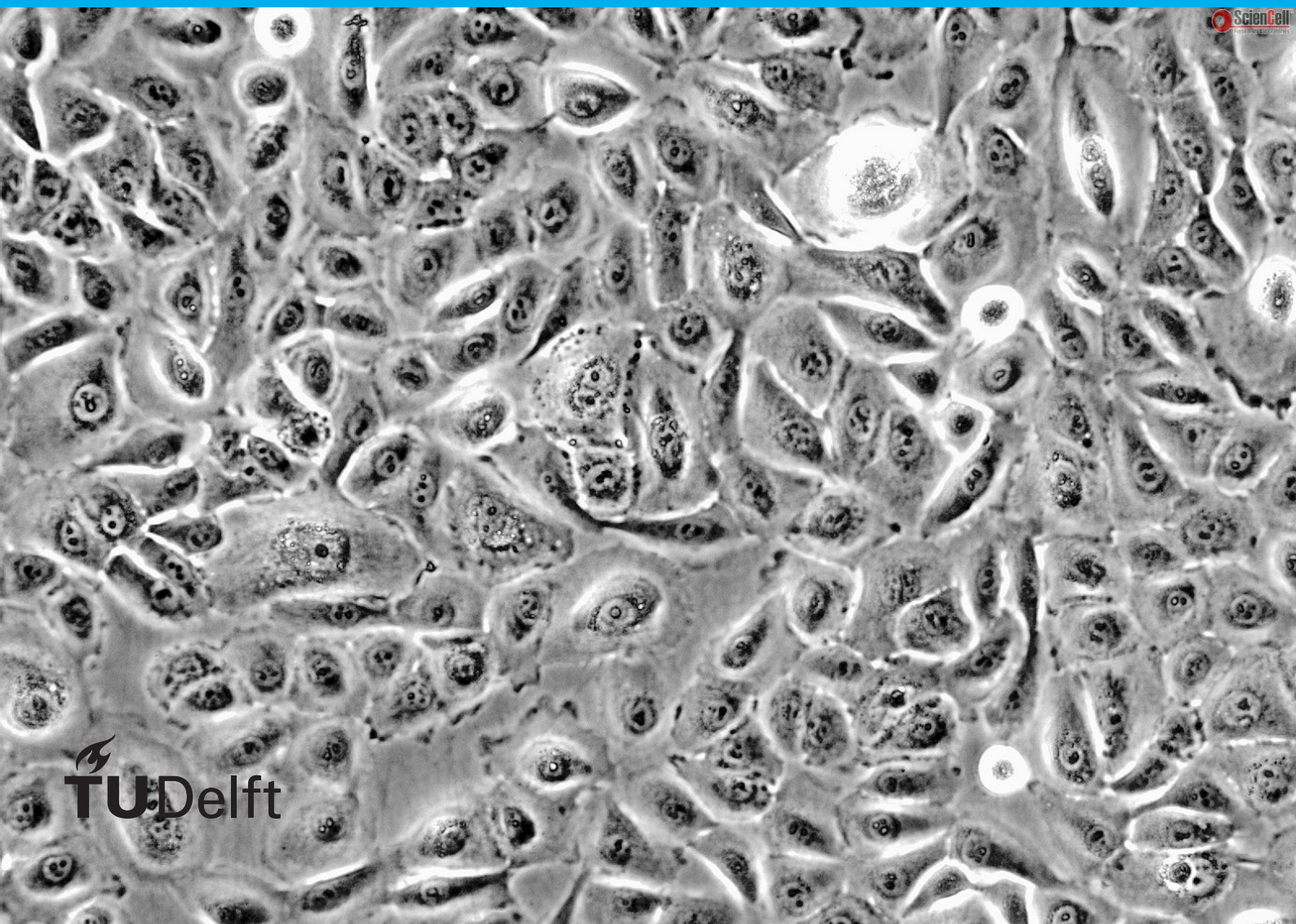
Mathematical modelling of burn injuries

Thesis report

E.D. Kleimann



Mathematical modelling of burn injuries

Thesis report

by

E.D. Kleimann

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday October 30, 2018 at 2:00 PM.

Program:	Applied Mathematics		
Specialization:	Computational Science and Engineering		
Student number:	4231791		
Project duration:	February 5, 2018 – October 30, 2018		
Thesis committee:	Dr. ir. F. J. Vermolen,	TU Delft,	supervisor
	Dr. ir. J. M. Tang,	VORtech,	supervisor
	Drs. ing. E. K. van Es,	VORtech,	supervisor
	Dr. H. M. Schuttelaars,	TU Delft	

TU Delft VORTECH

Abstract

Burn injuries can lead to serious complications that have a large influence on someone's quality of life. In order to help patients, we need to gain insight into the wound healing process and in the development of complications that come with serious burn injuries. The final goal would be to prevent or at least reduce these complications. With a mathematical model, we simulate the time-evolution of the skin after a burn injury. The objective is to be able to quantify the impact of (patient-specific) parameters on the evolution of the skin properties. In this thesis, a cell-based model for the cell migration during wound healing is presented. Two types of cells, macrophages and fibroblasts, migrate by incentives of the strain energy density and concentration fields. Two concentration fields are implemented in the model: Platelet Derived Growth Factor (PDGF) and Transforming Growth Factor β (TGF-β). PDGF is a chemical that occurs in the wound bed and TGF-β is secreted by the macrophages. The processes such as cell division and death are modelled through stochastic processes. In this way, the data-consuming cell history does not have to be taken into account.
The computational work of the corresponding simulations increases rapidly for larger numbers of cells. This holds in particular for the part that computes the strain energy density for every cell pair. This is tackled by employing the Graphics Processing Unit (GPU) for the largest bottlenecks. The CUDA framework is used to program the GPU, where certain parts of the computations can be run in parallel to make the computations more efficient. The GPU implementations are described in this report, alongside with the improved computation times. For a 2D simulation of one day, the speed-up in computation time from the CPU to the GPU implementation was a factor 58. To assess the accuracy of the computation of the concentration fields, Richardson's Extrapolation is used to estimate the order of the error. More research is required on this part to achieve reasonable outcomes for the order. Moreover, an alternate approach for determining the TGF-β field by Green's function was studied. The computational work for this method increased rapidly and was therefore unfit to be implemented in the model. Lastly, the influences of parameters on the model outcomes were investigated. Monte Carlo simulations are needed for the interpretation of the stochastic model. The influence of the time step and choice of normalization of the gradients were investigated. For the tested scenarios, the hypothesis that they behave similarly was not rejected. The single-precision implementation of the model did not lead to an overall speed-up. Especially, the increase in computation time of the solver for the concentration fields is unexpected. Using the GPU for efficiently modelling cell migration seems a good idea. The current model can be used as a basis for more sophisticated models in the future.

Preface

This thesis report marks the end of my Master's Program in Applied Mathematics. It is the last requirement to complete in order to obtain my Master's degree in the specialization of Computational Science and Engineering. Over the last nine months, I have worked on the subject of mathematical modelling of burn injuries. The main challenge in my thesis was to investigate the possibilities that GPU computing offers to speed up the wound healing model. The research was conducted in cooperation with VORtech, a company that supports developers and users of computational software. Their expertise lies in numerical modeling, scientific software development, applied mathematics and data science. For this project, especially their expertise on the subject of GPU computing was very valuable.

During this project, I have learned a lot about wound healing and about the mathematical models behind it. Moreover, I learned a lot about programming in C++ and CUDA. I would like to thank my supervisors for their constant support during this project. First of all, I would like to express my gratitude to my TU Delft supervisor, Fred Vermolen. Your enthusiasm for the research on wound healing is very contagious and by the endless amount of research ideas, you managed to keep me motivated during the whole project. Thanks for all the cups of coffee and reminding me that when something does not work as expected, it is the more interesting. Secondly, thanks to my supervisors from VORtech, Eli van Es and Jok Tang. Every time I encountered some problem, you were always willing to brainstorm with me about possible explanations and solutions. Especially the advice on how to debug a certain problem from time to time has been very helpful! I am also very grateful for the opportunity to conduct this research at VORtech. I found the ambiance at VORtech inspiring and open, which made working there very pleasant. Thanks to all the colleagues for showing interest in my work.

My thesis committee consists of the aforementioned supervisors and Henk Schuttelaars, a member of the Mathematical Physics group at TU Delft. I would like to thank Henk Schuttelaars for his willingness to be part of my graduation committee. Last but not least, I would like to thank my family and friends for always having supported me during my studies. And finally, I wish everyone interested in this thesis, much pleasure in reading my report.

E.D. Kleimann
Delft, October 2018

Contents

1 Introduction

Wound healing is a complex biological process with many components. It is an important research area as there are still complications occurring which we like to prevent, or at least reduce. Burn injuries can result in complications such as contractions and hypertrophic scars. These complications have a radical impact on someone's quality of life. In the first place, hypertrophic scars and contractions cause aesthetic problems. Furthermore, contractions can lead to reduced mobility. If the skin contracts strongly on an important area, this is very problematic. A burn injury in the neck, for example, can cause contraction that makes it impossible for someone to lift his or her head.

Ideally, complications are prevented from becoming chronic problems. The pathways that lead to these complications are unknown, and furthermore it seems difficult to influence the material properties of scar tissue [21]. Scar tissue has different properties than uninjured tissue. The contractions occur by pulling forces from fibroblasts and myofibroblasts. This process depends on the apoptosis (programmed cell death) rate of myofibroblasts and the secretion rate of collagen molecules. Hypertrophic scars are caused by a disruption of the restoration process. In this case the apoptosis rate is lower than usual. Research indicates that epithelial tension, topography and stiffness are influencing how the skin behaves [10].

Mathematical modelling is a useful tool for untangling the complexity of the wound healing process. The interactions can be investigated for specific parameters and then essential components can be discovered. We need to find out more about the mechanical properties of the skin to know which factors are of importance. With more insight into the wound healing process, we should be able to develop medicine and treatments to prevent complications. With a model, different treatments can be tested and the best option to prevent complications can be predicted.

The required computational work that is involved with large-scale simulations should be taken into consideration. The work for detailed simulations in which individual cells are tracked, increases rapidly when scaling up the domain or timescale. Moreover, performing the same stochastic simulation multiple times in order to predict the outcome, is a costly operation. In scientific computing, the use of the GPU (Graphics Processing Unit) to speed up computations is becoming more and more popular. With the arrival of new hardware and software (for example, the CUDA framework), using the GPU in programming became an accessible option.

1.1. Wound healing model

There are many possibilities to model the wound healing of burn injuries. In this thesis a cell-based model has been chosen. The cells are discrete entities, whereas the concentrations are continuous entities over the domain. Moreover, the equations for migration used in this modelling project have stochastic components and therefore the results have to be interpreted statistically. The final model can roughly be divided into three main components:

1. A particle model, which can be composed of different types of cells. The cell migration computation is based on the strain energy densities, chemical gradients and forces in the medium.

