t}\left(\mathbf{r}_i(t+\Delta t) - \mathbf{r}_i(t)\right) \tag{5.21}$$

Interlaced in between these three steps is the code that does some of the analysis portion of the simulation. Namely, the counting for the mitotic index [193, 194].

Program End

Not much needs to be done at the end of the program run, all of the memory allocated at the beginning is simply release and output files closed.

Program Summary:

1. Parse Arguments
   The arguments of the simulation must be parsed in order to begin the simulation. These arguments include values for the parameters that are used in the force field, the starting number of cells, additional simulation parameters such as whether there any walls (none by default), and the path to any output files.
   Assumptions:

   1 There is a CUDA capable GPU on the system not requiring a network connection. Running on a networked GPU is possible with CUDA, but is not supported by the code. Non NVIDIA GPUs are not supported.

   2 There are no other programs running that required the use of the GPU. This means that should a computer only have one GPU, it should not be using the GPU for any rendering. Alternatively, a computer with a dedicated GPU for simulation runs can be used.

3 There is enough space on the system available for arbitrary size data files.

2. Allocate Memory
   Allocate enough memory for a maximum of 250,000 cells. This is a very large number of cells and it is set only for the purpose of allowing a simulation environment that is as close to infinite as possible. It is not easy to deduce the population trend of the simulation from the input parameters, so a large "sandbox" is allocated. Note that the simulation does calculate the relevant quantities for all 250,000 cells from the start. The number of cells grows naturally over time.
   Assumptions:

   1 There is enough available memory on the GPU for 250,000 cells. This is close to 2GB of memory. Allocating memory while taking into account the memory available on the GPU is possible, such as 1GB or 512MB, or any other amount.

   2 In the case of more than one GPU is available, the first one, as set by the operating system, will be used. The simulation will not run on any other GPUs.

   3 Once allocated, memory need not be managed again. This means that there will not be any other programs running that will required access to this allocated memory (this assumption is valid for most software.)

3. Initialize the system
   Now, we must create the starting cells and make the necessary preparations for the simulation. There are two phases to this step. First, the positions of the centres of the starting cells (the centres of the C180 fullerenes) must be determined. Then, the positions of the atoms themselves do not change relative to their centres. The positions of the atoms themselves are read from an available data file.
   Assumptions:

   1 The cells are placed in a uniform 2D grid. This is to simplify the initialization and debugging. Other configurations can be added easily.

   2 The cells all have the same orientation, i.e. they are not rotated relative to each other. Given that the cells are spherical, this is theoretically true regardless of orientation. However, rotation would introduce minor variations in the C180 configuration. Granted that these variations would have no effect given our force-field, this possibility is eliminated nevertheless.

4. Find nearest neighbours
   There are dedicated kernels that handle the finding of nearest neighbours in the sys-

tem. These kernels are optimized to use the fastest algorithms described in Chapter 4. This step is repeated every time step.

5. Set pressure (growth rate)
   The growth rate of the cells must be set before the force calculations may proceed. This is because depending on which phase of their lives the cells are in, they will have a different pressure. However, for the sake of simplicity, the pressure is made equivalent for all cells. They will grow at the exact same rate regardless of their state.

6. Force calculations and position updates
   The force calculation is done for each particle in each cell. This part of the simulation is most susceptible to inefficiencies and may be the slowest step.

   1 Spring force calculation

   2 Pressure force calculation

   3 Internal Damping force calculation

   4 Interfullerene Force calculations (adhesive and repulsive)

   5 Time propagation

7. Analysis and cell division
   Here some analysis of the cells is done which is needed to determine which cells are ready to divide. The volume of each cell is calculated by using the geometry of each cell. Those cells with high enough volumes are divided. The division is not spontaneous and is done by introducing new mass points and separating the mass points of the parent cell into two daughter cells. The division plane is chosen so that it divides the parent into two equal halves, see Section 5.3.

Chapter 6

Results and Discussion

Until now the focus has been on the development of the model and simulation code. In this chapter, results of running this simulation code will be shown. The cell dynamics model described here began in 2006 when a basic version of this model was introduced by Karttunen and Åström [14]. Later, working with Mkrtchyan as well, they improved the model and were able to show a variety of cell behaviour [15]. That work was done entirely in 2D, the next step was to extend the model into 3D. The results shown in this chapter are a result of that work. Figure 6.1 shows some representative snapshots of a simulation run with the simulation code. At first glance, the cells seem to look like real cells. The powerful Graphics Processing Unit (GPU) hardware blazes through the simulation and can produce such systems in a few minutes. The system in Figure 6.1 contains 1100 cells, which can be simulated for $100T^{Div}$ in about 10 minutes ($\approx 10^{10}$ force calculation and position updates).

To compare with the 2D model of Mkrtchyan et al. [15], a system resembling epithelia was to be created. This is done by confining the cells between walls separated by a gap approximately equal to the diameter of a cell. This is a simple way of simulating epithelia which are two dimensional tissues. Figure 6.2 shows snapshots of cells confined within walls. All of the systems are initialized with a group of cells placed in a grid, which is why the snapshots seem to be square shaped. This shape is lost after longer simulation time.

6.1 Mitotic Index

As cell populations grow and pass through the various cell phases, the ability of cells to reproduce may change over generations. As tissue matures, some cells cease to divide, or

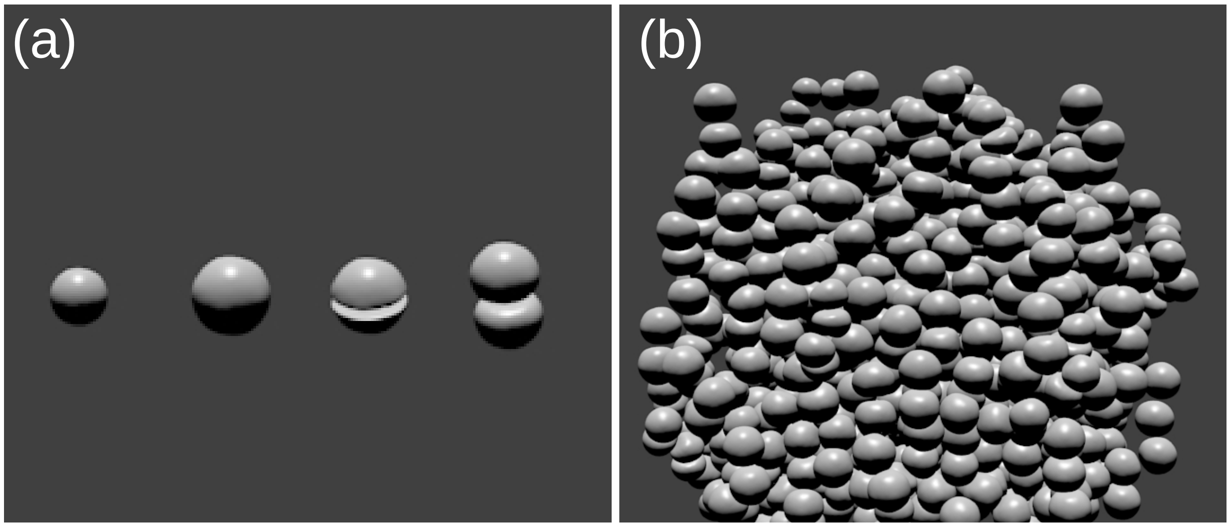


Figure 6.1: (a)Here a sample cell is shown at various stages of its cycle. The cell begins at a relatively small size, grows until it reaches a threshold V_D, then is divided into two cells. The system grows by successive divisions in this manner. The cell surface has been smoothed in post processing, the cells actually look something like what is shown in Figure 5.2(b). This sequence is over about $1T^{Div}$. (b) The system after $100T^{Div}$ time has passed, this image contains roughly 1100 cells. This state can be reached within 10 minutes of run time thanks to the GPU acceleration. These images were generated with the Blender software package [195].

divide rarely. Some other types of cells are able to reproduce rapidly and consistently, such as cells of the apical meristem in plant roots [196] or unimpeded cancer cells [193, 194].

A standard measure of proliferation (mitotic activity) is the Mitotic Index (MI). MI is basically the fraction of cells in mitosis in a sample of tissue. This index is useful in determining the health[197] or age of the cells being examined[198]. It is a useful indicator of the effects of the environment on cellular activity[199, 200]; MI can go up or down depending on how the environment affects the cells. For example, MI is used to measure the efficacy of chemotherapy on some types of cancer cells[193, 194, 200]. Typically tissue begins with a relatively high mitotic index and as it matures the mitotic index decays to a constant value[158]. Figure 6.3 shows the mitotic index of *Drosophila* wing disc measured by Wartlick et al. [158]. This type of exponential decay is expected with normal systems of cells.

Mitotic index is simply the fraction of cells undergoing mitosis at any one point in time. It is measured as

$$MI = 100 \times \frac{N_M}{N_c}, \tag{6.1}$$

where N_M is the number of cells in mitosis and N_c is the number of not dividing cells in the system. N_c can also be the total number of cells [194], or some constant such as 1000 [201]. In the results below, N_C will always be the number of not dividing cells as that is how it was calculated by Wartlick et al. [158]. This can be configured differently to compare to other experimental results. Figure 6.3 shows comparisons between the model of Mkrtchyan et al. [15] and some experimental results by Wartlick et al. [158].

Figure 6.4 was created with data from 3D simulations. A qualitatively similar trend of mitotic index can be seen in 3D as well as in 2D. The first few data points are ignored in the fit because they are measured at somewhat artificial conditions. The cells all start with identical sizes and internal pressures, so the mitotic index always starts at 100% as all cells divide at the same instant. The very high mitotic index can be seen in the inset of Figure 6.4 which contains a plot of average mitotic index of the same 10 simulations. This effect goes down dramatically and the behaviour becomes more comparable to experiment very quickly. Future versions of the simulation code will include options to randomize initial cell conditions, which may reduce this effect.

This behaviour of mitotic index can be rationalized by looking at what effects the growth of the system has on the inner most cells. As the number of cells increases, the outer cells constrain the growth of the inner cells, so the cells near and on the surface of the tissue can grow more easily. As the surface area to volume ratio decreases with increasing system size, the ratio of cells near the surface to cells inside the tissue goes down as well, which leads to a lower mitotic index overall. From a biological perspective, perhaps the cells near the surface of a system have easier access to nutrients than inner cells in the absence of vasculature.

Equation 6.2 is the equation of the function fitted to average mitotic index (blue line in Figure 6.4). The fit starts at ∼2.74% approaches 1.58% over the run time. The decay of mitotic index begins at approximately 4.32%, which is close to the experimental results shown in Figure 6.3. In Figure 6.4(b) the mitotic index evolution for multiple vales of γ_{ext} are shown. It seems that all of the trends are within the margin of error of each other. This is expected because γ_{ext} is only hinders the motion of cells past each other. Cell growth seems to be unaffected by γ_{ext}. Later, it will be shown that the medium can affect mitotic index.

$$MI_{3D} = 2.2737exp(-0.027182t) + 1.5802 \tag{6.2}$$

The next step is to observe how the mitotic index is affected by γ_{ext} in conditions similar to epithelia. An epithelium is a system of tightly bound cells in two dimensions. Examples of epithelia include the epidermis[202, 203] the wings of *Drosophila melanogaster* (fruit fly)[158, 204]. Epithelial cells were simulated by confined 3D cells between rigid walls set approximately one cell diameter apart. Figure 6.5 shows how the mitotic index of cells confined between two walls changes with time. In this case, mitotic index reduces quickly and becomes approximately constant. The confinement was done to simulate epithelial tissue. Experimental studies of the *Drosophila* wing disc show that, on average, the mitotic index is constant on average at about 1.7% [205]. This is in qualitative agreement with the results shown in Figure 6.5 which shows quick decay to constant mitotic index, though the actual average mitotic index produced by the simulation differs. Equation 6.3 reveals an average mitotic index of ∼0.7% for $\gamma_{ext} = 10$. But the mitotic index behaves unpredictably at higher γ_{ext}'s. The unpredictable nature of the mitotic at higher γ_{ext}'s may be caused by undefined behaviour in the event of a cell dividing with an approximately horizontal division line (Figure 6.5(c), far right), too high intercellular friction prevents the two daughter cells from separating and finding enough space to grow. This explains the lower mitotic index at $\gamma_{ext} = 15$, but not the higher mitotic index of $\gamma_{ext} = 20$. In Section 6.2, investigations of cell packing (Figure 6.9) will reveal strange behaviour at $\gamma_{ext} = 20$ there as well. It is unclear at this point what is happening at $\gamma_{ext} = 20$ for confined systems.

$$MI_{2D} = 1.1551 \exp(-0.094358t) + 0.7198 \tag{6.3}$$

Lastly, the effects of friction with the medium, set by γ_m, will be investigated. Readers are reminded that γ_m affects all cells equally, unlike γ_{ext} which only acts upon neighbouring cells. Figure 6.6 shows the mitotic index at different values of γ_m. The plots of mitotic index show that as the medium becomes more viscous, the cells take longer to divide the mitotic index reacts more slowly. The slowing down of division also leads to lower steady state values of mitotic index. This demonstrates the susceptibility of mitotic index to environment conditions.