2. A chemical field, by means of a Finite Element solution for the different chemical fields. This includes both the secretion and diffusion of chemicals. The equations used in this part are of the form

$$\frac{\partial c}{\partial t} - D\Delta c = \sum_{j \in \mathcal{S}(t)} \gamma_j \delta(x - x_j(t)),$$

where c is the concentration, D a diffusion coefficient and γ_j a coefficient for the amount of secreted chemical. The right hand side represents the secretion of a chemical by a certain cell j in the set of cells $\mathcal{S}(t)$, which is modelled by the Dirac Delta function $\delta(x)$. Another possibility is to have a chemical field as result of the wound, as in Section 4.3. The different molecules and growth factors are described in [21], which will be used as a basis for the chemical composition of the domain.

3. A mechanical field, to incorporate the forces that (myo)fibroblasts exercise on their environment. This will allow us to model the contraction of the wound. The equations to be implemented have the form

$$\begin{aligned} -\nabla \cdot \underline{\underline{\sigma}} &= \underline{F} \\ \underline{F} &= \sum_{j \in \mathcal{S}(t)} \int_{\Gamma_j} F(x'_j)\delta(x - x'_j)\underline{n}(x'_j)d\Gamma \\ &\approx \sum_{j \in \mathcal{S}(t)} \sum_{k=1}^{n} F_{jk}\delta(x - x_{jk})n_{jk}\Delta\Gamma_{jk} \qquad \text{for } \lim n \to \infty \end{aligned}$$

where $\underline{\underline{\sigma}}$ is the stress tensor and $\underline{F}$ the cell forces. If we simplify the cell boundary to be n straight edges, then the cell force can be approximated by a finite sum over point forces on these line segments.

In this project, the first two components of the previously mentioned list (hence no forces) have been dealt with. For these parts, we have attempted to speed up computations to attain an efficient code. The structure of the final program is a combination of different programming languages, as for the usage of the GPU an implementation in C or C++ is necessary. The interface is made in C as this makes it compatible with other languages (MATLAB, Python, Fortran, etc.). The cell migration algorithm is written in C++ and this part contains most of the total program. The GPU is used in certain parts to speed up the computations. A user program is made in Python, as this is available freely and an easy environment for most users. In this Python program, the user can change the input parameters for the program and display the results of the simulation.

1.2. Thesis outline

The model that has been made during this project needs to be extended with more features, such as the mechanical field, in order to be able to realistically model the wound healing. However, in order to successfully make a model as described above, we need efficient and accurate methods. In this thesis project, the goal was to find an efficient way to deal with computations of large cell colonies. The opportunities that GPU computing offers are therefore an important part of this research.

During the literature study, some research questions were posed to use as a guideline for the remaining thesis work [20]. The main research question is:

'How can a cell-based model that simulates the wound healing process be made as efficient as possible?'

Additionally, five sub-questions were drawn up:

1. Which parts of the cell migration program are suitable to be run on the GPU?
2. How accurate is the approach that uses the Finite Element Method to find the concentration on the nodes and subsequently maps this to the cell centers to find the gradient? Which alternatives can be used to improve the accuracy if necessary?
3. Can the model be run in single precision and obtain comparable results for when this is done in double precision?
4. What can be said about the numerical stability and the maximal time step?

5. What is the limit in terms of the number of cells and domain size for the program to be able to perform Monte Carlo simulations?

The research questions will be addressed at various chapters in this report. Most questions (2, 3 and 4) will be addressed in the Chapters 10 and 11. The computational work is investigated for various scenarios throughout the report, therefore questions 1 and 5 are not addressed in a specific section.

The structure of this thesis is as follows. Before going into detail about the mathematical model, some interesting concepts about the biological processes are discussed. In Chapter 2, firstly the cell dynamics in general are discussed and in the second part, we go into more detail about the processes that occur during wound healing. Subsequently, we zoom in on the processes on cell level in Chapter 3. Moreover, the results of some experiments are summarized and some simplifying assumptions for the cell-based model are described.

In Chapter 4, the mathematical framework of the wound healing model is given. This framework includes the computation of the strain energy densities, mechanical energy, the random walk and the concentration fields. Furthermore, the processes of cell death and division are described. After the mathematical description, the scaling of different methods and connection with input data is made in Chapter 5. Thereafter, additional implementations to the framework in Chapter 4 are described.

Subsequently, the options for speeding up computations are discussed. In particular, information about GPU computing with CUDA is given in Chapter 6. Different features of GPU computing are discussed and an analysis of the computational work for the MATLAB model from the literature study [20] is given. In Chapter 7, a description of the model from a programming view is given. The MATLAB model is compared statistically with the C++ model and the effect of using the GPU on different parts is measured. All the implementations to reach a more efficient program (often by using the GPU) are written down in the last section of Chapter 7.

Next, the wound quantifiers are defined in Chapter 8, in order to keep track of the wound healing progress and to be able to compare simulations with each other. In order to do the comparison for different scenarios, we introduce Kolmogorov-Smirnov Two-Sample Test in the third section. In Chapter 9, the results of the simulation are shown. This involves the behavior of the wound quantifiers over time and an investigation of the computational work. Additionally, the CPU and GPU implementations are compared with each other.

Finally, the results of analyzing the model are given in Chapter 10 and 11. In Chapter 10, the error estimation with Richardson's extrapolation is performed for the PDGF field and an alternative computation of the TGF-β field is studied by using the Green's function approximation. Thereafter, it is investigated which part of the GPU computation consists of the communication between the CPU and the GPU. Lastly, we compare scenarios with different time steps, precision and choice of gradient normalization with the help of the Kolmogorov-Smirnov Two-Sample Test. The report is concluded with a conclusion of the findings in this project and suggestions for future work.

2

Skin tissue

In this chapter some biology concepts that are relevant for the modelling of biological processes on cell level are described. In [1] the main concepts of extracellular control of cell division, cell growth and apoptosis, that is programmed cell death, are discussed. Interesting points are summarized in the first section. In the second section the process of wound healing is reviewed, as this is related to the topic of this research. References [21] and [10] are used in this section.

2.1. Cell biology

Cells of multicellular organisms do not simply divide if there are sufficient amounts of nutrients (unlike unicellular organisms). Neither do cells grow (in mass) without some signal. The signal molecules that regulate cell size and cell number are generally either secreted proteins, proteins bound to the cell surface or components of the extracellular matrix (ECM). Factors that promote cellular growth can be divided into three groups:

1. Mitogens: stimulate cell division by suppressing mechanisms that block progress through the cell cycle.
2. Growth factors: stimulate cell growth (increase in mass) by promoting the increase of proteins.
3. Survival factors: stimulate survival by suppressing apoptosis.

The cell population can only grow to some maximum. For example, cells cultured in a dish only divide until the surface is filled. Cells need signals from other cells (survival factor), in order to not undergo apoptosis. This regulates cells to live where they are necessary. Furthermore, it is thought that not only contact inhibition causes the limit, but mainly the capacity of a cell to locally take up the mitogens and growth factors and not leaving enough for its neighbors.

There are over 50 proteins known to act as mitogens. For example epidermal growth factor (EGF). Many mitogens have other actions as well. Transforming growth factor-β (TGF-β) can simulate and inhibit cell proliferation at different locations. TGF-β can inhibit growth by blocking the cell cycle progress or stimulating apoptosis. Organisms seem to sense the size of their body parts and let cells die according to this (hence not according to a specific number of cells).

Cells have intracellular mechanisms that limit cell proliferation. With cell proliferation, we mean the process where a cell divides in two. After a limited number of divisions, they will permanently go to a non-dividing state. The best-understood intracellular mechanism that limits cell proliferation occurs in human fibroblasts. They undergo about 25–50 population doublings when cultured in a standard mitogenic medium. Telomeric DNA is synthesized by the enzyme telomerase (which is a different process than for what happens with the rest of the genome). The telomeres become shorter with every division, until this damage activates a cell arrest. Lack of telomerase in most body cells prevents excessive cell proliferation. This is probably a key issue in the aging of animals and humans. Unfortunately, cancer cells can produce telomeric DNA autonomously.

During blood clotting, platelets are triggered to release platelet derived growth factors (PDGF). PDGF is a mitogen that causes fibroblasts to proliferate. This happens when the fibroblasts are exposed to serum, but not when exposed to plasma. PDGF is probably an important chemical in the wound healing process.

2.2. The human skin and wound healing

Skin tissue of humans can roughly be divided into three parts: the epidermis (usual thickness around 0.1 mm), the dermis (thickness between 0.5 and 5 mm) and the hypodermis. The epidermis is the top layer and the hypodermis is located under the dermis (see Figure 2.1). The basement membrane is located between the epidermis and dermis. Cells in contact with this membrane continually divide during the lifetime of an organism, so that a protecting barrier is created [10]. This enables the skin to cope with many (harmful) factors in its environment. Per location in the skin, the structure of the skin differs to optimize for some functionality (minimize heat loss, optimize sensitivity, etc.)

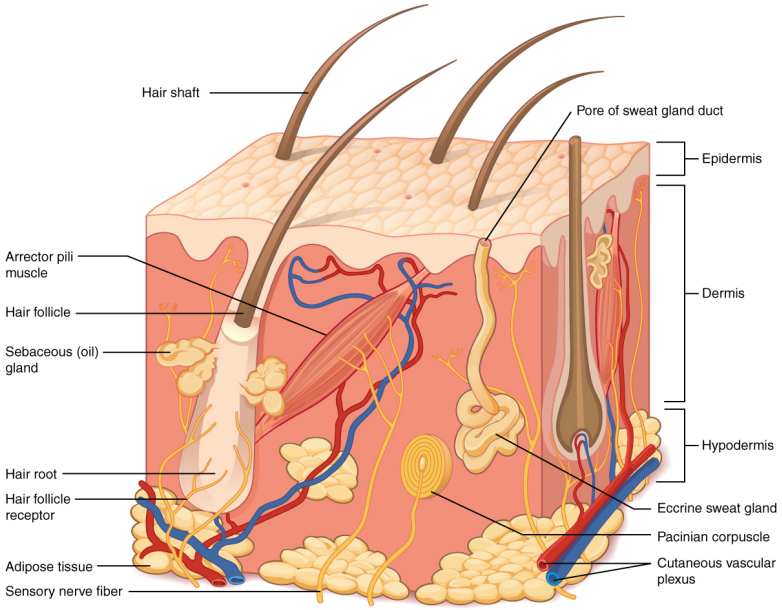


Figure 2.1: Structure of the skin, with its three main layers: the epidermis, the dermis and hypodermis from [29].

Keratinocytes are the most common cells in the epidermis. Their primary function is to protect against possible damages from the environment (for example: pathogens, UV, water loss) by producing a barrier. The keratinocytes also secrete various signaling molecules to stimulate maintenance processes. One of those processes is the initiation of the inflammatory response after dermal wounding. The keratinocytes differentiate into corneocytes during the ascend toward the top layer of the skin. Eventually, these cells lose their cohesion and separate from the surface, after which new cells will take their place.

The dermis does not contain cells for the largest part, but has many different cell types otherwise. The most important non-cellular components that maintain the integrity of the dermis are the basement membrane, the fibers and the extrafibrillar matrix components. Together they form the extracellular matrix (ECM). The most common cells in the dermis are the fibroblasts. They secrete precursors, partially differentiated cells that are used for other components in the dermis, signaling molecules, and more importantly they break down the fibrin cloth and secrete new extracellular matrix. Furthermore, fibroblasts exert contractile forces. Fibroblasts can differentiate, for example to myofibroblasts, which exert even stronger contractile forces [2].

The natural wound healing process (without intervention and in absence of complications) can be divided into four stages:

1. **Hemostasis**
   The first process is to stop the bleeding of a wound (in case of cuts). Blood vessels around injured vessels contract to reduce blood loss. Nearby specific blood elements, platelets, stick to each other

and to the injured parts. The platelets also cause reactions such that the protein fibrin forms bundles attached to the platelets. This temporary structure allows for the healing of the area. In this healing process, plasmin molecules are slowly created, which after some time break the structure down.

2. **Inflammation**
   The contraction of the blood vessels stops quickly again and the walls of the vessels become more permeable. This phase starts when plasma leaks and types of leukocytes (white blood cells) migrate from the intravascular space to the extravascular space. There are many different types of leukocytes. One important type of leukocytes is the macrophage. Macrophages are cells that clear up relatively big substances such as pathogens, chemicals and debris. Next to their clearing task, the macrophages initiate other immune reactions. Initially there are more M1 macrophages, these cells help to remove bacteria. Later, the M2 macrophages are dominating, which clean up the debris and stimulate angiogenesis. Angiogenesis is the formation of new blood vessels from pre-existing ones. Additionally, signaling molecules are secreted, which is important for the completion of the immune response.

3. **Proliferation**
   Quickly after the start of the inflammatory phase, this phase starts as well. In this phase, different processes take place: re-epithelialization (restoring the epidermis), angiogenesis, fibroplasia (restoring presence of fibroblasts and producing a new ECM) and wound contraction. Wound contraction ensures (a large) reduction of the injured area, without creating new tissue. In the context of cutaneous wounds in less favorable hygienic circumstances, this contraction is an essential evolutionary defense mechanism. However, in the wake of burn injuries under clean circumstances, this mechanism causes permanent skin deformation and permanent stresses in the skin tissue, which can lead to mobility reduction of patients. Wounds can reduce 40 to 80% in size as a consequence of wound contraction. The contraction commences approximately after a week and lasts several weeks (longer than the re-epithelialization) [42].

4. **Maturation / remodeling**
   This phase starts shortly after the start of the previous phase, but takes place much longer (more than a year). The ECM is remodeled: the amounts of molecules are changing and the collagen molecules are adjusted in the way they are aligned and interconnected. The result is a grid of collagen bundles that is aligned more parallel to the surface than is the case for undamaged tissue. Eventually, the density of several cell types, in particular of (myo)fibroblasts, decreases in the scar tissue.

Fibroblasts migrate as a result of the chemokine gradient and collagen orientation and deposit collagen along their trails. Fibroblasts differentiate into myofibroblasts as a result of increasing stress and changes in the ECM. Myofibroblasts come into action after fibroblasts stop migrating and stick to each other and the wound edges. They link to ECM molecules (fibronectin and collagen) and pull on these molecules when contracting. The contraction caused by myofibroblasts is much higher than the contraction that was caused by the fibroblasts at the earliest stages.

Scarred skin tissue is different from undamaged skin. There is absence of hair follicles and sweat glands and scarred skin tissue has a longitudinal structure (opposed to a more 'basket weave' pattern normally). Scars can occur depending on the wound size and in case of tissue contraction hypertrophic scars may occur. Hypertrophic scars can be caused by a lack of apoptosis of myofibroblasts, this results in an excessive production of extracellular matrix protein. The mechanical behavior of soft tissues depends highly on concentration and structure of its constituents, such as collagen and elastin. Fibers have preferred directions and therefore the tissue behaves anisotropically (has direction dependent behavior).

In case of severe wounding, the main issue is to cover up the injured area to prevent dehydration and infections. This is ideally done by another part of the patients own skin. A donor or a suitable other dressing (technologies involving synthetic, human or animal matrix proteins) can be used otherwise. Skin ulcers (area where skin dies and becomes loose) can occur with patients that have diseases or syndromes that affect the micro-circulation (diabetes for example). Very few drugs have proven to be useful enough to be applied into practice. Epidermal growth factor (stimulated division and migration of keratinocytes in vitro) was thought to have much potential, however, it did not turn out to be successful.

3

Cell dynamics

Different small scale processes can have a joint effect on a higher scale, which makes it difficult to unravel the structure of the cell dynamics during wound healing. Before starting to describe a model for the general wound healing processes from Chapter 2, the small scale interactions are important to know about, in order to build up the model from this. The most important factors that influence cell movement are presented in this chapter.

3.1. Cell migration

Despite the fact that cells in natural environments live and migrate in three-dimensional tissues, we consider experiments where cells migrate over a planar, two-dimensional elastic substrate. The experimental conditions that we model are investigated in many in vitro studies, that are studies on a laboratory scale, such as in [30]. A living cell exerts an upward force on its substrate. This is the traction force (F) and the influence of this is felt by other cells in the environment. The cells interact by migrating towards each other as the result of detecting mechanical signals. Cells that are in mechanical contact, are pushed away from each other (hard impingement). This field with distortions decides how cells move and can be expressed by the strain energy density [37]. For dead cells, the traction force is zero. The traction force is dependent on the phenotype (physical appearance), so we take it constant for all cells of a certain phenotype.

Cell migration can depend on many different properties of the substrate. First of all, there are the chemical properties. Chemotaxis is the migration depending on the gradient of concentration, where it can also depend on the cellular adhesion (haptotaxis). Migration caused by mechanical influences are tensotaxis, which entails migration depending on the gradient of strain, and durotaxis, which is migration depending on the gradient in stiffness. Furthermore, migration can depend on electrical cues and light activation. Finally, the extracellular matrix contains unpredictable irregularities and cells also contain unpredictable mobility, which results in random walk.

3.2. Experiments

The easiest way to find out more about specific dynamics is by doing experiments in vitro. Many groups carry out experiments in vitro and these experiments are considered as a model for the in vivo, that is clinical, processes. Although the results from in vitro experiments cannot always be linked one-to-one to in vivo processes, these experiments provide very valuable insights for understanding clinical processes. In general the in vitro experiments help estimating certain rate parameters as cell migration speed or division rates and can be used to examine the impact of isolated parameters and sub-processes on other processes. In general it is easier to validate mathematical models to in vitro experiments than to in vivo experiments. The findings described in this section have been observed in vitro.

Experiments are usually performed for a mono-layer of cells on a chosen substrate. Scrape wounding, a technique where a mono-layer of cells is disturbed, exists in many different forms [3]. If for example a rotating silicone tip is used, we get circular wounds as in Figure 3.1. The wound size can then be measured more easily.

Another more sterile approach is having a removable fence in a cell culture that prevent cells to enter a certain area.

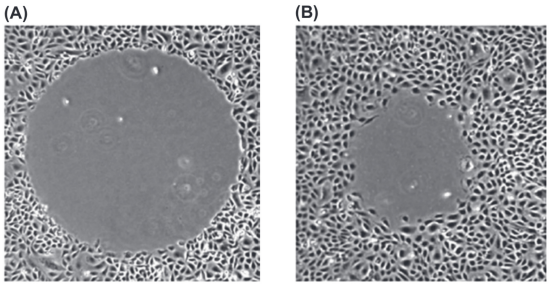


Figure 3.1: Wounding YAMC cells (conditionally immortalized mouse colon epithelial cell) with rotating silicone tip initially (A) and after 8 hours (B) in the presence of 10 ng/ml EGF [3].

In [12] it is found that isolated cells can have a near spherical shape and move randomly through a medium. In multicellular aggregates, their shape resembles a polyhedron and its two-dimensional projection resembles a polygonal shape. When cells divide, it is assumed this happens at the location where the largest force was directed to. A cell population was found to grow exponentially for a small time scale and later on proportionally to t^n with n around 2 or 3 (tumor cells grow faster). The latter behavior can be seen as a saturation stage. In practice, a sufficient amount of cell death needs to happen to have actual saturation.

Similar results are found in [8], where cells also move randomly and are (near) spherical after division. Furthermore, it is stated that during mitosis, the shape resembles a dumbbell. For proliferation the glucose amount should be above a certain level, while for apoptosis this should be below a certain level. An increase of motility causes an increase of the growth rate.

In [3] it is described how cells around a wound change their appearance to be flatter and broader. Their movement towards a chemical gradient or empty space can be described by change from leading to trailing edge. The phenomena may be more accurately described by a change in phenotype. Furthermore, cell migration has a positive relation with adhesion to the substrate.

The stiffness of the substrate influences the extent of a cell's response to the traction forces of neighboring cells [30]. It is found that cells exert more force on stiffer substrates, but that disturbances do not reach very far. On softer substrates, this is the other way around. Observations for two cells are as follows: On soft substrates (500 Pa) the cells touch and remain that way. On intermediate substrates (2500 and 5500 Pa), the cells tend to contact, separate and retouch repeatedly. They do not move significantly far away from each other, unless there is a third cell. On stiff substrates (33000 Pa) the cells touch and migrate away. The research shows that endothelial cells prefer to remain in contact at tissue-like stiffness, but migrate away on stiffer ones.

In [10] it is described that the elastic modulus of the skin greatly varies with respect to the location, the state of underlying muscle and the shear forces during contact with objects. A stiffer substrate makes cells spread out more and promotes cell division [10]. A study on epithelial cells found that cell migration is more rapid on stiffer substrates. This would indicate quicker wound healing. Another study found that during wound healing, the wound becomes stiffer. This could be a mechanism to ensure wound coverage, without being dependent on a stiff location.

Experiments [42] on porcine skin (similar to human skin) led to believe that central granulation tissue (new connective tissue and microscopic blood vessels that form on the surfaces of a wound during the healing process) is not responsible for contraction. Neither are the edges pushed in by processes occurring in peripheral tissue nor is a purse string mechanism responsible. The researchers believe that the contraction is caused by fibroblasts.

In wound healing, many types of growth factors act as chemokines for fibroblasts: TGF-β, PDGF (platelet derived growth factor) and FGF (fibroblast growth factor), etc. Cells that generate the force are located in the

wound margins. A rim of densely packed fibroblasts pull the dermal edges inward, while fibroblasts inside the wound cause a more normal arrangement of collagen. Fibroblasts cause two forces: an isometric contractile force (fibroblasts contract and pull surrounding towards itself) and a traction force. The traction force pulls the environment in the direction of the migration.