The initial results are at least qualitatively comparable to experimental results. This means that the framework developed in the code can used with correctly determined parameters to study real cells more realistically in the future.

6.2 Cell Packing

In isolation, many cells prefer a roughly spherical shape. Though in tissues where cells interact strongly with neighbours, cells can take a more polygonal shape [151]. In two dimensional tissue such as epithelia, experiments have shown that cells pack together with a particular topology. Lewis[152] showed that epithelial cells pack as mostly hexagonal cells, with lower fractions of pentagonal and heptagonal cells. Gibson et al. [151] later showed that this distribution of cell packing is conserved among different species, which suggests a common mechanism to the emergence of this packing. Figure 6.7 shows the cross section of the cells that form an epithelium, and a Voronoi tessellation done to determine the number of neighbours that the cells have.

Figure 6.8 shows the fraction of cells with different number of nearest neighbours for a the cells of a number of different species. The rough shape of a cell is determined by the number of neighbours it has, cells are usually in contact in tissue. So pentagonal cells have five neighbours, hexagonal have six, etc. There tends to be a relatively high proportion of hexagonal cells, with lower portion of pentagonal cells, lower portion still of heptagonal, and then some small amount of other polygons. It stands to reason that the packing of different cells is affected by intercellular interactions. Operating on this assumption, the effects of γ_{ext} on cell packing are studied in this section.

Figure 6.8 shows how the packing behaves with changes in γ_{ext}. It appears as if the distribution of polygon types is indeed affected by γ_{ext} with comparison to experimental results of *Drosophila* taken from [158]. Most of the different values of γ_{ext} produce distributions that are comparable to the experimental results in Figure 6.8(a). Except the strange behaviour at $\gamma_{ext} = 20$ is seen again. Further study into this value of γ_{ext} will be done to determine if this effect is real (valid in the context of the simulation) or a result of some error in the code. $\gamma_{ext} = 20$ was simulated many times with the same result. Furthermore, strange behaviour with the same value of $\gamma_{ext} = 20$ was seen in Figure 6.5(b).

Figure 6.10 shows the effects higher values of γ_m have on cellular packing. Higher γ_m hinders all cell motion more, which prevents cell rearrangements and growth. The flat regions of Figure 6.10(b) inset can be explained by noting that higher γ_m slows down cell growth and rearrangement, so that the fraction of cells jumps abruptly between different values and can stay constant over longer periods of time. The distribution at the end of is simulations is somewhat comparable to experimental ones though less so when compared to Figure 6.9.

6.3 Summary

Mitotic index and cell packing with various parameters were tested in this section. The results shown in this chapter aim to demonstrate the efficacy of the model, that was developed throughout Chapter 5, by comparing two aspects of cell behaviour to experimental results.

The mitotic index was replicated successfully for 3D free cell systems, and mostly for cells confined in between two walls as well—except for one case in which unexpected behaviour was seen. Differences in cell packing were also observed in the packing distribution of simulated cells. The reason for this aberration is unclear and will be studied further in the future. It is likely that this is caused by bug in the simulation code. There is also the remote possibility that the cell behaviour actually changes drastically at specific values of adhesion strength, this will be revisited again once the code has been exhaustively tested.

At least at a qualitative level, the results shown in this chapter can be taken as a preliminary indication of the validity of the 3D model that was a result of the work done by Karttunen, Åström, and Mkrtchyan in 2D, and the work done as part of this thesis in 3D. The next step would be refine the model further to enable it to reproduce general as well as specific cell behaviour.

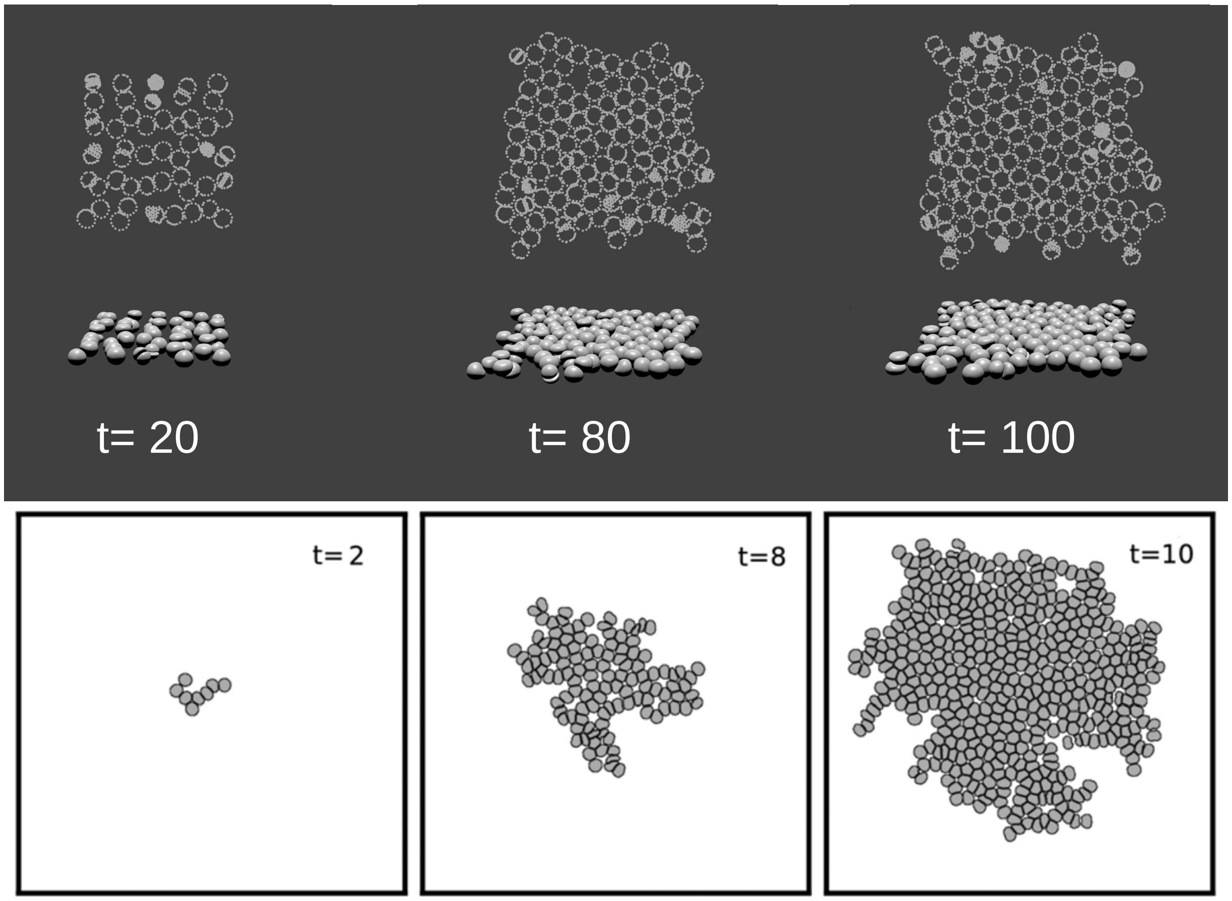
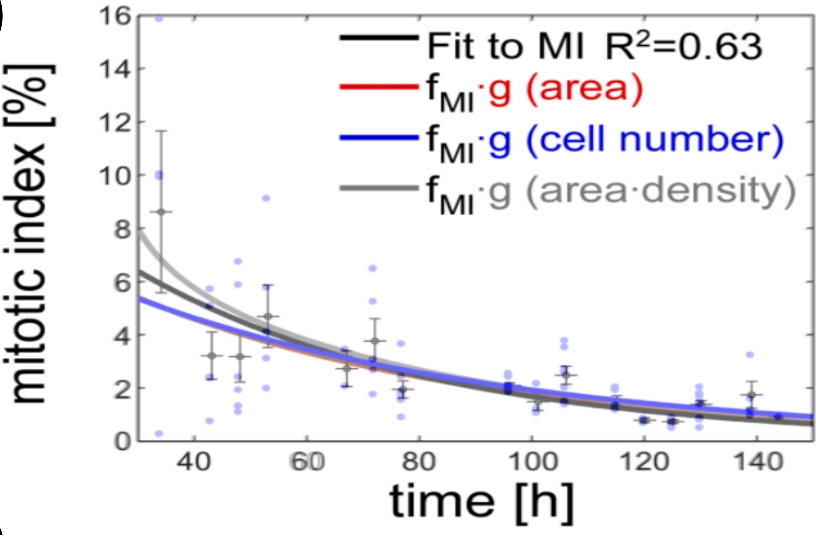


Figure 6.2: **Top: Snapshots of an epithelium at three different times chosen to coincide with the 2D system of Mkrtchyan et al. [15] The system was set up as a grid of cells, which is why it has a square like shape, this shape is lost after longer simulation time. The dotted loops are cross sections of cells at roughly halfway between two walls in the XY plane. Bottom: Snapshots of the 2D model, taken from [15], reprinted with permission from the Royal Society of Chemistry. Note that the 2D model uses a unit of time that is 10 times larger.**

(a)



(b)

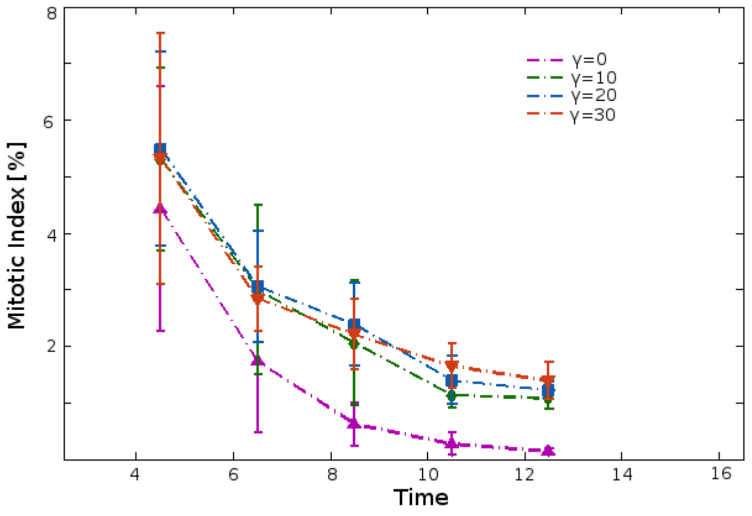
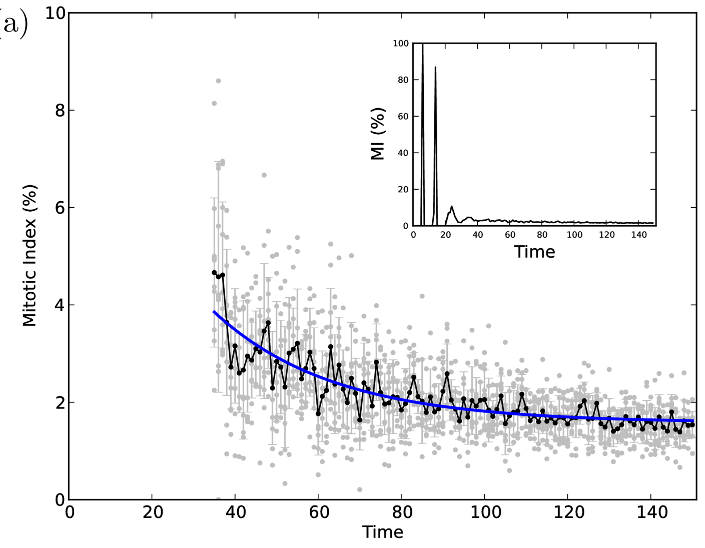


Figure 6.3: (a)Measurement of MI of *Drosophila* citeWartlick2011. The black line is the fit to the MI data, the other lines are the growth rate (can be related to MI) fits which are, for the purposes of this thesis, irrelevant. The mitotic index starts off at roughly 6% then decays to about 1%–2%. From [158], reprinted with permission from AAAS. (b) Mitotic index with 2D simulations at different γ_{ext} with the code of Mkrtchyan et al. [15, 49], the friction term between cells which affects cell rearrangements. Adapted from [15] with permission of The Royal Society of Chemistry.