The ECM is modelled anisotropically in [42]. With collagen orientation, the wound granulation tissue grows into fiber-reinforced anisotropic soft tissue. Furthermore, the fiber direction is inhomogeneous. Collagen responds to the mechanical field, in a way that will align them with the tension lines.

Mechanical stress is found to have a significant influence [12], [8], [10]. It influences the growth of cells. For realistic ranges of cell volume compressions, the specific representation of cell shape has negligible effects on the outcome of the model.

3.3. Initialization of the model

The model needs parameters as input, for example the elastic modulus, the cell mobility rate and cell-substrate adhesion. The possibilities of collecting information on parameters are constantly improving, but specific information remains hard to obtain.

Cells behave like viscoelastic bodies, or even isotropic bodies on small time scales. Different models can be chosen for the simulation of cell dynamics. There is a choice between lattice-based models and off-lattice models. For off-lattice models the cells can be chosen to have the form of a quasi-spherical particle, a deformable ellipsoid or a Voronoi polygon. Simulations according to this approach are limited to numbers around 10^6 cells [12].

To model cell dynamics, we need some simplifying assumptions. In [12] it was found that on scales larger than 10 times the cell diameter, some simplifications do not alter the qualitative behavior of the system. The specific representation of cells, the orientation of division and the exact shape of interaction force do not influence the system too much. Furthermore, cell migration and growth can be modelled by a Monte Carlo approach, instead of taking the entire cell history into account.

In some modelling scenarios, the apoptosis rate does not influence the resulting number of the cell population. For example, when simulating tumor growth, every cell that dies is almost immediately replaced by a new cell as space is the limiting factor.

4

Mathematical model

Different types of models can be chosen for the modelling of biological processes. The actual implementation varies for specific processes, however, in general we can divide the models into space-free level, compartment level, cellular based and tissue level models. Within the cellular based models, a distinction is made between changing cell shape and constant cell shape models [36]. In this chapter, an overview will be given of the chosen mathematical model, including the specific equations for the occurring processes.

4.1. Constant cell shape model

In this research a cell-based model is chosen. This means that throughout the simulation a cell is tracked. The advantage of this modelling approach is that it can easily be adjusted to the set-up of a specific experiment. In vitro experiments can then give precise information on the correctness of the model. In the models a probabilistic approach is used to simulate cell dynamics. In contrast to the biological concept, the history regarding a cell is not known in the model and therefore it is not possible to determine the cell differentiation and divisions in cells exactly.

Furthermore, the constant cell shape model will be used in this research. This approach is computationally less expensive than the form with cell deformation and can therefore be applied on a larger scale. The two main assumptions in this model dictate that the geometry of the cells is fixed and that this geometry depends on the phenotype. In the 2D model hemi-spherical cells (the projection on the substrate is assumed circular) will be used and in 3D spherical cells.

4.1.1. Strain energy density, mechanical energy and random walk

The strain energy density is a main component for calculating the cell migration. The advantage of this quantity is its additivity. For circular projections onto the substrate, the strain energy density M_i^0 is defined as

$$M_i^0 = \frac{1}{2}\sigma\epsilon = \frac{F_i^2}{2E_s(x_i)\pi^2 R^4}, \qquad \text{where} \quad F_i = \begin{cases} F_i\,, & \text{if viable,} \\ 0\,, & \text{if dead or proliferating} \end{cases} \text{[4].}$$

Here, σ is the stress, ϵ the strain, E_s is the elasticity modulus of the substrate and R is the radius of the cell. The cell traction force is displayed as F_i, this force exists because the cell is 'pulling' on its environment (the substrate in the current case).
The strain density from a cell is weaker further away from the cell. In [37] it is found that this can be approximated by exponential decay. The strain density at x resulting from a cell located at x_i then depends on the distance as

$$M_i(x) = M_i^0 \, \exp\left\{-\lambda_i \frac{\|x - x_i\|}{R}\right\}.$$

Here, $\lambda_i = \frac{E_s(x_i)}{E_c^i}$ is the ratio between the elasticity modulus of the substrate and the elasticity modulus of the cell, which represents a factor determining the attenuation of the signal.

Summing over all densities that influence a single cell, gives the strain energy density on a cell located at x_i:

$$M(x_i) = M_i^0 + \sum_{j \neq i} M_j^0 \exp\left\{-\lambda_j \frac{\|x_i - x_j\|}{R}\right\}. \tag{4.1}$$

As cells can be influenced by different cells at different locations, the direction of the final movement $\hat{z}_i$ is calculated. The M_j's operate as weighing factors.

$$z_i = \sum_{j \neq i} M_j(x_i) \frac{x_j - x_i}{\|x_j - x_i\|} \qquad \text{and then} \qquad \hat{z}_i = \frac{z_i}{\|z_i\|} \quad \text{after normalization [4].}$$

Combining the above components with a time step, gives the displacement

$$dX_i(t) = \kappa_i M(x_i) \hat{z}_i \, dt, \tag{4.2}$$

where κ_i is a parameter that can be adjusted for different types of cells. It is defined as $\kappa_i = \frac{\gamma_i R^3}{\mu \tilde{F}^2} F_i$. Here, γ_i is the mobility of the cell (depending on the cell viability) and μ is the resistance parameter of the substrate friction. Note that κ_i becomes zero when the cell is not viable.

It is not plausible that cells from any arbitrary distance can sense each other. Therefore a parameter ϵ is set that indicates the minimal strain energy density needed to detect a cell [30]. Thus the criterion $M_i(x) = M_i^0 \exp\{-\lambda_i \frac{\|x - x_i\|}{R}\} \geq \epsilon$ should be satisfied. Reference [30] suggests that a distance $d = 29.5\mu$ m represents the maximal distance between which two cells can sense each other's strain field. Filling this d in for the M_i formula, allows us to find an ϵ. This value is then used for all the cell relations, to either allow the cells to detect each other or to set the energy contribution to zero.

To prevent cells from being able to move through each other, an opposed energy is created when cells collide. This energy needs to be subtracted from the energy density relation M_i from the two cells in question. The magnitude of the energy depends on the overlap h of the particles and is zero otherwise. Hence h is defined as $h = \max(2R - \|x_i - x_j\|, 0)$. The collision energy will be given by $M^{ij} = \frac{4}{15\sqrt{2}} \frac{\sqrt{R} E_c h^{(5/2)}}{\pi R^3}$, as was stated in [4] and [37].

As we adjust the value of $M_i(x)$ directly after computing the strain energy density, the potential change of direction is already included in the direction vector $\hat{z}_i$. Hence, the absolute value has to be taken when calculating the energy between two cells and to be added up afterwards, to prevent cancellation of forces. Cells also move randomly to a small extent. This is simulated by adding a term based on a Wiener process to the displacement [4]. Equation 4.2 is extended to

$$dX_i(t) = \kappa_i M(x_i) \hat{z}_i \, dt + \sqrt{2D} dW(t), \tag{4.3}$$

where D is the cell diffusivity parameter and $dW(t)$ is a Wiener process based on the normal distribution with mean 0 and variance dt.

4.2. Proliferation and Apoptosis

Next, the proliferation and apoptosis of cells are incorporated in the model. In order to potentially do this, a cell should have enough space to divide or be under enough force to die. Furthermore, we require a cell to be mature enough to perform these actions. The existence time of a cell is used as a prerequisite:

$$\text{Existence time } \tau \text{ should satisfy: } \begin{cases} \tau \geq 300s & \text{for proliferation,} \\ \tau \geq 300s & \text{for apoptosis.} \end{cases}$$

If these requirements are fulfilled, the decision of these processes are modelled as a (memoryless) exponential distribution. Let p_i be the probability rate, that is probability per unit of time, for cell proliferation p_d or apoptosis p_a. The probability for proliferation or apoptosis during an interval $(t, t + \Delta t)$ is then given by

$$\begin{aligned} P(t < \tau < t + \Delta t) &= \int_t^{t+\Delta t} p_i \exp\{-p_i(s - t)\} \, ds \qquad , \quad p_i \in \{p_d, p_a\}, \\ &= [-\exp\{-p_i(s - t)\}]_t^{t+\Delta t}, \\ &= 1 - \exp\{-p_i \Delta t\}. \end{aligned} \tag{4.4}$$

To determine if a cell will divide or die, we pick the stochastic variable ξ out of a uniform distribution on $[0,1]$. The process will take place if $0 \leq \xi \leq 1 - \exp\{-p_i \Delta t\}$ is satisfied.

For the modelling of the process of apoptosis and proliferation, we make the following choices. First, the variables are computed as before, thereafter it is checked if the requirements for proliferation or apoptosis are satisfied. With a random number from the uniform distribution it is decided if this action should take place. If so, this cell goes into the respective process, while the other cells move as determined before.

When a cell dies, the cell starts to shrink and its cell membrane becomes more permeable. When the contents of the cell that just died, diffuses through the extracellular space, other processes might be initiated. The endothelial cells (which can be seen as the spine of the blood vessel) grow towards this location. This is an important process in the process of tumor growth. In our simulations, the cell immediately disappears when it dies and hence it is immediately removed from the cell list.

When a cell divides, it first grows and then splits into two. In the simulation, this will happen in one time step. The cell splits into two, so that the cell center of the daughter cell is located on the boundary of the other cell. On which side the new cell comes will be determined randomly in the model.
We consider a cell that met the requirements for division. Let (x, y) be the location that would occur normally after incorporating the displacement via Equation (4.3). The division results in two cells, c_{d1} and c_{d2}, that will be allocated as below:

$$\theta \sim \mathrm{U}[0, 2\pi] \quad \Rightarrow \quad c_{d1} = \begin{cases} x_{c_{d1}} = \frac{R}{2}\cos(\theta) + x \\ y_{c_{d1}} = \frac{R}{2}\sin(\theta) + y \end{cases}$$
$$c_{d2} = \begin{cases} x_{c_{d2}} = \frac{R}{2}\cos(\theta + \pi) + x \\ y_{c_{d2}} = \frac{R}{2}\sin(\theta + \pi) + y \end{cases}$$

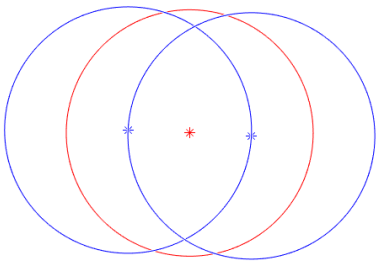


Figure 4.1: Example of allocation of cells after division, the blue cells are the new cells based on the location of the initial cell (red).

4.3. Finite elements implementation for concentration

The model is further extended with the influence on concentrations in the substrate. The displacement depends on the gradient of this concentration:

$$dX_i(t) = \kappa_i M(x_i)\hat{z}_i\, dt + \sqrt{2D}\, dW(t) + s_c \nabla c(t, x_i)\, dt, \tag{4.5}$$

where c is a function determining the concentration depending on t and x and s_c is the chemotactic sensitivity. If $s_c > 0$ then one speaks of positive chemotaxis, in which a cell is stimulated to move towards the gradient of a chemical. Whereas $s_c < 0$ corresponds to negative chemotaxis, in which a cell moves away from a gradient of a chemical. Negative chemotaxis is important in the migration away from toxic agents.

The general form of transport-reaction equations are:

$$\frac{\partial c}{\partial t} - \nabla \cdot J = f(t, x, c) \qquad + \text{ initial and boundary conditions,}$$

where J is the flux term and $f(t, x, c)$ the reactive term.

In our model, two concentration fields are used: Platelet Derived Growth Factor (PDGF) and Transforming Growth Factor β (TGF-β) [21]. The system for PDGF is given by:

$$\begin{cases} \frac{\partial c_P}{\partial t} - D_{c_P}\Delta c_P = 0\ ,\ t>0\ ,\ x\in\Omega, \\ D_{c_P}\frac{\partial c_P}{\partial n} + \kappa c_P = 0\ ,\ t>0\ ,\ x\in\partial\Omega, \\ c_P(0,x) = f(x), \end{cases} \tag{4.6}$$

where $f(x)$ is the initial concentration field (described in Section 4.3.1) and the system for TGF-β by:

$$\begin{cases} \frac{\partial c_\beta}{\partial t} - D_{c_\beta}\Delta c_\beta = \kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)\ ,\ t>0\ ,\ x\in\Omega, \\ D_{c_\beta}\frac{\partial c_\beta}{\partial n} + \kappa c_\beta = 0\ ,\ t>0\ ,\ x\in\partial\Omega, \\ c_\beta(x,t) = 0. \end{cases} \tag{4.7}$$

The Finite Element Method is used to compute the solutions in the domain. The derivations of the Galerkin equations for these systems are given in Appendix A. The systems that need to be solved have the form $M\frac{dc}{dt} + Sc = f$. This system is solved with the Euler-Backward method. By setting $R(t,c) = \frac{dc}{dt}$, we can write

$$\begin{aligned} c^{n+1} &= c^n + \Delta t R(t^{n+1}, c^{n+1}), \\ c^{n+1} &= c^n + \Delta t M^{-1}(f^{n+1} - Sc^{n+1}), \\ (M+\Delta t S)c^{n+1} &= Mc^n + \Delta t f^{n+1}. \end{aligned}$$

Solving this system gives the solution of the concentration at time t^{n+1}.

4.3.1. Concentration field descriptions

The concentration TGF-β is assumed to be zero over the whole domain, as there are no macrophages initially. The concentration field of PDGF is initially set by the formula: $c_P(0,\mathbf{X}) = \omega(\mathbf{X})c^\omega$ [21], where

$$\omega(\mathbf{X}) = \frac{1}{16}\left(1+\tanh\left(\frac{s_1-\mathbf{X}}{\psi}\right)\right) * \left(1+\tanh\left(\frac{s_1+\mathbf{X}}{\psi}\right)\right) * \left(1+\tanh\left(\frac{s_2-\mathbf{Y}}{\psi}\right)\right) * \left(1+\tanh\left(\frac{s_2+\mathbf{Y}}{\psi}\right)\right),$$

s_1 = length vertical part of wound,

s_2 = length horizontal part of wound,

and $\psi = \frac{s_2}{10}$.

ψ is a factor to tune the transition at the wound boundary, as this is zero outside the wound and goes up to c^ω, the chosen maximum amount of PDGF, in the wound area. The hyperbolic tangent functions in $\omega(\mathbf{X})$ provide a smooth transition between the wound and the undamaged area. As this transition is quite sharp, this would locally cause a very large concentration gradient. To prevent unrealistic displacement of the cells in reaction to this gradient, the displacement of cells is not directly correlated with the gradient anymore. The displacement caused by chemotaxis is set to

$$dx_{\text{conc}} = v_{\text{cell}}\left(\frac{\nabla c}{\|\nabla c\|_2}\right)dt\ ,\quad \text{for } \left(\frac{\partial c}{\partial x}\right)^2 \text{ and } \left(\frac{\partial c}{\partial y}\right)^2 \text{ large enough.} \tag{4.8}$$

By 'large enough' in Equation 4.8, we mean that the gradients should be non-zero and furthermore, not be rounded to zero when they are squared. The precise value of the smallest number that can be used without problems depends on the precision that is used in the program. Whenever the gradient in both directions is zero, we need to make adjustments to this formula in order to prevent undefined behavior by division by zero, thus a small, non-zero constant is added to the denominator. To prevent problems when the norm becomes zero, we use an alternate approximation than used in Equation 4.8. In this case we approximate the norm as

$$\|\nabla c\|_2 \approx \sqrt{2}\ \max\left(\left|\frac{\partial c}{\partial x}\right|, \left|\frac{\partial c}{\partial y}\right|\right). \tag{4.9}$$

In the last term, we are not squaring the derivative anymore and thus prevent the norm from becoming zero.

The speed of the fibroblasts and macrophages depends on the portion of receptors, r, that are bound to molecules as follows [21]:

$$v^i_{cell} = v * S(r^i) * (1 - S(r^i)), \qquad \text{where } S(r) = \frac{1}{2}\left(1 + \sin\left((r - \frac{1}{2})\pi\right)\right). \tag{4.10}$$

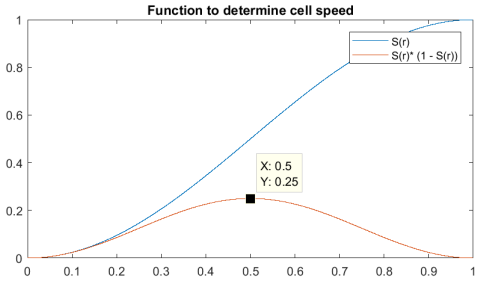


Figure 4.2: Functions $S(r)$ and $S(r) * (1 - S(r))$.

The cells attain their maximum speed when half of the receptors are bound, as can be seen in Figure 4.2. Furthermore, the speed is zero when no receptors are bound, which is the case for new cells. The portion of bound receptors for macrophages, r^i_{mp} and fibroblasts, r^i_f , changes over time as

$$\frac{dr^i_{mp}}{dt} = -\gamma^u_{mp} r^i_{mp} + \gamma^b_{mp} c_P(\mathbf{X}, t)\left(1 - r^i_{mp}\right), \tag{4.11}$$

$$\frac{dr^i_f}{dt} = -\gamma^u_f r^i_f + \gamma^b_f c_\beta(\mathbf{X}, t)\left(1 - r^i_f\right), \tag{4.12}$$

where γ^u and γ^b are the unbinding and binding rates respectively [21]. These are the rates for the PDGF molecules in the case of macrophages and the rates for TGF-β molecules for the fibroblasts.

4.3.2. Concentration gradient at cell centers

Solving the systems of PDGF and TGF-β gives us the concentrations on the grid vertices of the Finite Element mesh. For the displacement of the cells (see Equation 4.5), the concentration gradient is required at specific locations, namely at the cell centers.

The concentration gradient is constant over an element, as we are using linear basis functions. For the concentration on a location x in a triangle with vertices p_1, p_2, p_3 we can write

$$c(t,x) = \sum_{l \in \{p_1,p_2,p_3\}} c(t,x_l)\phi_l(x),$$

$$\nabla c(t,x) = \sum_{l \in \{p_1,p_2,p_3\}} c(t,x_l)\nabla\phi_l(x) = \sum_{l \in \{p_1,p_2,p_3\}} c(t,x_l)\begin{bmatrix}\beta_l \\ \gamma_l\end{bmatrix}.$$

For the Finite Element algorithm we need to know in which triangle a specific cell is located. Finding the three nearest nodes, does not naturally lead to the right triangle or even a triangle that exists in the mesh. Therefore, within the program it is checked whether a triangular element contains a cell.

Let p be a $n \times 2$ dimensional vector containing all x and y locations of the cells. Furthermore, consider a triangle T with vertices $\{(p1_x, p1_y), (p2_x, p2_y), (p3_x, p3_y)\}$. These points are ordered counter-clockwise (as is done automatically while making a Finite Element mesh). Define

$$\Delta = p2_x p3_y - p2_y p3_x + p1_y(p3_x - p2x) + p1_x(p2_y - p3_y),$$

$$s_{check} = \frac{1}{\Delta}\left[p1_y p3_x - p1_x p3_y + (p3_y - p1_y)p(i,1) + (p1_x - p3_x)p(i,2)\right],$$

$$\text{and} \quad t_{check} = \frac{1}{\Delta}\left[p1_x p2_y - p1_y p2_x + (p1_y - p2_y)p(i,1) + (p2_x - p1_x)p(i,2)\right].$$

Here Δ is twice the signed area of T. A point $\big(p(i,1), p(i,2)\big)$ is located in T if $(s_{\text{check}} > 0)$, $(t_{\text{check}} > 0)$, $(1 - s_{\text{check}} - t_{\text{check}})$ are all true. This test is based on the barycentric coordinate system, where $s_{\text{check}}, t_{\text{check}}$ and $(1 - s_{\text{check}} - t_{\text{check}})$ are the barycentric coordinates [19].

5

Modelling on realistic scale

The model made during this thesis project follows the general structure of the wound healing model described in [21]. The most radical change is the addition of the strain energy densities in the model, which occur in other cell migration models as well [42]. In the first section, it is described how the strain energy densities are incorporated, such that it fits with the rest of the wound healing model. In the second section an overview of the parameters and their values is given. Finally, in Sections 5.3, 5.4 and 5.5 some implementations added to the mathematical model from Chapter 4 are described.

5.1. Combining two models

During the literature study [20], the influence of the strain energy densities was incorporated into the displacement formula as follows:

$$dX_i(t) = \kappa_i M(x_i)\hat{z}_i dt, \tag{5.1}$$

where κ_i is a parameter that can be adjusted for different types of cells. It is defined as $\kappa_i = \frac{\gamma_i R^3}{\mu F_i}$ for viable cells and is zero otherwise. Here, γ_i is the mobility of the cell (depending on the cell viability) and μ is the resistance parameter of the substrate friction.

When adopting the parameters of both models, the strain energy density had a dominating influence in the model. The displacement caused by strain energies was about seven times larger than the displacement caused by chemotaxis, which had a limited response by the implementation of cell speed (Equation 4.10). In order to match with the structure from [21], the magnitude of the displacement due to strain energy densities will be adjusted to depend on the cell speed as described in Equation 4.10 as well. In the case of fibroblasts, this speed depends on the number of receptors that is bound to TGF-β and thus very dependent on the available concentration of this molecule. The attraction of cells by the strain energy densities and the repulsion of cells by mechanical force, should however also work in absence of any concentration of TGF-β.