(a)



(b)

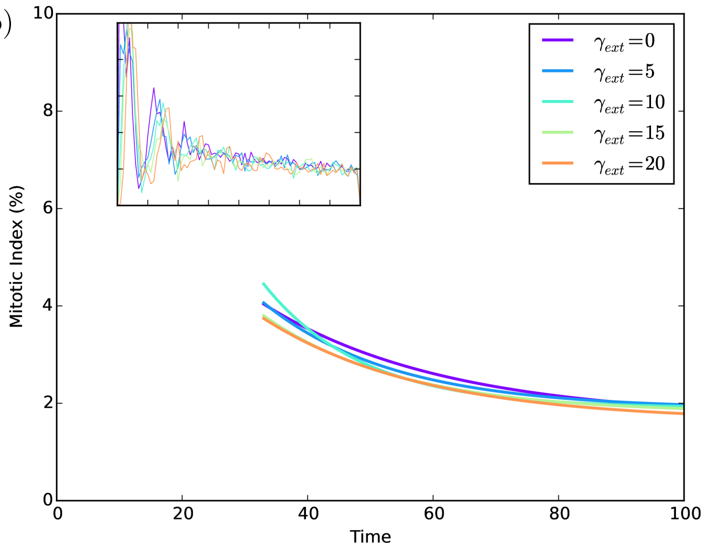
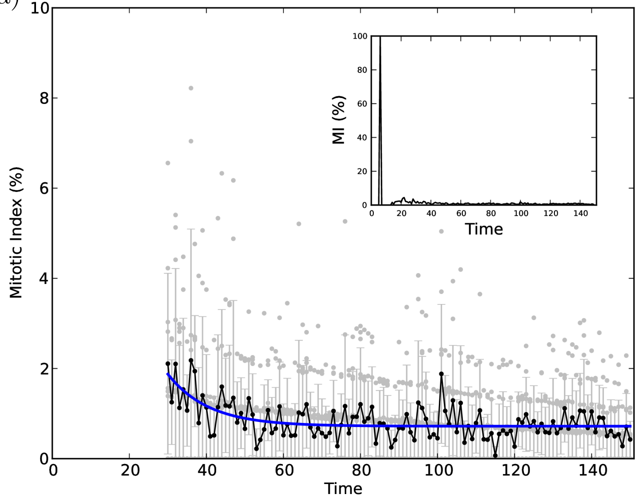
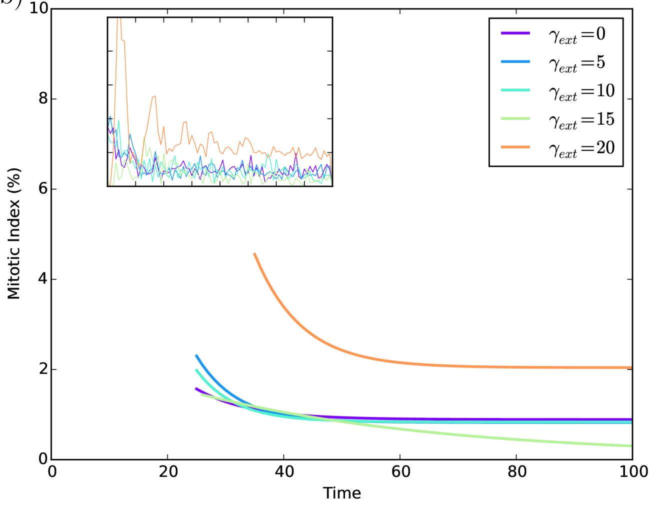


Figure 6.4: (a) Plot of mitotic index of data generated by 3D accelerated cell dynamics. The blue line shows a fit to the black points which are the average of 10 simulations (grey points) done at $\gamma_{ext} = 10.0$. The first twenty or so time points were ignored when creating the fit as the first few data points are always unnaturally high due to the initial conditions of the cells. (b) The same data for different values of γ_{ext}. The noisy lines are the average mitotic index of 10 simulations. Measurements were done at T^{Div} intervals (1000 time steps).

(a)


(b)


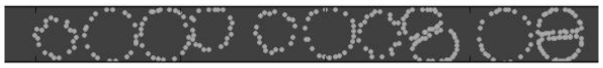
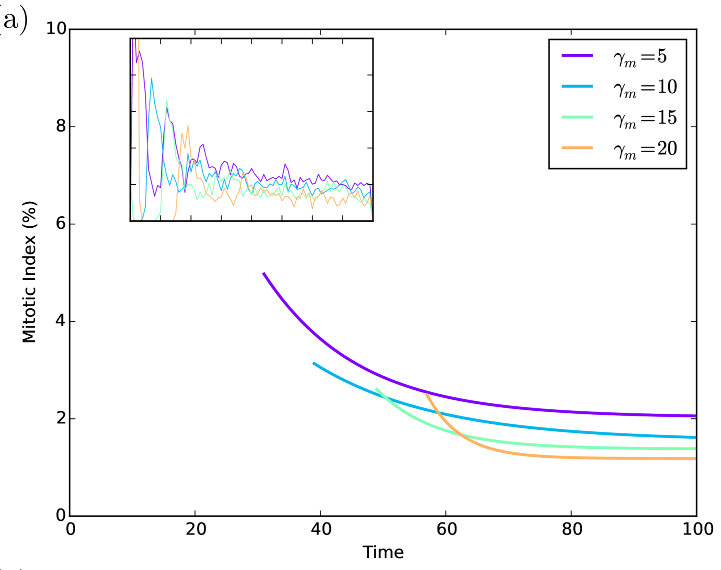
(c)


Figure 6.5: **These graphs were created with simulations with confining walls placed roughly one cell diameter apart. (a) Shows the average mitotic index of ten simulations. Inset shows the mitotic index for the whole simulation time at $\gamma_{ext} = 10$. (b) Shows the same for different values of γ_{ext}. Inset shows the raw data (average of ten simulations) over the same time periods that were used for the fits. The fits at $\gamma_{ext} = 15$ and $\gamma_{ext} = 20$ show a different behaviour compared to lower γ_{ext}'s. (c) shows the cross-sections of some cells in confinement. The right most daughter cells formed almost perfectly parallel to the confining walls.**

(a)



(b)

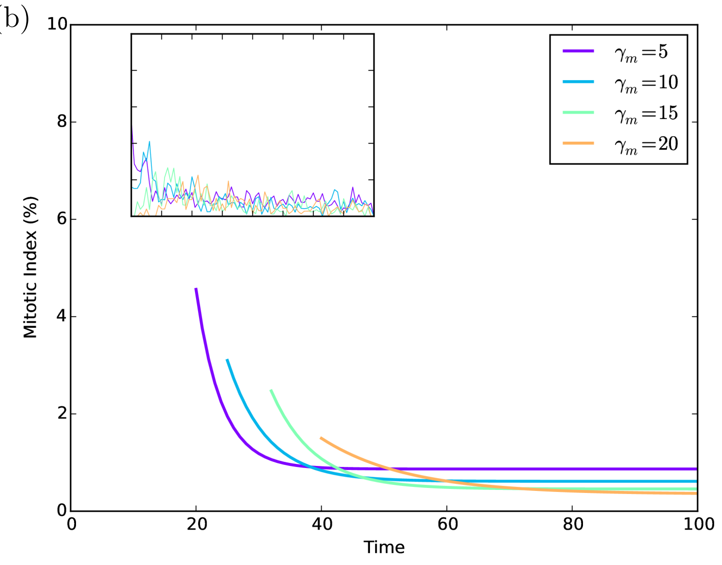
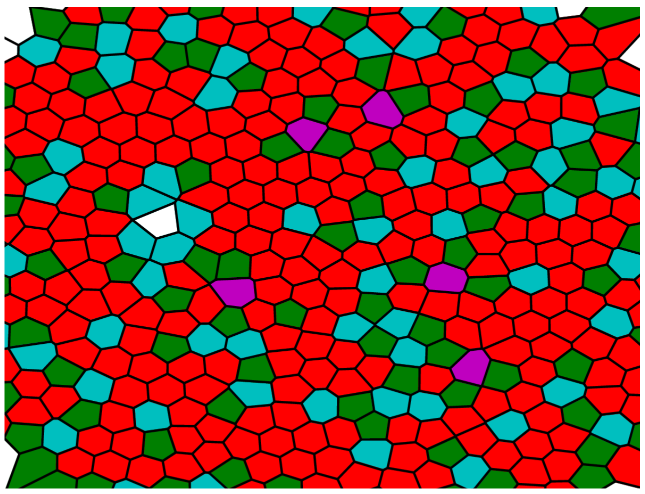


Figure 6.6: **Here the affects of γ_m on the mitotic indices of free and confined systems are explored. (a) Free 3D cells are placed in simulations with varying levels of γ_m, this parameter can affect mitotic index dramatically. Similar exponential decay is observed but at different rates and to lower steady states with higher γ_m(b) Mitotic index of 3D cells confined between walls. γ_m affects the confined cells just like the unconfined cells.**

(a)



(b)

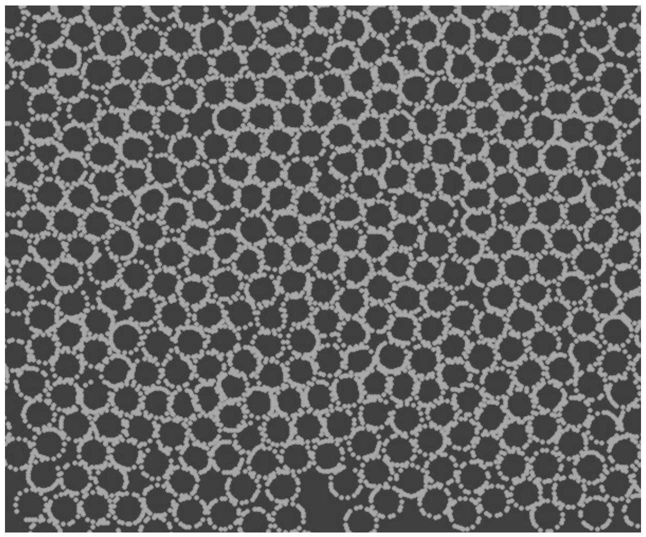
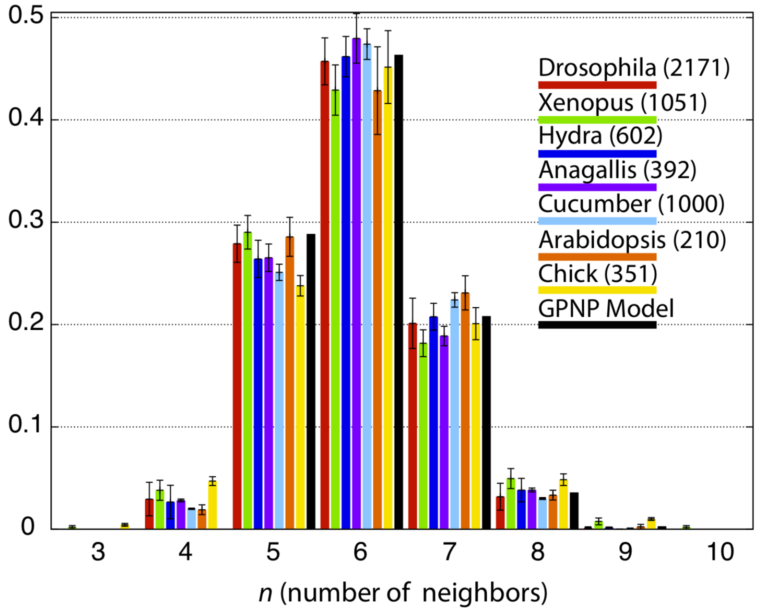