The coefficient that determines the magnitude of the displacement by chemotaxis is $v_f^i = v * S(r^i) * (1 - S(r^i))$, with r_i the portion of receptors bound to molecules, as was stated in Equation 4.10. Given this, the coefficient for the energy is chosen to be $v_E^i = \frac{1}{2} v_f^i + v * 0.1$. Similarly to the magnitude coefficient for chemotaxis, we want to multiply the speed by a unit length vector that dictates the direction of the displacement. The displacement obtained from Equation 5.1 is therefore divided by the constants κ_i and M_0 before multiplying it by v_E^i. In the case all neighboring cells are outside the maximal detectable range, there will be no strain energy density or mechanical energy contributions and then $M = M_0$. In other cases, M will be larger, which will lead to more displacement during that iteration.

When at first, there are no macrophages, the concentration TGF-β will be zero and thus $v_f^i = 0$. The speed will be constant initially, then will increase during the entry of macrophages and finally will decrease again when the macrophage population depletes.

5.2. Input data

In this section, the parameters used in the simulations are described. Most parameters are adopted from [21] and converted to standard units. However, with the influences of the other model with strain energy densities, some parameters are not suitable for this model. In Table 5.1 all parameters are listed and special cases (labeled with *) are described in more detail in this section. Parameters with 'E' in the reference list were estimated in this thesis.

Parameter	Symbol	Value	Dimension	Standard dimension	Ref
Cell radius	R	2	μm	$2\cdot10^{-6}$ m	[9]
Substrate Elasticity	E_s	5	kPa	$5\cdot10^{3}$ kg/(m·s^2)	[9]
Cell Elasticity	E_c	0.5	kPa	$5\cdot10^{2}$ kg/(m·s^2)	[9]
Cell Force	F	$10\cdot10^{2}$	kg·μm/min^2	$2.78\cdot10^{-7}$ kg·m/ s^2	[4]
Maximal detectable range	ϵ	29.5	μm	$29.5\cdot10^{-6}$ m	[30]
Cell Mobility coefficient	β	1	min^{-1}	0.0167 s^{-1}	[4]
Friction coefficient	μ	0.2	–	0.2	[4]
Diffusion rate PDGF	D_{c_p}	0.00288	cm^2/day	$3.33\cdot10^{-12}$ m^2/s	[26]
Max. initial PDGF	c^w	10	ng/mL	10^{-5} kg/m^3	[26]
Diffusion rate TGF-β	D_{c_β}	0.0254	cm^2/day	$2.94\cdot10^{-11}$ m^2/s	[26]
Rate molecule leave domain	κ	1	1/mm	10^{3} m^{-1}	[21]
Constant for speed fibroblasts	ν	$2.5\cdot10^{-1}$	mm/h	$6.94\cdot10^{-8}$ m/s	[21]
Constant for speed macrophages	ν_m	$0.5\cdot10^{-1}$	mm/h	$1.388\cdot10^{-8}$ m/s	E
Constant for mp entry	$\beta_{M\Phi}$	5.2075	m^2/(kg·s)	$5.2075\cdot10^{4}$ m^2/ (kg·s)	E
Minimal conc for mp entry	$\beta^l_{M\Phi}$	$3.33\cdot10^{-9}$	kg/m^3	$3.33\cdot10^{-9}$ kg/m^3	E
Receptor binding rate PDGF	γ^b_m	$5\cdot10^{11}$	mm^3/(g·h)	$1.39\cdot10^{2}$ m^3/(kg·s)	[21]
Receptor unbinding rate PDGF	γ^u_m	1	1/h	$\frac{1}{3600}$ s^{-1}	[21]
Receptor binding rate TGF-β	γ^b_f	$5\cdot10^{11}$	mm^3/(g·h)	$1.39\cdot10^{2}$ m^3/(kg·s)	[21]
Receptor unbinding rate TGF-β	γ^u_f	1	1/h	$\frac{1}{3600}$ s^{-1}	[21]
Fibroblast diffusion	D_N	10^{-15}	m^2/s	10^{-15} m^2/s	E*
Macrophages diffusion	$D_{M\Phi}$	10^{-15}	m^2/s	10^{-15} m^2/s	E*
Death rate fibroblasts	d_N	$3.739\cdot10^{6}$	s	$3.739\cdot10^{6}$ s	[28]
Death macrophages constant	$d_{M\Phi}$	$7.2\cdot10^{12}$	mm^3/g	$7.2\cdot10^{6}$ m^3/kg	[21]
Magnitude TGF-β secretion	κ_{c_β}	$2.5\cdot10^{-14}$	g / (mm^3·h)	$2.082\cdot10^{-19}$ kg /s	E*

Table 5.1: Parameters for fibroblasts, macrophages and the concentration fields in the wound healing model. Parameters with reference E were estimated in this study and parameters labeled with * are explained in more detail in Section 5.2 .

The TGF-β concentration originates from the molecules that the macrophages secrete. The location where this happens is modelled with a Dirac Delta distribution. The parameter that determines the magnitude of the TGF-β secretion is $\kappa_{c_\beta} = 2.5\cdot10^{-14}$ g /(mm^3·h) in [21]. The area dependency is not convenient to use with this distribution and therefore we take a secretion rate per time unit. Converting $2.5\cdot10^{-14}$ g/(mm^3·h) to standard units gives $6.94\cdot10^{-21}$ kg/(m$^3\cdot$ s). We set the secretion per second to $2.082\cdot10^{-19}$ kg/s.

We want to have a relatively large random motility coefficient, as is prescribed by the parameters D_N and $D_{M\Phi}$ in [21], where the values are set to $9.25\cdot10^{-14}$ m^2/s. The problem is however that this will lead to the stability criterion being violated, that is the displacement of a cell is larger than half the cell radius. For now $D_N = D_{M\Phi}$ are set to 10^{-15} m^2/s, while the allowed time step in the stability check is set two times larger than as was described in the literature study [20].

From [28] we find that the death rate d_N is approximated by $3.739\cdot10^{6}$ s. Hence, the death probability rate of fibroblasts is set to $d_f = \frac{1}{d_N} = 2.674\cdot10^{-7}$ s^{-1}. The proliferation rate of fibroblasts is chosen to be $p_f = 2.5 * d_f = 6.68\cdot10^{-7}$. The death rate for macrophages depends on the concentration of PDGF at the location of the cell, such that the death rate is set to $d_{M\Phi}c_P(\mathbf{x})d_N$ for a macrophage at location $\mathbf{x}$. The probability rate is one divided by the death rate. The probability rates are used in the exponential distribution (Equation 4.4) to compute the probabilities of a cell undergoing proliferation or apoptosis.

The aforementioned parameters for apoptosis and proliferation are for the case that the cell is in absence of impingement with surrounding cells, in other words, the mechanical energy for that cell is zero. It seems natural that the probabilities on cell death and division are dependent on the current state of the cell. In [21] it is for example assumed that a cells needs a minimal distance of 4 times the cell radius with any other cell in order to proliferate. In the case that a cell has a lot of freedom of movement, it can easier proliferate, while the opposite is true for a cell that is compressed by surrounding cells. Therefore, it is assumed that the mechanical force has a positive relation with respect to the apoptosis probability and a negative relation with respect to the proliferation probability.

From [4] we adopt a similar principle for determining the parameters that influence the dependency of the proliferation and apoptosis probabilities on the mechanical energy. Instead of looking at values of $\|M\|$, the values of the mechanical energy are considered. The values of $\|M_{\text{mech}}\|$ are plotted in Figure 5.1 for simple scenarios. One cell is surrounded by 1 to 6 cells which all have a certain identical overlap with the middle cell. The limiting value for a cell being able to divide is chosen to be the magnitude of $\|M_{\text{mech}}\|$ for the case that four cells overlap around $1/4$ of the radius. Furthermore, we assume that the case where two cells have an overlap of $\frac{1}{4}R$ can be seen as an equilibrium between the probabilities rates for cell division and death.

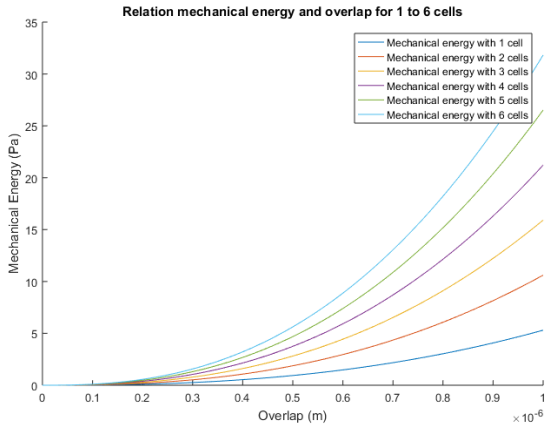


Figure 5.1: Mechanical energy in relation to a certain identical overlap of 1 to 6 surrounding cells.

Overlap	Mechanical Energy
1 cell × $\frac{1}{4}R$	0.9378 Pa
2 cells × $\frac{1}{4}R$	1.876 Pa
3 cells × $\frac{1}{4}R$	2.813 Pa
4 cells × $\frac{1}{4}R$	3.751 Pa
5 cells × $\frac{1}{4}R$	4.689 Pa
6 cells × $\frac{1}{4}R$	5.627 Pa
1 cell × $\frac{1}{2}R$	5.305 Pa
2 cells × $\frac{1}{2}R$	10.61 Pa
3 cells × $\frac{1}{2}R$	15.92 Pa

Table 5.2: Mechanical Energy for some numbers of cells with identical overlap of a $\frac{1}{4}$ or $\frac{1}{2}$ of the cell radius.

The proliferation probability for fibroblasts is assumed to be 2.5 times the apoptosis probability of fibroblasts, in absence of mechanical energy. With the values found from Figure 5.1, we can find the probability rate depending on the mechanical energy. In this simulation there is no proliferation of macrophages, solely the entry from the intravascular space to the extravascular space.
The probability rates per second are

$$p_d = \max(p_f - 1.78 \cdot 10^{-7} * M_{\text{mech}}, 0), \qquad \text{for fibroblasts,} \tag{5.2}$$

$$p_a = d_f + 3.56 \cdot 10^{-8} * M_{\text{mech}}, \qquad \text{for fibroblasts,} \tag{5.3}$$

$$\text{and } p_a = \frac{1}{d_{M\Phi}(c_P(\mathbf{x}) + 10^{-9}) d_N} + 3.56 \cdot 10^{-8} * M_{\text{mech}} \qquad \text{for macrophages.} \tag{5.4}$$

The actual probabilities are then given by $P(t < \tau < t + \Delta t) = 1 - \exp\{-p_i \Delta t\}$, $\in \{p_d, p_a\}$. The probabilities are visually displayed in Figure 5.2. The death rate for macrophages varies with the concentration of PDGF, therefore a few possible relationships are plotted in the Figure 5.2.