Figure 6.7: **These figures show a two dimensional cross section done halfway in between the confining walls. (a) Voronoi tessellation that was used to determine the number of neighbours each cell has. Cells with four neighbours are coloured white, pentagonal cells are coloured green, hexagonal cells are coloured red, heptagonal cyan, and octagonal magenta. (b) This figures shows the cross section of the mass points that make up the cells. This images were generated by taking the projection of all points on a plan parallel to the walls to simulate an epithelium.**

(a)


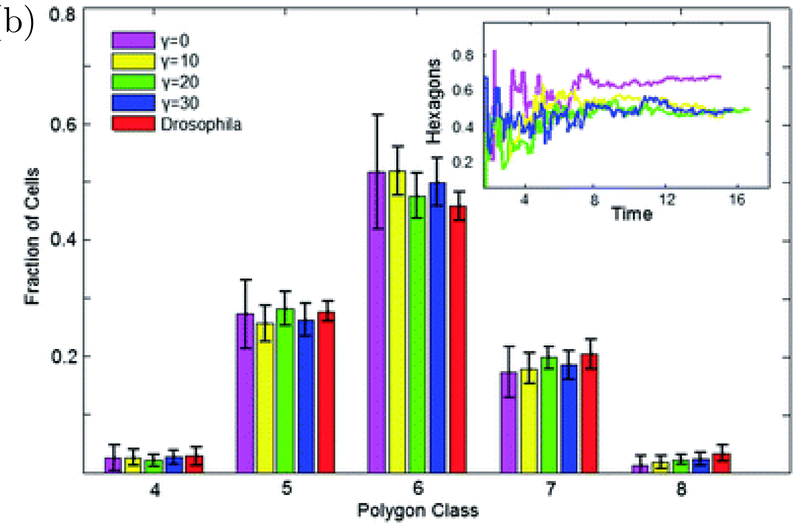
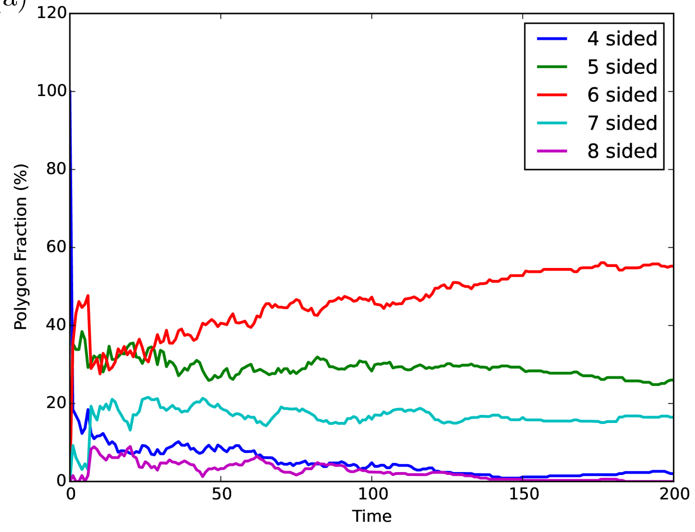
(b)


Figure 6.8: (a)This figure the fraction of cell polygon types for a number of species. The number in brackets is the number of cells counted, and the GPNP model refers to the results of a topological model (see Section 3.2.3) that was used by Gibson et al. [151]. Taken from[204], © 2011 Sandersius et al., CC license. (b) Similar measurement using the 2D model done by Mkrtchyan et al. [15] at various values of γ_{ext}, with comparison with *Drosophila* [158].

(a)



(b)

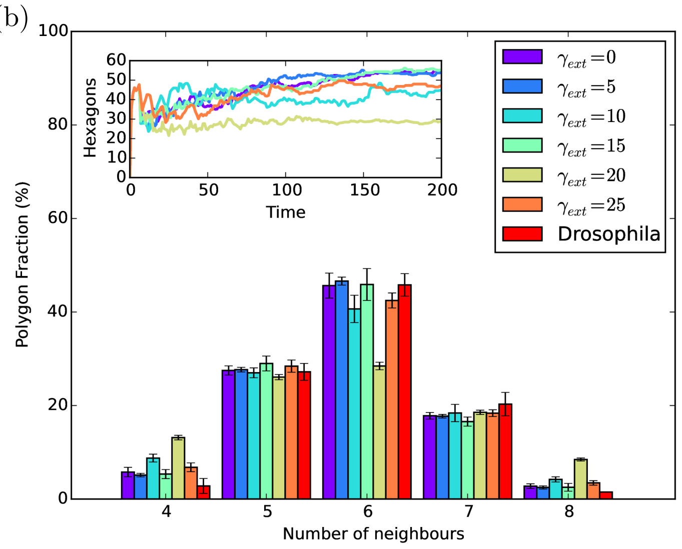


Figure 6.9: **All measurements were done with $\gamma_m = 5$. (a) The change in fraction of polygons over the simulation time of $\gamma_{ext} = 15$. Cell growth was halted at $t = 100T^{Div}$, the system then relaxes by rearranging cell distribution towards more favourable packing distribution. (b) The packing distribution at different γ_{ext} compared to a experimental measurement [158]. Inset shows the change in fraction of hexagons over simulation time, cell division was halted at $t = 100T^{Div}$. Error bars are the variation in packing from $t = 100T^{Div}$ and onward. Most values of γ_{ext}, except $\gamma_{ext} = 20$ produce distributions comparable to the results shown in Figure 6.8.**

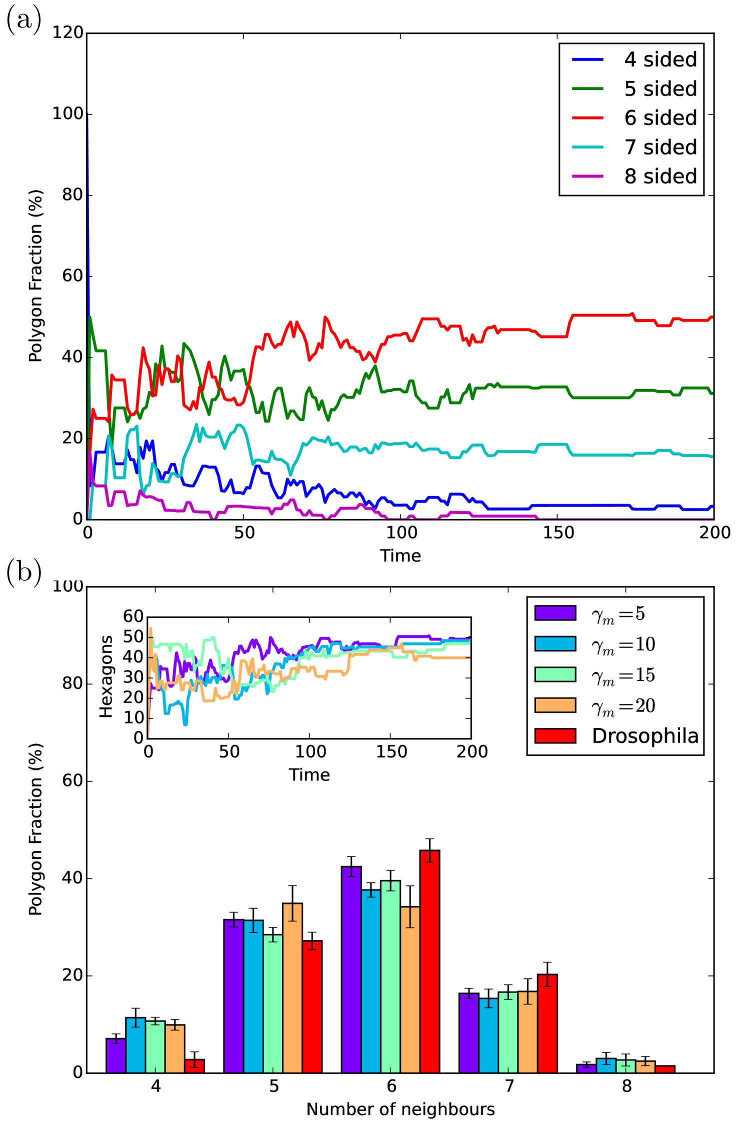


Figure 6.10: γ_m **has a different effect on packing. All of these measurements were done at $\gamma_{ext} = 2$ to minimize its effects while still maintaining a modicum of intercellular adhesion. (a) The evolution of the fractions of different polygon types over the simulation time, growth was halted at $t = 100$. (b) The proportions of different polygon types measured with various γ_m, inset shows evolution of hexagonal cell fraction over time. This figure indicates that while γ_m does affect the distribution, it is γ_{ext} that is instrumental in this process. γ_m slows all cell motion down, while γ_{ext} only hinders rearrangement.**

Chapter 7

Conclusions

The goal of this project was to develop a novel method of cell modelling. Some of the existing models were described in Chapter 3, and the problems associated with them discussed. It was found that these models, while excelling in certain situations, were insufficient in replicating cell behaviour fully. Due to the complexity of cellular structure, it is difficult to completely simulate cellular behaviour. This problem is compounded by limitations in computational power. Therefore, the models that have been developed thus far, were limited by necessity rather than lack of interest or capability. The goal of this project was to develop a Molecular Dynamics (MD) based method to simulate cells in a more realistic manner than the techniques used in the field today while taking advantage of the novel computational techniques in the area of scientific computing.

With the use of GPU acceleration, a novel method of modelling cells as three dimensional meshes based on previous work by Karttunen, Åström [14], and Mkrtchyan [15]. This method is described was described in some detail in Chapters 5 and 4. The code was first developed by Jan Westerholm from the Åbo Akademi University, then modified to replicate cell behaviour more accurately. GPU acceleration, even though not fully optimized yet, has made the run times much more manageable without adding much to resource cost. In this respect, it is the opinion of this author, that the GPU acceleration was successful and further optimization should be pursued.

Once the simulation was developed, it needed to be validated. After validation, the program can be used to study new cells in different ways, and even to make predictions of cell behaviour. The first step of validation was to compare to the results produced in 2D by Mkrtchyan et al. in their work [15]. To that end, the behaviour of Mitotic index was studied in Chapter 6. It was found that the mitotic indices produced behaved as expected.

The simulated cells were also placed in between confining walls to emulate epithelia and to study how the cells packed into two dimensional tissue. It was found that that the simulation produced expected packing distributions. There is also still more work to be done towards refining the model further and studying other aspects of cell behaviour.

7.1 Future Plans

The next steps of this project can be divided into three broad areas: Technical improvements, further validation, and model refinement. On the technical side progress will be made on two fronts. There is still some more work that could be done to improve performance by optimizing the GPU kernels. This will entail algorithmic improvements in the implementation of MD on GPUs correctly. Some further research will be done into the hardware and software nuances of NVIDIA devices. It is also highly desirable to use other devices, such as multicore or Multiple Integrated Core (MIC) devices, for acceleration. GPUs are of course amazing devices, but that is no need to shun other perfectly fine high performance hardware. This will be done with OpenCL which has the added advantage of being supported by most processors in the in the market.

Further validation and model refinement are related. Firstly, the results in Chapter 6 only compare to two dimensional experimental results, the MD method of simulating cells has to be modified to reproduce cell behaviour in three dimensions. This will be begin by adding cell diffusion to the model. The aim in that case will be to study cell motilities. After that, the next target will be various kinds of cell migration; diffusion directed by some potential such as a concentration gradient(chemotaxis), changes in adhesion strength of the Extracellular Matrix (ECM) (haptotaxis). So far only single types of cells have been discussed. This model also allows for modelling different types of cells, which leads to various interesting phenomena such as cell sorting [135] and morphogenesis [134]. The difference in cells may be introduced as initial conditions, or changes may be introduce in the model itself that would affect cell behaviour. The cell division done for this thesis was random symmetric division, but we saw in Chapter 2 that cells may divide different ways to lead to different daughter cells. Asymmetric, and division by the longest axis rule will also be implemented and their affects on the parameters measured in Chapter 6 will be tested and validated.

References

[1] D Boal. *Mechancis of the Cell*. 1st ed. Cambridge University Press, 2002.

[2] H. Lodish. *Molecular Cell Biology*. Macmillan, 2008.

[3] T. Rozario and D. W. DeSimone. “The extracellular matrix in development and morphogenesis: A dynamic view”. In: *Dev. Biol.* 341.1 (2010), pp. 126–140. DOI: 10.1016/j.ydbio.2009.10.026.

[4] G. K. Gittes. “Developmental biology of the pancreas: A comprehensive review”. In: *Dev. Biol.* 326.1 (2009), pp. 4–35. DOI: 10.1016/j.ydbio.2008.10.024.

[5] T. J. Mitchison and E. D. Salmon. “Mitosis: a history of division.” eng. In: *Nat. Cell Biol.* 3.1 (2001), E17–E21. DOI: 10.1038/35050656.

[6] P. Bheda and R. Schneider. “Epigenetics reloaded: the single-cell revolution”. In: *Trends Cell Biol.* 24.11 (2014), pp. 712–723. DOI: 10.1016/j.tcb.2014.08.010.

[7] D. E. Koshland Jr. “SPECIAL ESSAY: The Seven Pillars of Life”. In: *Science* 295.5563 (2002), pp. 2215–2216. DOI: 10.1126/science.1068489.

[8] S. A. Adcock and J. A. McCammon. “Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins”. In: *Chem. Rev.* 106.5 (2006), pp. 1589–1615. DOI: 10.1021/cr040426m.

[9] J. J. Janke, W. F. D. Bennett, and D. P. Tieleman. “Oleic Acid Phase Behavior from Molecular Dynamics Simulations”. In: *Langmuir* 30.35 (2014), pp. 10661–10667. DOI: 10.1021/la501962n.

[10] W. D. Bennett and D. P. Tieleman. “Computer simulations of lipid membrane domains”. In: *Biochim. Biophys. Acta - Biomembranes* 1828.8 (2013), pp. 1765–1776. DOI: 10.1016/j.bbamem.2013.03.004.

[11] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten. “Accelerating Molecular Modeling Applications with Graphics Processors”. In: *J. Comput. Chem.* 28.16 (Dec. 2007), pp. 2618–2640. DOI: 10.1002/jcc.20829.

[12] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. “A Survey of General-Purpose Computation on Graphics Hardware”. In: *Comput. Graph. Forum* 26.1 (Mar. 2007), pp. 80–113. DOI: 10.1111/j.1467-8659.2007.01012.x.

[13] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips. “GPU Computing”. In: *Proc. IEEE* 96.5 (May 2008), pp. 879–899. DOI: 10.1109/JPROC.2008.917757.

[14] J. A. Åström and M. Karttunen. “Cell aggregation: Packing soft grains”. In: *Physical Review E* 73 (6 2006), p. 062301. DOI: 10.1103/PhysRevE.73.062301.

[15] A. Mkrtchyan, J. Åström, and M. Karttunen. “A new model for cell division and migration with spontaneous topology changes”. In: *Soft Matter* 10.24 (2014), pp. 4332–4339. DOI: 10.1039/c4sm00489b.

[16] T. Pollard, W. Earnshaw, and J. Lippincott-Schartz. *Cell Biology*. Ed. by W. Schmidt. 2nd ed. Saunders Elsevier, 2008.

[17] G. M. Cooper and R. E. Hausman. *The Cell*. 5th ed. Sinauer Associates, 2000.

[18] D. Voet, J. G. Voet, and C. W. Pratt. *Fundamentals of Biochemistry: Life at the Molecular Level*. Wiley, 2008.

[19] Q. Wen and P. A. Janmey. “Polymer physics of the cytoskeleton”. In: *Curr. Opin. Solid. St. M.* 15.5 (2011), 177–182. DOI: 10.1016/j.cossms.2011.05.002.

[20] H. Herrmann, H. Bär, L. Kreplak, S. V. Strelkov, and U. Aebi. “Intermediate filaments: from cell architecture to nanomechanics”. In: *Nat. Rev. Mol. Cell Biol.* 8.7 (2007), pp. 562–573. DOI: 10.1038/nrm2197.

[21] J. I. Castrillo, L. A. Zeef, D. C. Hoyle, N. Zhang, A. Hayes, D. C. J. Gardner, M. J. Cornell, J. Petty, L. Hakes, L. Wardleworth, B. Rash, M. Brown, W. B. Dunn, D. Broadhurst, K. O’Donoghue, S. S. Hester, T. P. J. Dunkley, S. R. Hart, N. Swainston, P. Li, S. J. Gaskell, N. W. Paton, K. S. Lilley, D. B. Kell, and S. G. Oliver. “Growth control of the eukaryote cell: a systems biology study in yeast.” eng. In: *J. Biol.* 6.2 (2007), p. 4. DOI: 10.1186/jbiol54.

[22] L. H. Hartwell. “Cell division from a genetic perspective.” eng. In: *J. Cell Biol.* 77.3 (1978), pp. 627–637.

[23] B. Guo, Q. Liang, L. Li, Z. Hu, F. Wu, P. Zhang, Y. Ma, B. Zhao, A. L. Kovcs, Z. Zhang, and et al. "O-GlcNAc-modification of SNAP-29 regulates autophagosome maturation". In: *Nat. Cell Biol.* (2014). DOI: 10.1038/ncb3066.

[24] S. J. Silverman, A. A. Petti, N. Slavov, L. Parsons, R. Briehof, S. Y. Thiberge, D. Zenklusen, S. J. Gandhi, D. R. Larson, R. H. Singer, and et al. "Metabolic cycling in single yeast cells from unsynchronized steady-state populations limited on glucose or phosphate". In: *Proc. Natl. Acad. Sci. U. S. A.* 107.15 (2010), pp. 6946–6951. DOI: 10.1073/pnas.1002422107.

[25] A. D. Chalmers. "Oriented cell divisions asymmetrically segregate aPKC and generate cell fate diversity in the early Xenopus embryo". In: *Development* 130.12 (2003), 26572668. DOI: 10.1242/dev.00490.

[26] H. Zhang and Z. Z. Wang. "Mechanisms that mediate stem cell self-renewal and differentiation." eng. In: *J. Cell. Biochem.* 103.3 (2008), pp. 709–718. DOI: 10.1002/jcb.21460.

[27] J. A. Knoblich. "Assymetric Cell Division During Animal Development". In: *Nat. Rev. Mol. Cell Biol.* 2.1 (2001), pp. 11–20. DOI: 10.1038/35048085.

[28] R. A. Neumuller and J. A. Knoblich. "Dividing cellular asymmetry: asymmetric cell division and its implications for stem cells and cancer". In: *Genes & Development* 23.23 (2009), pp. 2675–2699. DOI: 10.1101/gad.1850809.

[29] Y. Miyaoka and A. Miyajima. "To divide or not to divide: revisiting liver regeneration". In: *Cell Div* 8.1 (2013), p. 8. DOI: 10.1186/1747-1028-8-8.

[30] T. Gillies and C. Cabernard. "Cell Division Orientation in Animals". In: *Curr. Biol.* 21.15 (Aug. 2011), pp. 599–609. DOI: 10.1016/j.cub.2011.06.055.

[31] D. Hanahan and R. A. Weinberg. "Hallmarks of cancer: the next generation." eng. In: *Cell* 144.5 (2011), pp. 646–674. DOI: 10.1016/j.cell.2011.02.013.

[32] H. R. Horvitz and I. Herskowitz. "Mechanisms of asymmetric cell division: two Bs or not two Bs, that is the question." eng. In: *Cell* 68.2 (1992), pp. 237–255.

[33] M. Tian, J. R. Neil, and W. P. Schiemann. "Transforming growth factor-β and the hallmarks of cancer." eng. In: *Cell. Signal.* 23.6 (2011), pp. 951–962. DOI: 10.1016/j.cellsig.2010.10.015.

[34] W. Flemming. *Zellsubstanz, kern und zelltheilung.* Leipzig, F.C.W Vogel, 1882.

[35] M. P. Stewart, J. Helenius, Y. Toyoda, S. P. Ramanathan, D. J. Muller, and A. A. Hyman. "Hydrostatic pressure and the actomyosin cortex drive mitotic cell rounding". In: *Nature* 469.7329 (2011), 226230. DOI: 10.1038/nature09642.

[36] T. Lecuit and P.-F. Lenne. “Cell surface mechanics and the control of cell shape, tissue patterns and morphogenesis”. In: *Nat. Rev. Mol. Cell Biol.* 8.8 (2007), pp. 633–644. DOI: 10.1038/nrm2222.

[37] C. Roubinet, P. T. Tran, and M. Piel. “Common mechanisms regulating cell cortex properties during cell division and cell migration.” eng. In: *Cytoskeleton (Hoboken)* 69.11 (2012), pp. 957–972. DOI: 10.1002/cm.21086.

[38] S. W. Grill. “Forced to Be Unequal”. In: *Science* 330.6004 (2010), 597598. DOI: 10.1126/science.1198343.

[39] Y. Imoto, Y. Yoshida, F. Yagisawa, H. Kuroiwa, and T. Kuroiwa. “The cell cycle, including the mitotic cycle and organelle division cycles, as revealed by cytological observations”. In: *Microscopy* 60.suppl 1 (2011), S117S136. DOI: 10.1093/jmicro/dfr034.

[40] P. Kunda and B. Baum. “The actin cytoskeleton in spindle assembly and positioning.” eng. In: *Trends Cell Biol.* 19.4 (2009), pp. 174–179. DOI: 10.1016/j.tcb.2009.01.006.

[41] M. T. Tyree and H. T. Hammel. “The Measurement of the Turgor Pressure and the Water Relations of Plants by the Pressure-bomb Technique”. In: *J. Exp. Bot.* 23.1 (1972), 267282. DOI: 10.1093/jxb/23.1.267.

[42] N. Minc, D. Burgess, and F. Chang. “Influence of Cell Geometry on Division-Plane Positioning”. In: *Cell* 144.3 (2011), pp. 414–426. DOI: 10.1016/j.cell.2011.01.016.

[43] R. Rappaport. *Cytokinesis in Animal Cells.* Cambridge University Press, 1996.

[44] J. M. Scholey, I. Brust-Mascher, and A. Mogilner. “Cell division.” eng. In: *Nature* 422.6933 (2003), pp. 746–752. DOI: 10.1038/nature01599.

[45] O. Hertwig. *Das Problem der Befruchtung und der Isotropie des Eies, eine Theorie der Vererbung.* Jena, 1884.

[46] D. St Johnston and J. Ahringer. “Cell Polarity in Eggs and Epithelia: Parallels and Diversity”. In: *Cell* 141.5 (2010), pp. 757–774. DOI: 10.1016/j.cell.2010.05.011.

[47] Y. N. Jan and L. Y. Jan. “Assymetric cell division”. In: *Nature* 392.6678 (1998), pp. 775–778. DOI: 10.1038/33854.

[48] J. A. Knoblich. “Mechanisms of Asymmetric Stem Cell Division”. In: *Cell* 132.4 (2008), pp. 583–597. DOI: 10.1016/j.cell.2008.02.007.

[49] A. Mkrtchyan. “A Single Cell Based Model for Cell Divisions with Spontaneous topology changes”. PhD thesis. 2013.

[50] B. M. Gumbiner. "Cell adhesion: the molecular basis of tissue architecture and morphogenesis." eng. In: *Cell* 84.3 (1996), pp. 345–357.

[51] G. Bell, M. Dembo, and P. Bongrand. "Cell adhesion. Competition between nonspecific repulsion and specific bonding". In: *Biophys. J.* 45.6 (1984), 10511064. DOI: 10.1016/s0006-3495(84)84252-6.

[52] C. D. Buckley, G. E. Rainger, P. F. Bradfield, G. B. Nash, and D. L. Simmons. "Cell adhesion: More than just glue (Review)". In: *Mol. Membr. Biol.* 15.4 (1998), 167176. DOI: 10.3109/09687689709044318.

[53] J. T. Parsons, A. R. Horwitz, and M. A. Schwartz. "Cell adhesion: integrating cytoskeletal dynamics and cellular tension." eng. In: *Nat. Rev. Mol. Cell Biol.* 11.9 (2010), pp. 633–643. DOI: 10.1038/nrm2957.

[54] J. Käfer, T. Hayashi, A. F. M. Marée, R. W. Carthew, and F. Graner. "Cell adhesion and cortex contractility determine cell patterning in the Drosophilaretina". In: *Proc. Natl. Acad. Sci. U. S. A.* 104.47 (2007), 1854918554. DOI: 10.1073/pnas.0704235104.