As the death rate of the macrophages depends on the concentration PDGF, this will mimic their influence during the wound healing process. Initially, the concentration PDGF is high, which will cause a low death rate during this period. After a while, the PDGF will decrease until no more macrophages are entering from the intravascular space. Moreover, the death rate will increase, what will result in depletion of the macrophages in the domain eventually. In Equation 5.4 the influence of the concentration is the factor $(c_P(\mathbf{x}) + 10^{-9})$, where the value 10^{-9} prevents the apoptosis probability rate from becoming too large when the concentration PDGF goes to zero.

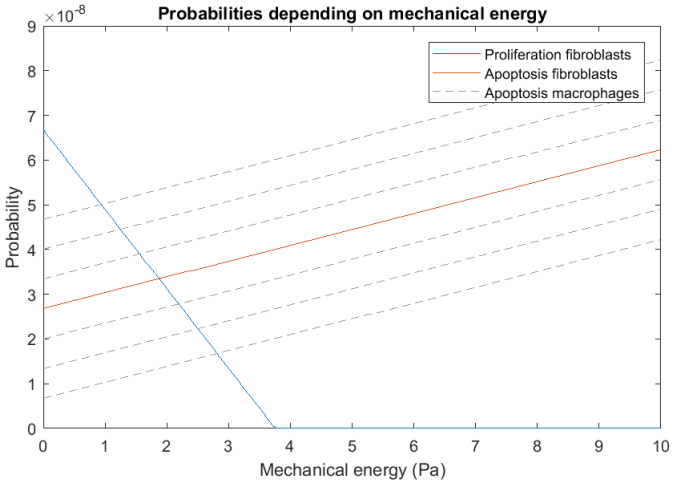


Figure 5.2: Visual representation of the probabilities for proliferation of fibroblasts and apoptosis of fibroblasts and macrophages.

5.3. Fibroblast density

In order to make a good initial scenario, an estimate for the fibroblast density in undamaged tissue is needed. Moreover, this 'normal' density can be used to quantify in which part of the tissue the fibroblast density is restored. In [25] an image analysis method for counting fibroblasts was investigated. Here it was found that the number of fibroblasts ranged from 2100 to 4100 per mm^3 of undamaged tissue. Furthermore, it was found that in samples of hypertrophic scars the fibroblast density could in some locations reach more than double the amount of fibroblasts found in other parts.

In the case of proliferative scarring, that is overhealing, the fibroblast density overshoots the normal density, due to a failing apoptosis rate. Normally, fibroblasts reach the wound during the second or third day and have a maximum population between the seventh and fourteenth day [13]. The increase of fibroblasts is initially caused by migration from surrounding undamaged tissue and later from proliferation of fibroblasts.

For the initial set-up of the scenario, we want to set a normal fibroblast density in the undamaged area and have no fibroblasts in the wound area. The domain in the model is only in 2D, therefore it is difficult to determine a normal density as most experiments state a density expressed in mm^3 or mL. The initial density is set to 10^9 fibroblasts per m^2. To make an initial setting, we determine the x and y coordinates of the fibroblasts with two random numbers, where the ones that fall in the wound region are being left out. This procedure is repeated until we have reached the predefined density of fibroblasts in the undamaged area.

As described in Section 2.1, the amount of cells is limited by different influences from the environment. It depends on the modelling scenario, which limiting factor is important in the model. For very dense populations, as can be the case for tumor growth, the space in the domain is the limiting factor. In a normal scenario, without the extra stimuli for the proliferation of cells that occurs for tumor growth, the cell growth halts before space becomes a problem. The dependency of the proliferation and apoptosis on the mechanical energy does not limit the population growth as would be expected in normal scenarios. For modelling the limiting factor of nutrients, the magnitude of the strain energy density per cell (without mechanical energies) is taken as a measure for the compactness of its neighborhood.

A reference value of the energy density is defined as $M_{ref} = 4M_0 \exp(-2.5\lambda)$. This value is based on the strain energy that occurs for a cell that has four cells in its neighborhood, which are located around the cell with half a radius space in between. This means the distance between the cell centers becomes two times the radius plus the half radius in between. The strain energy density becomes $M = M_0 + 4M_0 \exp(-2.5\lambda)$ according to Equation 4.1. We disregard the first term M_0, as this is included in the strain energy density for all cells. For every cell, the fraction $G = \frac{M-M_0}{M_{ref}}$ is computed and used to influence the proliferation of fibroblasts. We distinguish between two scenarios, one where the concentration of TGF-β is sufficiently present and one where this is not the case. When more TGF-β is present in the domain, this gives an incentive to fibroblasts

to fill up the gap that is created by the wound, in this case the proliferation of fibroblasts should not be halted based on the amount of other cells in the neighborhood. However, if the concentration of TGF-β is (almost) zero, then the proliferation will be halted whenever the fraction $G > 1$.

5.4. Macrophages entry

The macrophages will enter via the wound boundary during the inflammatory phase, when plasma leaks from the extravascular space. In the model, the macrophages will enter the domain as long as the concentration PDGF exceeds a certain threshold value β^l_{mp} [21]. Initially the concentration PDGF will be high in the wound and thus this criterion will be satisfied. After a while, the PDGF molecules will have diffused through the domain, such that the concentration does not exceed the threshold anymore.

The entry of macrophages is modelled via the description in Algorithm 1. A domain with size $[2 \cdot w, 2 \cdot h]$ of which the center is located at $(0,0)$ is considered. The probability that macrophage will enter the wound is dependent on the concentration PDGF. We determine an estimate of the average PDGF on the wound interface, $\bar{c}_p$, and use this value to find the number of macrophages that will enter per iteration. The estimate of the concentration PDGF is based on eight points along the wound boundary, as displayed in Figure 5.3. In these eight points, the concentration of PDGF is computed and subsequently, linear interpolation is used to determine the average along the wound boundary.

Algorithm 1: Macrophages entry

Parameters: $\xi \sim \mathrm{U}[0,1]$, w, h

1 $x = [-\frac{w}{2}, -\frac{w}{2}, 0, \frac{w}{2}, \frac{w}{2}, \frac{w}{2}, 0, -\frac{w}{2}]$
2 $y = [0, \frac{h}{2}, \frac{h}{2}, \frac{h}{2}, 0, -\frac{h}{2}, -\frac{h}{2}, -\frac{h}{2}]$
3 $d = [\frac{h}{2}, \frac{w}{2}, \frac{w}{2}, \frac{h}{2}, \frac{h}{2}, \frac{w}{2}, \frac{w}{2}, \frac{h}{2}]$
4 **for** $i \leftarrow 0$ **to** 7 **do**
5 $\quad c_p[i] = \text{findcp}\,(x(i), y(i))$ `// function that maps the concentration to` $(x(i), y(i))$
6 $c_p[8] = c_p[0]$
7 **for** $i \leftarrow 0$ **to** 7 **do**
8 $\quad \bar{c}_p += \frac{c_p[i]+c_p[i+1]}{2} * d[i]$
9 $\bar{c}_p = \bar{c}_p / (2 * w + 2 * h)$
10 **if** $\bar{c}_p > \beta^l_{mp}$ **then**
11 $\quad N_{mp} = \text{getPoisson}(\beta_{mp} * \bar{c}_p * (2 * w + 2 * h) * dt)$
12 $\quad$ **for** $i \leftarrow 0$ **to** N_{mp} **do**
13 $\quad\quad z = \xi * (2 * w + 2 * h)$
14 $\quad\quad$ `// place macrophage on wound boundary based on value of` z.
15 $\quad\quad$ mp_push() `// function to move cells to a location with enough space.`

The number of macrophages that enters the domain is determined by a Poisson distribution with mean $\beta_{mp} * \bar{c}_p * (2 * w + 2 * h) * dt$. Basically, this mean consists of a coefficient tuning the entry probability times the total concentration along the wound times the time step. If the lower threshold criterion is satisfied (line 10 of Algorithm 1), then we find the number of macrophages from a Poisson distribution. Finally the location of the added macrophages is determined randomly along the wound interface.

After obtaining a random value for the location of the macrophage, it is checked if placement in this location will not cause any problems. If there is contact with other cells, the macrophage will be displaced in the direction is determined by magnitude of the mechanical energy. This procedure is repeated until the minimal overlap with any other cell is greater than R or if the maximum of 10 repetitions is reached. In this way, the excessive displacement of the macrophage in the subsequent iteration due to a high mechanical energy is prevented.

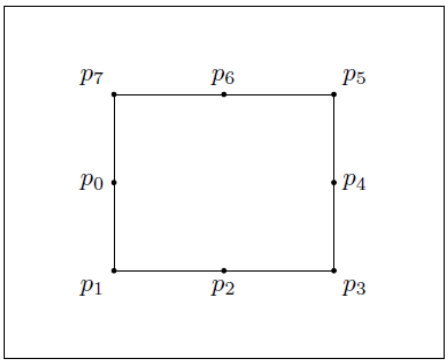


Figure 5.3: Schematic view of points used for determining the average concentration PDGF along the wound boundary, which is displayed as the rectangle with corners p_1, p_3, p_5, p_7.

5.5. Domain boundaries

Only the cells in a small part of the dermal layer are considered during the simulations. At the boundaries, Robin boundary conditions are imposed. The mathematical formulation of this is stated in Section 4.3. With a constant rate factor κ the concentrations PDGF and TGF-β leave the domain. This boundary condition assumes that concentrations can flow into an adjacent area and allows us to model only the area of interest, the wounded area, instead of the whole skin.

During the literature study [20], this domain was modelled as a closed region where cells were not able to move out. A more realistic approach would model a part of the domain, while allowing interaction with the surrounding parts of the dermal layer. Furthermore, the walls sometimes led to problems for the cell interactions [20]. This happened when cell division took place at the corners for crowded domains. Furthermore, with the extension of the model a larger time step is used, such that the random walk is allowed to be larger and can result in cells that have a displacement that crosses the boundary of the closed region.

To allow the interactions with adjacent regions, the wall forces are removed from the model. For every cell that leaves the domain over the horizontal boundary, a new cell will enter on the other horizontal boundary at a random location. A similar procedure will be applied for the cells leaving over a vertical boundary. Consider a domain with size $[2 \cdot w, 2 \cdot h]$ of which the center is located at $(0,0)$. Given a cell that has left the domain and has location (x_c, y_c), then the new cell will be located via procedure as described in Algorithm 2.

Algorithm 2: Cells boundary crossing

Parameters: $\xi \sim \mathrm{U}[0,1]$

if $(|x_c| > w)$ **then**
  if $(x_c > w)$ **then** $x_c = x_c - 2 \cdot w$
  else $x_c = x_c + 2 \cdot w$
  $y_c = -h + \xi \cdot 2 \cdot h$
else if $(|y_c| > h)$ **then**
  $x_c = -w + \xi \cdot 2 \cdot w$
  if $(y_c > h)$ **then** $y_c = y_c - 2 \cdot h$
  else $y_c = y_c + 2 \cdot h$

As is stated in pseudocode of Algorithm 2, the cell that leaves the domain will be overwritten with the new cell that enters. All properties (for example, the life span) of the old cell are transferred to the new cell.