[55] P. DiMilla, K. Barbee, and D. Lauffenburger. "Mathematical model for the effects of adhesion and mechanics on cell migration speed". In: *Biophys. J.* 60.1 (1991), 1537. DOI: 10.1016/s0006-3495(91)82027-6.

[56] T. Hayashi and R. W. Carthew. "Surface mechanics mediate pattern formation in the developing retina". In: *Nature* 431.7009 (2004), 647652. DOI: 10.1038/nature02952.

[57] H. Byrne and D. Drasdo. "Individual-based and continuum models of growing cell populations: a comparison". In: *J. Math. Biol.* 58.4-5 (2008), 657687. DOI: 10.1007/s00285-008-0212-0.

[58] D. Stamenović and D. E. Ingber. "Models of cytoskeletal mechanics of adherent cells." eng. In: *Biomech. Model. Mechanobiol.* 1.1 (2002), pp. 95–108. DOI: 10.1007/s10237-002-0009-9.

[59] B. G. Sengers, M. Taylor, C. P. Please, and R. O. C. Oreffo. "Computational modelling of cell spreading and tissue regeneration in porous scaffolds." eng. In: *Biomaterials* 28.10 (2007), pp. 1926–1940. DOI: 10.1016/j.biomaterials.2006.12.008.

[60] A. Voss-Böhme. "Multi-Scale Modeling in Morphogenesis: A Critical Analysis of the Cellular Potts Model". In: *PLoS ONE* 7.9 (2012). Ed. by C. M. Aegerter, e42852. DOI: 10.1371/journal.pone.0042852.

[61] A. B. Patel, W. T. Gibson, M. C. Gibson, and R. Nagpal. "Modeling and Inferring Cleavage Patterns in Proliferating Epithelia". In: *PLoS Comput. Biol.* 5.6 (2009). Ed. by J. Axelrod, e1000412. DOI: 10.1371/journal.pcbi.1000412.

[62] A. J. Kabla. "Collective cell migration: leadership, invasion and segregation". en. In: *Journal of The Royal Society Interface* 9.77 (Dec. 2012), pp. 3268–3278. DOI: 10.1098/rsif.2012.0448.

[63] T. J. Newman. "Modeling multicellular systems using subcellular elements." eng. In: *Math. Biosci. Eng.* 2.3 (2005), pp. 613–624.

[64] T. Beyer and M. Meyer-Hermann. "Multiscale modeling of cell mechanics and tissue organization". In: 28.2 (2009), 3845. DOI: 10.1109/memb.2009.931790.

[65] J. A. Anderson, C. D. Lorenz, and A. Travesset. "General purpose molecular dynamics simulations fully implemented on graphics processing units". In: *J. Comput. Phys.* 227.10 (May 2008), pp. 5342–5359. DOI: 10.1016/j.jcp.2008.01.047.

[66] S. Plimpton. "Fast Parallel Algorithms for Short-Range Molecular Dynamics". In: *J. Comput. Phys.* 117.1 (Mar. 1995), pp. 1–19. DOI: 10.1006/jcph.1995.1039.

[67] W Smith and T. R. Forester. "DL_POLY_2.0: a general-purpose parallel molecular dynamics simulation package." In: *J. Mol. Graph.* 14.3 (June 1996), pp. 136–41.

[68] H. Berendsen, D. van der Spoel, and R. van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation". In: *Comput. Phys. Commun.* 91.1-3 (Sept. 1995), pp. 43–56. DOI: 10.1016/0010-4655(95)00042-E.

[69] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. "Scalable molecular dynamics with NAMD." In: *J. Comput. Chem.* 26.16 (Dec. 2005), pp. 1781–802. DOI: 10.1002/jcc.20289.

[70] H. Limbach, A. Arnold, B. Mann, and C. Holm. "ESPResSoan extensible simulation package for research on soft matter systems". In: *Comput. Phys. Commun.* 174.9 (May 2006), pp. 704–727. DOI: 10.1016/j.cpc.2005.10.005.

[71] K. Hinsen. "The molecular modeling toolkit: A new approach to molecular simulations". In: *J. Comput. Chem.* 21.2 (Jan. 2000), pp. 79–85. DOI: 10.1002/(SICI)1096-987X(20000130)21:2<79::AID-JCC1>3.0.CO;2-B.

[72] B. J. Alder and T. E. Wainwright. "Phase Transition for a Hard Sphere System". In: *J. Chem. Phys.* 27.5 (1957), p. 1208. DOI: 10.1063/1.1743957.

[73] A. Rahman. "Correlations in the Motion of Atoms in Liquid Argon". In: *Phys. Rev.* 136.2A (Oct. 1964), A405–A411. DOI: 10.1103/PhysRev.136.A405.

[74] R. Allen, F. de Wette, and A. Rahman. "Calculation of Dynamical Surface Properties of Noble-Gas Crystals. II. Molecular Dynamics". In: *Phys. Rev.* 179.3 (1969), 887891. DOI: 10.1103/physrev.179.887.

[75] B. J. Alder and T. E. Wainwright. "Studies in Molecular Dynamics. VIII. The Transport Coefficients for a Hard-Sphere Fluid". In: *J. Chem. Phys.* 53.10 (1970), p. 3813. DOI: 10.1063/1.1673845.

[76] J. A. McCammon, B. R. Gelin, and M. Karplus. "Dynamics of folded proteins". In: *Nature* 267.5612 (1977), 585590. DOI: 10.1038/267585a0.

[77] W. D. Bennett, A. W. Chen, S. Donnini, G. Groenhof, and D. P. Tieleman. "Constant pH simulations with the coarse-grained MARTINI model – Application to oleic acid aggregates". In: *Can. J. Chem.* 91.9 (Apr. 2013), pp. 839–846. DOI: 10.1139/cjc-2013-0010.

[78] N. Nisoh, M. Karttunen, L. Monticelli, and J. Wong-ekkabut. "Lipid monolayer disruption caused by aggregated carbon nanoparticles". In: *RSC Adv.* 5.15 (2015), 1167611685. DOI: 10.1039/c4ra17006g.

[79] B. Liu, M. I. Hoopes, and M. Karttunen. "Molecular Dynamics Simulations of DPPC/CTAB Monolayers at the Air/Water Interface". In: *J. Phys. Chem. B* 118.40 (2014), 1172311737. DOI: 10.1021/jp5050892.

[80] W. Bennett, N. Sapay, and D. Tieleman. "Atomistic Simulations of Pore Formation and Closure in Lipid Bilayers". In: *Biophys. J.* 106.1 (2014), 210219. DOI: 10.1016/j.bpj.2013.11.4486.

[81] T. N. Do, W.-Y. Choy, and M. Karttunen. "Accelerating the Conformational Sampling of Intrinsically Disordered Proteins". In: *J. Chem. Theory Comput.* 10.11 (2014), 50815094. DOI: 10.1021/ct5004803.

[82] A. H. Elcock. "Molecular simulations of diffusion and association in multimacromolecular systems." eng. In: *Methods Enzymol.* 383 (2004), pp. 166–198. DOI: 10.1016/S0076-6879(04)83008-8.

[83] A. Warshel. "Computer simulations of enzyme catalysis: methods, progress, and insights." eng. In: *Annu. Rev. Biophys. Biomol. Struct.* 32 (2003), pp. 425–443. DOI: 10.1146/annurev.biophys.32.110601.141807.

[84] W. F. van Gunsteren and H. J. C. Berendsen. "Computer Simulation of Molecular Dynamics: Methodology, Applications, and Perspectives in Chemistry". In: *Angew. Chem. Int. Ed. Engl.* 29.9 (1990), pp. 992–1023. DOI: 10.1002/anie.199009921.

[85] J.-P. Ryckaert and A. Bellemans. “Molecular dynamics of liquid alkanes”. In: *Faraday Discuss. Chem. Soc.* 66 (1978), p. 95. DOI: 10.1039/dc9786600095.

[86] J.-P. Ryckaert, G. Ciccotti, and H. J. Berendsen. “Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes”. In: *J. Comput. Phys.* 23.3 (1977), 327341. DOI: 10.1016/0021-9991(77)90098-5.

[87] D. Frenkel, B. Smit, and M. A. Ratner. “Understanding Molecular Simulation: From Algorithms to Applications”. In: *Phys. Today* 50.7 (1997), p. 66. DOI: 10.1063/1.881812.

[88] T. Darden, D. York, and L. Pedersen. “Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems”. In: *J. Chem. Phys.* 98.12 (1993), p. 10089. DOI: 10.1063/1.464397.

[89] P. P. Ewald. “Die Berechnung optischer und elektrostatischer Gitterpotentiale”. In: *Annalen der Physik* 369.3 (1921), pp. 253–287. DOI: 10.1002/andp.19213690304.

[90] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. “Implementing molecular dynamics on hybrid high performance computers Particleparticle particle-mesh”. In: *Comput. Phys. Commun.* 183.3 (Mar. 2012), pp. 449–459. DOI: 10.1016/j.cpc.2011.10.012.

[91] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen. “A smooth particle mesh Ewald method”. In: *J. Chem. Phys.* 103.19 (1995), p. 8577. DOI: 10.1063/1.470117.

[92] R. W. Hockney and K. W. Eastwood. *Computer Simulations Using Particles.* CRC Press, 2010.

[93] M. Karttunen, J. Rottler, I. Vattulainen, and C. Sagui. “Chapter 2 Electrostatics in Biomolecular Simulations: Where Are We Now and Where Are We Heading?” In: *Curr. Top. Membr.* (2008), pp. 49–89. DOI: 10.1016/s1063-5823(08)00002-1.

[94] G. A. Cisneros, M. Karttunen, P. Ren, and C. Sagui. “Classical Electrostatics for Biomolecular Simulations”. In: *Chem. Rev.* 114.1 (2014), pp. 779–814. DOI: 10.1021/cr300461d.

[95] T. Murtola, A. Bunker, I. Vattulainen, M. Deserno, and M. Karttunen. “Multiscale modeling of emergent materials: biological and soft matter”. In: *Phys. Chem. Chem. Phys.* 11.12 (2009), p. 1869. DOI: 10.1039/b818051b.

[96] D. Tieleman, S. Marrink, and H. Berendsen. “A computer perspective of membranes: molecular dynamics studies of lipid bilayer systems”. In: *Biochimica et Biophysica Acta (BBA) - Reviews on Biomembranes* 1331.3 (1997), 235270. DOI: 10.1016/s0304-4157(97)00008-7.

[97] L. Monticelli, S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, and S.-J. Marrink. “The MARTINI Coarse-Grained Force Field: Extension to Proteins”. In: *J. Chem. Theory Comput.* 4.5 (2008), 819834. DOI: 10.1021/ct700324x.

[98] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. de Vries. “The MARTINI Force Field: Coarse Grained Model for Biomolecular Simulations”. In: *J. Phys. Chem. B* 111.27 (2007), 78127824. DOI: 10.1021/jp071097f.

[99] Y. Wang, W. Jiang, T. Yan, and G. A. Voth. “Understanding ionic liquids through atomistic and coarse-grained molecular dynamics simulations.” In: *Acc. Chem. Res.* 40.11 (Nov. 2007), pp. 1193–9. DOI: 10.1021/ar700160p.

[100] R. Rudd and J. Broughton. “Coarse-grained molecular dynamics and the atomic limit of finite elements”. In: *Phys. Rev. B* 58.10 (Sept. 1998), R5893–R5896. DOI: 10.1103/PhysRevB.58.R5893.

[101] L. Verlet. “Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules”. In: *Phys. Rev.* 159.1 (1967), 98103. DOI: 10.1103/physrev.159.98.

[102] E. Hairer, C. Lubich, and G. Wanner. “Geometric numerical integration illustrated by the StrmerVerlet method”. In: *Acta Numerica* 12 (2003), 399450. DOI: 10.1017/s0962492902000144.

[103] I. Geudens and H. Gerhardt. “Coordinating cell behaviour during blood vessel formation”. In: *Development* 138.21 (2011), 45694583. DOI: 10.1242/dev.062323.

[104] K. Riento and A. J. Ridley. “Rocks: multifunctional kinases in cell behaviour”. In: *Nat. Rev. Mol. Cell Biol.* 4.6 (2003), 446456. DOI: 10.1038/nrm1128.

[105] M. C. Frame. “Src in cancer: deregulation and consequences for cell behaviour”. In: *Biochimica et Biophysica Acta (BBA) - Reviews on Cancer* 1602.2 (2002), 114130. DOI: 10.1016/s0304-419x(02)00040-9.

[106] R. M. Merks and J. A. Glazier. “A cell-centered approach to developmental biology”. In: *Physica A: Statistical Mechanics and its Applications* 352.1 (2005), 113130. DOI: 10.1016/j.physa.2004.12.028.

[107] S. Turner. “Using cell potential energy to model the dynamics of adhesive biological cells”. In: *Phys. Rev. E* 71 (4 2005), p. 041903. DOI: 10.1103/PhysRevE.71.041903.

[108] N. Bellomo and B. Carbonaro. “Toward a mathematical theory of living systems focusing on developmental biology and evolution: A review and perspectives”. In: *Phys. Life Rev.* 8.1 (2011), 118. DOI: 10.1016/j.plrev.2010.12.001.

[109] N. NDri, W. Shyy, and R. Tran-Son-Tay. “Computational Modeling of Cell Adhesion and Movement Using a Continuum-Kinetics Approach”. In: *Biophys. J.* 85.4 (2003), 22732286. DOI: 10.1016/s0006-3495(03)74652-9.

[110] D. A. Baltazar and A. Friedman. *Models of Cellular Regulation*. Oxford University Press (OUP), 2008.

[111] J. C. Sible and J. J. Tyson. “Mathematical modeling as a tool for investigating cell cycle control networks”. In: *Methods* 41.2 (2007), pp. 238–247. DOI: 10.1016/j.ymeth.2006.08.003.

[112] J. A. D. Wattis, B. O’Malley, H. Blackburn, L. Pickersgill, J. Panovska, H. M. Byrne, and K. G. Jackson. “Mathematical model for low density lipoprotein (LDL) endocytosis by hepatocytes.” eng. In: *Bull. Math. Biol.* 70.8 (2008), pp. 2303–2333. DOI: 10.1007/s11538-008-9347-9.

[113] R. Robeva, R. Davies, T. Hodge, and A. Enyedi. “Mathematical biology modules based on modern molecular biology and modern discrete mathematics.” eng. In: *CBE Life Sci. Educ.* 9.3 (2010), pp. 227–240. DOI: 10.1187/cbe.10-03-0019.

[114] N. F. Britton. *Essential mathematical biology*. Springer, 2003.

[115] J. D. Murray. *Mathematical Biology: I. An Introduction: Pt. 1 (Interdisciplinary Applied Mathematics)*. Springer New York, 2008.

[116] J. Schnute. “A Versatile Growth Model with Statistically Stable Parameters”. In: *Can. J. Fish. Aquat. Sci.* 38.9 (1981), 11281140. DOI: 10.1139/f81-153.

[117] M. H. Zwietering, I. Jongenburger, F. M. Rombouts, and K. van ’t Riet. “Modeling of the bacterial growth curve.” eng. In: *Appl. Environ. Microbiol.* 56.6 (1990), pp. 1875–1881.

[118] F. J. Richards. “A Flexible Growth Function for Empirical Use”. In: *J. Exp. Bot.* 10.2 (1959), pp. 290–301. DOI: 10.1093/jxb/10.2.290. eprint: http://jxb.oxfordjournals.org/content/10/2/290.full.pdf+html.

[119] G. C. Jahn, L. P. Almazan, and J. B. Pacia. “Effect of Nitrogen Fertilizer on the Intrinsic Rate of Increase of *Hysteroneura setariae* (Thomas) (Homoptera: Aphididae) on Rice (*Oryza sativa* L.)” In: *Environmental Entomology* 34.4 (2005), 938943. DOI: 10.1603/0046-225x-34.4.938.

[120] B. Gompertz. "On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies". In: *Philos. Trans. Roy. Soc. London* (1825), pp. 513–583.

[121] M. Ben Amar, C. Chatelain, and P. Ciarletta. "Contour Instabilities in Early Tumor Growth Models". In: *Phys. Rev. Lett.* 106.14 (2011). DOI: 10.1103/physrevlett.106.148101.

[122] J. Ranft, M. Basan, J. Elgeti, J.-F. Joanny, J. Prost, and F. Julicher. "Fluidization of tissues by cell division and apoptosis". en. In: *Proc. Natl. Acad. Sci. U. S. A.* 107.49 (Dec. 2010), pp. 20863–20868. DOI: 10.1073/pnas.1011086107.

[123] T. Bittig, O. Wartlick, A. Kicheva, M. González-Gaitn, and F. Jlicher. "Dynamics of anisotropic tissue growth". In: *New Journal of Physics* 10.6 (2008), p. 063001. DOI: 10.1088/1367-2630/10/6/063001.

[124] G. Schaller and M. Meyer-Hermann. "Multicellular tumor spheroid in an off-lattice Voronoi-Delaunay cell model". In: *Phys. Rev. E* 71.5 (2005). DOI: 10.1103/physreve.71.051910.

[125] G. Schaller and M. Meyer-Hermann. "Kinetic and dynamic Delaunay tetrahedralizations in three dimensions". In: *Comput. Phys. Commun.* 162.1 (2004), 923. DOI: 10.1016/j.cpc.2004.06.066.

[126] M. Meyer-Hermann. "Delaunay-Object-Dynamics: Cell Mechanics with a 3D Kinetic and Dynamic Weighted Delaunay-Triangulation". In: *Curr. Top. Dev. Biol.* (2008), 373399. DOI: 10.1016/s0070-2153(07)81013-1.

[127] D. T. Lee and B. J. Schachter. "Two algorithms for constructing a Delaunay triangulation". In: *International Journal of Computer & Information Science* 9.3 (1980), 219242. DOI: 10.1007/bf00977785.

[128] L. Paul Chew. "Constrained delaunay triangulations". In: *Algorithmica* 4.1-4 (1989), 97108. DOI: 10.1007/bf01553881.

[129] H. Edelsbrunner and N. R. Shah. "Incremental topological flipping works for regular triangulations". In: *Algorithmica* 15.3 (1996), 223241. DOI: 10.1007/bf01975867.

[130] B. Delaunay. "Sur la sphère vide. A la mémoire de Georges Voronoï". In: *Bulletin de l'Académie des Sciences de l'URSS* 6 (1934), pp. 793–800.

[131] F. Aurenhammer. "Power Diagrams: Properties, Algorithms and Applications". In: *SIAM Journal on Computing* 16.1 (1987), 7896. DOI: 10.1137/0216006.

[132] T. Beyer and M. Meyer-Hermann. "Modeling emergent tissue organization involving high-speed migrating cells in a flow equilibrium". en. In: *Phys .Rev. E* 76.2 (Aug. 2007), p. 021929. DOI: 10.1103/PhysRevE.76.021929.

[133] A. Szabó and R. M. H. Merks. "Cellular Potts Modeling of Tumor Growth, Tumor Invasion, and Tumor Evolution". In: *Front. Oncol.* 3 (2013). DOI: 10.3389/fonc.2013.00087.

[134] J. Glazier and F. Graner. "Simulation of the differential adhesion driven rearrangement of biological cells". In: *Phys. Rev. E* 47.3 (1993), pp. 2128–2154. DOI: 10.1103/physreve.47.2128.

[135] F. Graner and J. Glazier. "Simulation of biological cell sorting using a two-dimensional extended Potts model". In: *Phys. Rev. Lett.* 69.13 (1992), pp. 2013–2016. DOI: 10.1103/physrevlett.69.2013.

[136] R. B. Potts and C. Domb. "Some generalized order-disorder transformations". In: *Math. Proc. Camb. Phil. Soc.* 48.01 (1952), pp. 106–109. DOI: 10.1017/s0305004100027419.

[137] E. Ising. "Beitrag zur theorie des ferromagnetismus". In: *Zeitschrift für Physik A Hadrons and Nuclei* 31.1 (1925), pp. 253–258.

[138] F. Y. Wu. "The Potts model". In: *Rev. Mod. Phys.* 54.1 (1982), pp. 235–268. DOI: 10.1103/revmodphys.54.235.

[139] N. Chen, J. A. Glazier, J. A. Izaguirre, and M. S. Alber. "A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis". In: *Comput. Phys. Commun.* 176.11-12 (2007), pp. 670–681. DOI: 10.1016/j.cpc.2007.03.007.

[140] N. Metropolis and S. Ulam. "The Monte Carlo Method". In: *J. Am. Stat. Assoc.* 44.247 (1949), pp. 335–341. DOI: 10.1080/01621459.1949.10483310.

[141] N. Metropolis. "The beginning of the Monte Carlo method". In: *Los Alamos Science* 15.584 (1987), pp. 125–130.

[142] A. Köhn-Luque, W. de Back, J. Starru, A. Mattiotti, A. Deutsch, J. M. Prez-Pomares, and M. A. Herrero. "Early Embryonic Vascular Patterning by Matrix-Mediated Paracrine Signalling: A Mathematical Model Study". In: *PLoS ONE* 6.9 (2011). Ed. by B. Riley, e24175. DOI: 10.1371/journal.pone.0024175.

[143] M. Krieg, Y. Arboleda-Estudillo, P.-H. Puech, J. Kfer, F. Graner, D. J. Mller, and C.-P. Heisenberg. "Tensile forces govern germ-layer organization in zebrafish". In: *Nat. Cell Biol.* 10.4 (2008), 429436. DOI: 10.1038/ncb1705.

[144] J. Starruß, T. Bley, L. Søgaard-Andersen, and A. Deutsch. “A New Mechanism for Collective Migration in Myxococcus xanthus”. In: *J Stat Phys* 128.1-2 (2007), 269286. DOI: 10.1007/s10955-007-9298-9.

[145] R. M. H. Merks, E. D. Perryn, A. Shirinifard, and J. A. Glazier. “Contact-Inhibited Chemotaxis in De Novo and Sprouting Blood-Vessel Growth”. In: *PLoS Comput. Biol.* 4.9 (2008). Ed. by P. E. Bourne, e1000163. DOI: 10.1371/journal.pcbi.1000163.

[146] J. Käfer, P. Hogeweg, and A. F. M. Mare. “Moving Forward Moving Backward: Directional Sorting of Chemotactic Cells due to Size and Adhesion Differences”. In: *PLoS Comp Biol* 2.6 (2006), e56. DOI: 10.1371/journal.pcbi.0020056.

[147] R. J. Matela and R. J. Fletterick. “Computer simulation of cellular self-sorting: A topological exchange model”. In: *J. Theor. Biol.* 84.4 (1980), 673690. DOI: 10.1016/s0022-5193(80)80027-0.

[148] R. J. Matela and R. J. Fletterick. “A topological exchange model for cell self-sorting”. In: *J. Theor. Biol.* 76.4 (1979), 403414. DOI: 10.1016/0022-5193(79)90009-2.

[149] S. Duvdevani-Bar and L. Segel. “On Topological Simulations in Developmental Biology”. In: *J. Theor. Biol.* 166.1 (1994), 3350. DOI: 10.1006/jtbi.1994.1003.

[150] A Tardieu and M Delaye. “Eye Lens Proteins and Transparency: From Light Transmission Theory to Solution X-Ray Structural Analysis”. In: *Annu. Rev. Biophys. Biophys. Chem.* 17.1 (1988), 4770. DOI: 10.1146/annurev.bb.17.060188.000403.

[151] M. C. Gibson, A. B. Patel, R. Nagpal, and N. Perrimon. “The emergence of geometric order in proliferating metazoan epithelia”. In: *Nature* 442.7106 (2006), 10381041. DOI: 10.1038/nature05014.

[152] F. T. Lewis. “The effect of cell division on the shape and size of hexagonal cells”. In: *The Anatomical Record* 33.5 (1926), 331355. DOI: 10.1002/ar.1090330502.

[153] Y.-i. Nakajima, E. J. Meyer, A. Kroesen, S. A. McKinney, and M. C. Gibson. “Epithelial junctions maintain tissue architecture by directing planar spindle orientation”. In: *Nature* 500.7462 (2013), 359362. DOI: 10.1038/nature12335.

[154] R. Nagpal, A. Patel, and M. C. Gibson. “Epithelial topology”. In: *Bioessays* 30.3 (2008), 260266. DOI: 10.1002/bies.20722.

[155] P. M. Morse. “Diatomic Molecules According to the Wave Mechanics. II. Vibrational Levels”. In: *Phys. Rev.* 34.1 (1929), 5764. DOI: 10.1103/physrev.34.57.

[156] F. van Roy and G. Berx. “The cell-cell adhesion molecule E-cadherin.” eng. In: *Cell. Mol. Life Sci.* 65.23 (2008), pp. 3756–3788. DOI: 10.1007/s00018-008-8281-1.

[157] M. P. Stemmler. “Cadherins in development and cancer.” eng. In: *Mol. Biosyst.* 4.8 (2008), pp. 835–850. DOI: 10.1039/b719215k.

[158] O. Wartlick, P. Mumcu, A. Kicheva, T. Bittig, C. Seum, F. Julicher, and M. Gonzalez-Gaitan. “Dynamics of Dpp Signaling and Proliferation Control”. In: *Science* 331.6021 (2011), 11541159. DOI: 10.1126/science.1200037.

[159] A. Mkrtchyan, P. Madhikar, J. Åström, and M. Karttunen. “Working title:Effects of Cell Division Variance on Epithelial Topologies.” In: *Manuscript to be submitted.* (2015).

[160] H. Sutter. “The free lunch is over: A fundamental turn toward concurrency in software”. In: *Dr. Dobbs journal* 30.3 (2005), pp. 202–210.

[161] H. Sutter. *Welcome to the Jungle.* http://herbsutter.com/welcome-to-the-jungle/. Blog. 2012.

[162] M. Rupp. *CPU, GPU and MIC Hardware Characteristics over Time.* Blog. 2013.

[163] J. E. Stone, D. Gohara, and G. Shi. “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems.” In: *Computing in science & engineering* 12.3 (May 2010), pp. 66–72. DOI: 10.1109/MCSE.2010.69.

[164] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten. “GPU-accelerated molecular modeling coming of age.” In: *Journal of molecular graphics & modelling* 29.2 (Sept. 2010), pp. 116–25. DOI: 10.1016/j.jmgm.2010.06.010.

[165] W. Liu, B. Schmidt, G. Voss, and W. Mller-Wittig. “Molecular Dynamics Simulations on Commodity GPUs with CUDA”. In: *Lecture Notes in Computer Science* (2007), 185196. DOI: 10.1007/978-3-540-77220-0_20.

[166] J. Yang, Y. Wang, and Y. Chen. “GPU accelerated molecular dynamics simulation of thermal conductivities”. In: *J. Comput. Phys.* 221.2 (2007), 799804. DOI: 10.1016/j.jcp.2006.06.039.

[167] J. L. Klepeis, K. Lindorff-Larsen, R. O. Dror, and D. E. Shaw. “Long-timescale molecular dynamics simulations of protein structure and function”. In: *Curr. Opin. Struct. Biol.* 19.2 (2009), 120127. DOI: 10.1016/j.sbi.2009.03.004.

[168] A. Sunarso, T. Tsuji, and S. Chono. “GPU-accelerated molecular dynamics simulation for study of liquid crystalline flows”. In: *J. Comput. Phys.* 229.15 (2010), 54865497. DOI: 10.1016/j.jcp.2010.03.047.

[169] A. W. Götz, M. J. Williamson, D. Xu, D. Poole, S. Le Grand, and R. C. Walker. “Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born.” In: *J. Chem. Theory Comput.* 8.5 (May 2012), pp. 1542–1555. DOI: 10.1021/ct200909j.

[170] R. Salomon-Ferrer, A. W. Gtz, D. Poole, S. Le Grand, and R. C. Walker. “Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald”. In: *J. Chem. Theory Comput.* 9.9 (2013), 38783888. DOI: 10.1021/ct400314y.

[171] J. Nickolls, I. Buck, M. Garland, and K. Skadron. “Scalable parallel programming with CUDA”. In: *Queue* 6.2 (2008), p. 40. DOI: 10.1145/1365490.1365500.

[172] S. Cook. *CUDA programming: a developer’s guide to parallel computing with GPUs.* Newnes, 2012.

[173] J. Hoberock and N. Bell. *Thrust.* 2008.

[174] Khronos OpenCL Working Group and others. “The opencl specification”. In: *version* 1.29 (2008), p. 8.

[175] C. Nvidia. *Programming guide.* 2008.

[176] NVIDIA. “NVIDIA’s Next Generation CUDA^{TM} Compute Architecture: Kepler^{TM} GK110”. In: NVIDIA. NVIDIA, 2015, pp. 9–10.

[177] H. Shen. “The compressive mechanical properties of C n (n = 20, 60, 80, 180) and endohedral M@C 60 (M = Na, Al, Fe) fullerene molecules”. In: *Mol. Phys.* 105.17-18 (2007), pp. 2405–2409. DOI: 10.1080/00268970701679467.

[178] H. Shen. “Mechanical properties and electronic structures of compressed C60, C180 and C60@C180 fullerene molecules”. In: *J. Mater. Sci.* 42.17 (2007), 73377342. DOI: 10.1007/s10853-007-1576-z.

[179] P. Fowler and D. Manolopoulos. *An Atlas of FULLERENES.* Dover Publications, Inc., 2007.

[180] L. Hayflick. “The limited in vitro lifetime of human diploid cell strains”. In: *Exp. Cell Res.* 37.3 (1965), 614636. DOI: 10.1016/0014-4827(65)90211-9.

[181] L. Hayflick and P. Moorhead. “The serial cultivation of human diploid cell strains”. In: *Exp. Cell Res.* 25.3 (1961), pp. 585–621. DOI: 10.1016/0014-4827(61)90192-6.

[182] D. Bakowies, M. Bhl, and W. Thiel. “A density functional study on the shape of C180 and C240 fullerenes”. In: *Chem. Phys. Lett.* 247.4-6 (1995), pp. 491–493. DOI: 10.1016/s0009-2614(95)01222-2.

[183] M. Yoshida. *M. Yoshida's Fullerene Library*. 1999. URL: http://www.jcrystal.com/steffenweber/gallery/Fullerenes/Fullerenes.html.

[184] K. K. Hirschi. "PDGF, TGF-beta , and Heterotypic Cell-Cell Interactions Mediate Endothelial Cell-induced Recruitment of 10T1/2 Cells and Their Differentiation to a Smooth Muscle Fate". In: *The Journal of Cell Biology* 141.3 (1998), 805814. DOI: 10.1083/jcb.141.3.805.

[185] A Pierres, A. Benoliel, and P Bongrand. "Cell-cell interactions". In: *Physical chemistry of biological interfaces* (2000), pp. 459–522.

[186] U Rutishauser, A Acheson, A. Hall, D. Mann, and J Sunshine. "The neural cell adhesion molecule (NCAM) as a regulator of cell-cell interactions". In: *Science* 240.4848 (1988), 5357. DOI: 10.1126/science.3281256.

[187] H. W. Jones. "Record of the first physician to see Henrietta Lacks at the Johns Hopkins Hospital: History of the beginning of the HeLa cell line". In: *Am. J. Obstet. Gynecol.* 176.6 (1997), s227s228. DOI: 10.1016/s0002-9378(97)70379-x.

[188] The Johns Hopkins Hospital. *Surgical Biopsy Number 92498*. Baltimore, MD, 1951.

[189] L. Zhao, C. D. Kroenke, J. Song, D. Piwnica-Worms, J. J. H. Ackerman, and J. J. Neil. "Intracellular water-specific MR of microbead-adherent cells: the HeLa cell intracellular water exchange lifetime". In: *NMR Biomed.* 21.2 (2008), 159164. DOI: 10.1002/nbm.1173.

[190] R. Milo, P. Jorgensen, U. Moran, G. Weber, and M. Springer. "BioNumbers–the database of key numbers in molecular and cell biology". In: *Nucleic Acids Res.* 38.Database (2009), D750D753. DOI: 10.1093/nar/gkp889.

[191] K. Park, J. Jang, D. Irimia, J. Sturgis, J. Lee, J. P. Robinson, M. Toner, and R. Bashir. "'Living cantilever arrays' for characterization of mass of single live cells in fluids." eng. In: *Lab. Chip* 8.7 (2008), pp. 1034–1041. DOI: 10.1039/b803601b.

[192] S. W. Grill and A. A. Hyman. "Spindle positioning by cortical pulling forces." eng. In: *Dev. Cell* 8.4 (2005), pp. 461–465. DOI: 10.1016/j.devcel.2005.03.014.

[193] J. P. A. Baak, E. Gudlaugsson, I. Skaland, L. H. R. Guo, J. Klos, T. H. Lende, H. Søiland, E. A. M. Janssen, and A. Zur Hausen. "Proliferation is the strongest prognosticator in node-negative breast cancer: significance, error sources, alternatives and comparison with molecular prognostic markers." eng. In: *Breast Cancer Res. Treat.* 115.2 (2009), pp. 241–254. DOI: 10.1007/s10549-008-0126-y.

[194] L. Medri, A. Volpi, O. Nanni, A. M. Vecci, A. Mangia, F. Schittulli, F. Padovani, D. C. Giunchi, A. Zito, A. Vito, D. Amadori, A. Paradiso, and R. Silvestrini. "Prognostic relevance of mitotic activity in patients with node-negative breast cancer." eng. In: *Mod. Pathol.* 16.11 (2003), pp. 1067–1075. DOI: 10.1097/01.MP.0000093625.20366.9D.

[195] *Blender*. http://www.blender.org.

[196] N. Darbelley, D. Driss-Ecole, and G. Perbal. "Elongation and mitotic activity of cortical cells in lentil roots grown in microgravity." In: *Plant Physiological Biochemistry* 27 (1989), pp. 341–347.

[197] L. Jin, T. H. Murakami, N. A. Janjua, and Y. Hori. "The effects of zinc oxide and diethyldithiocarbamate on the mitotic index of epidermal basal cells of mouse skin." eng. In: *Acta Med. Okayama* 48.5 (1994), pp. 231–236.

[198] P. Muehlbauer and M. Schuler. "Measuring the mitotic index in chemically-treated human lymphocyte cultures by flow cytometry". In: *Mutation Research/Genetic Toxicology and Environmental Mutagenesis* 537.2 (2003), 117130. DOI: 10.1016/s1383-5718(03)00076-7.

[199] J. N. Miller and G. W. Milton. "An experimental comparison between tumour growth in the spleen and liver". In: *The Journal of Pathology and Bacteriology* 90.2 (1965), 515521. DOI: 10.1002/path.1700900219.

[200] T. Nakano and K. Oka. "Differential values of ki-67 index and mitotic index of proliferating cell population. An assessment of cell cycle and prognosis in radiation therapy for cervical cancer". In: *Cancer* 72.8 (1993), 24012408. DOI: 10.1002/1097-0142(19931015)72:8<2401::aid-cncr2820720818>3.0.co;2-d.

[201] A. V. Pisciotta, D. W. Westring, C. Deprey, and B. Walsh. "Mitogenic Effect of Phytohaemagglutinin at Different Ages". In: *Nature* 215.5097 (1967), 193194. DOI: 10.1038/215193a0.

[202] M. Stücker, A. Struk, P. Altmeyer, M. Herde, H. Baumgärtl, and D. W. Lübbers. "The cutaneous uptake of atmospheric oxygen contributes significantly to the oxygen supply of human dermis and epidermis". In: *The Journal of Physiology* 538.3 (2002), 985994. DOI: 10.1113/jphysiol.2001.013067.

[203] E. Proksch, J. M. Brandner, and J.-M. Jensen. "The skin: an indispensable barrier". In: *Exp. Dermatol.* 17.12 (2008), 10631072. DOI: 10.1111/j.1600-0625.2008.00786.x.

[204] S. A. Sandersius, M. Chuai, C. J. Weijer, and T. J. Newman. “Correlating Cell Behavior with Tissue Topology in Embryonic Epithelia”. In: *PLoS ONE* 6.4 (2011). Ed. by J. Langowski, e18081. DOI: 10.1371/journal.pone.0018081.

[205] M. Milán, S. Campuzano, and A. García-Bellido. “Cell cycling and patterned cell proliferation in the wing primordium of Drosophila.” eng. In: *Proc. Natl. Acad. Sci. U. S. A.* 93.2 (1996), pp. 640–645.