Future extensions of this concept could include the extra inflow of cells as response on the wounding. Additionally, one could think of implementing the entry and outflow of cells over the whole domain, as we place the modelled domain in a 3D setting [21].

6

GPU computing

More and more applications, such as large data amounts or systems, require large computational power. In a biology setting, there are models for which this is important as well. Classic sequential computers can be classified as the SISD (single instruction, single data) type. If multiple processors can be controlled, we get the SIMD (single instruction, multiple data) type. Most computers, however, have the MIMD (multiple instruction, multiple data) type. Often, a mix of the types can occur within a computer. In this chapter, the possibilities, in particular by using the GPU, to make a more efficient code are investigated. Furthermore, the speed-up between the MATLAB and C++ program is shown in Section 6.5.

6.1. High performance computing

High-performance computing can be used without adjusting much in the CPU code. For example, there exists OpenMP, which is software that helps to express shared memory parallelism. A sequential program can be taken and some parts, such as loops, can be parallelized. This approach is called the fork/join model and can be easily applied without needing to rewrite much in the sequential code. It is therefore widely used, but also has some considerable drawbacks. GPUs can often obtain better performance [27].

It is worth noting that an alternative parallelization of Agent-Based Models by means of grid computing, that is using the resources of multiple computers, would not scale well: the running time could not be reduced below a fixed threshold [27]. This is caused by memory bandwidth restrictions, by which the time that is needed per data item to move it to another location (to another processor) is meant.

6.2. GPUs

General-purpose GPUs (GPGPUs) are designed to execute similar instructions for different data inputs (SIMD processor), so called data-parallelization. A GPU has a fixed number of processors, called streaming multiprocessors (SM). Every one of those streaming multiprocessors exists of eight scalar processors.
Every thread executes on a scalar processor. A number of threads are bundled together in blocks. The blocks are parts of the grid. When issuing a task, the SM receives a number of blocks. The threads in this block are split up in pieces: warps. As sometimes they cannot all be executed simultaneously, they are executed per one or multiple warps.

6.2.1. CUDA and OpenCL

There are several software frameworks that can be used to program on the GPUs. CUDA (Compute Unified Device Architecture) is the leading software for proprietary GPUs and can operate on Nvidia's GPUs. Alternatively OpenCL can be used, which is the leading open source software. The large advantage of OpenCL is that it is hardware-independent. The execution and memory hierarchy models are similar to CUDA. Differently from CUDA, the kernel compilation phase is performed at run-time. CUDA appeared to be more efficient in reducing register usage, which affects the number of concurrently executed threads. Further, the kernel launch cost is around 9 times larger than CUDA (only important for kernels with short execution time) [27].

CUDA seems to be the most often used option in academia, for example in [5], [27], [34], [35]. However sometimes it is mentioned that OpenCL could have been suitable as well. While the improvement from the CPU to GPU is a significant number of times faster, the improvement from OpenCL to CUDA is reported to be a small percentage. In [24] it is described that OpenCL takes more programming effort than CUDA, but the performance and energy consumption were depending on the tested application.
In [11] an extensive comparison is made between OpenCL and CUDA, both on NVIDIA's GPUs. Their main conclusion was that CUDA performs 30% better than OpenCL at most of their applications, due to an unfair comparison. An example they give is that OpenCL avoids features, such as texture memory, for portability. After removing the use of texture memory in CUDA, the performance was comparable. The compiler has influence on the amount and type of operations that are executed, this caused differences between OpenCL and CUDA as well. They show that equal performance can be obtained by changing the code appropriately, such that the comparison is fair. Furthermore, CUDA and OpenCL are operate quite similar, so translation from one to the other should be doable. There are some differences in the terminology of which the main ones are stated in Table 6.1.

CUDA terminology	**OpenCL terminology**
Global memory	Global memory
Constant memory	Constant memory
Shared memory	Local memory
Local memory	Private memory
Thread	Work-item
Thread - block	Work-group

Table 6.1: Comparison of CUDA and OpenCL terminology from [11].

The program should be useful for users without specific knowledge of the GPU. While OpenCL has the advantage of being operable on multiple GPUs, tuning of the algorithm is necessary in order to obtain good performance. Having a CUDA program only being applicable on NVIDIA's GPUs, lowers the need for adjusting the code (adapting to the NVIDIA version might improve performance, but the code should work fine without adjusting as well).

6.2.2. Programming on the GPU

The programmer has the responsibility to decide how many threads are given per block and how many blocks will be used in the grid. An important factor to consider while making this distribution is the memory on the GPU. There are different memories, given below and ordered from fast to slow access (for CUDA).

- Registers - Exclusively accessible to threads
- Shared memory - Exclusively accessible to all threads in a block
- Texture memory - Read-only for the streaming multiprocessor
- Global memory

In a scenario where very fast access is required, we want to use registers only. Using many blocks can result in large numbers of parallel operations, so that there is not enough register space to put all the outputs. Instead, we can decide to use only a limited number of blocks and make sure every block has enough registers.

Dynamics on cell level seem suitable for GPU programming, as many of the computations have to be done for each cell. Furthermore, each cell needs to address nearby cells to check if they collide or else to find the strain energy density. CPUs work more according to task-parallelization, where different tasks are executed at the same time. A CPU code can therefore not simply be taken over to be used on a GPU. It has a different architecture and set of functionalities and accordingly the code needs to be rewritten before usage. Tasks that are very short, process little data or are different from each other will not perform much better using a GPU [7].

Serial code is executed on the host (CPU), while parallel code is executed on the device (GPU). Kernels operate with the memory on the device. It is therefore necessary to send data and allocate this on the device. After

the computations, data needs to be send back to the host. The data transfer is a synchronous operation, this means that the transfer will only take place if all CUDA operations started previously have ended [18]. There are also asynchronous operations. For example, when a kernel is launched on the GPU, the CPU continues in the meantime with the code on the next lines. Ideally, the computational work is divided among the CPU and GPU in such a way, that the waiting times for either CPU or GPU are small.

IEEE754 is the technical standard for computations with floating point numbers. It assures that when doing computations, the number is rounded to the nearest floating point. NVIDIA GPUs have the implementation to do fused multiply-addition [40]. This is faster and more accurate than multiplying first and doing addition after, as prescribed by the IEEE754 standard.

6.3. GPU specifics for this project

In order to tune the performance of a program using a GPU, we need to be aware of the type of GPU that is used. The compute capability is a number that gives an indication of what kind of features are possible. The GPU used in this project has compute capability 6.1. For certain parts in the model, a minimal compute capability is required. An example is the 64-bit floating-point version of atomicAdd(), which is only supported by devices of compute capability 6.x and higher. In the Table 6.2 specific information about the GPU used in this thesis is given.

Device name:	GeForce GTX 1080
Compute capability – Major.minor:	6.1
Memory Clock Rate (KHz):	5505000
Memory Bus Width (bits):	256
Peak Memory Bandwidth (GB/s):	352.320000
Total global memory:	8114 MByte
Shared Memory per block:	48 kByte
Registers per block:	65536
Number of multiprocessors:	20
Warp size:	32
Max blocks per Multiprocessor:	32
Max Threads per Multiprocessor:	2048
Max threads per block:	1024
Max threads dimension:	1024 x 1024 x 64
Max grid size :	2147483647 x 65535 x65535
Mapped memory	Possible
Concurrent data transfers	Possible
Concurrent kernel execution	Possible

Table 6.2: CUDA device information.

The threads are scheduled per warp, which consists of 32 threads. Therefore, the chosen block size should always be a multiple of 32. The occupancy is the number of treads used on the multiprocessor relative to the maximal amount. With a maximum of 32 blocks per multiprocessor, we need at least $\frac{2048}{32} = 64$ threads per block to reach full occupancy. The choices 128 and 256 threads per block could work as well. In this case, the numbers of blocks that can be used is just lower, namely 16 and 8 respectively, while the occupancy remains 100%.

6.4. Tracking performance

To optimize performance, it is important to investigate which tools are available to measure this. According to [18], there are two options. First of all, there are the CPU Timers. In C++, we can for example use the `clock()` function or the `high_resolution_clock::now()` function from the chrono package. To mea-

sure the performance of some kernel execution on the GPU, we can set `t1 = someCPUTimer()` before the kernel launch and `t2 = someCPUTimer()` after. As kernel launches are asynchronous, it is necessary to put `cudaDeviceSynchronize()` after the kernel launch. The CPU is forced to wait and the kernel execution time is measured, instead of the kernel launch time. The disadvantage of `cudaDeviceSynchronize()`, is that the progress is stalled.

The second option is to use CUDA events. These timers are a more lightweight alternative to the CPU timers. Two CUDA events need to be created:

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
```

Before and after the kernel lauch, we put `cudaEventRecord(start)` and `cudaEventRecord(stop)`, respectively. At the end of the code, we need to make sure that the event has finished (`cudaEventSynchronize(stop)`). The elapsed time can be found with the command `cudaEventElapsedTime(&milliseconds, start, stop);` Furthermore, the profiler option can be used to look into more detail of the computation times. Within this profiler, a more specific part of the computing time of the kernel launch is given.

6.5. Speed-up and scalability

Although GPU computing is a powerful tool, it is good to be aware of the limits of the speed up as well. Depending on how the problem scales and on which parts can be parallelized, we can make an estimate of the expected speed up. According to [38], a fair comparison can be made if we consider the fastest sequential program on a sequential machine and compare this with the fastest parallel program on p parallel processors. There are two models, Amdahl's law and Gustafson's law, that express the effect of parallel computing.

Consider a simplified process where all computations are executed with the same speed. The speed-up is computed as the fraction between the time a serial program would take t_s, divided by the time a parallel implementation would take t_p. The amount of available parallel processors is p.

Amdahl's law [38]: In this model, the part of computations that can be done in parallel remains constant as the problem becomes larger. Let f be the fraction of operations that can be parallelized. The speed-up is

$$S = \frac{t_s}{t_p} = \frac{t_s}{\frac{t_s}{p} f + (1-f) t_s} = \frac{1}{\frac{1}{p} f + (1-f)} < \frac{1}{1-f}.$$

The problem is not scalable, as the extra speed-up gained decreases with the total number of processors. The fraction $S \to \frac{1}{1-f}$ when $p \to \infty$, where the speed-up does not depend on the number of processors anymore. For a number of parallel processors, every extra processor results in less speed-up than the previous extra processor did.

Gustafson's law [38]: In this model, the part of computations that can be done in parallel scales with increasing problem size. Now the fraction g that can be done in parallel is a fraction of the wall clock time. The speed-up is

$$S = \frac{t_s}{t_p} = \frac{(1-g) t_p + g p t_p}{t_p} = 1 - g + g p.$$

Computations following this model are scalable. The speed-up increases with a fixed amount with every extra processor, independently of the total number of processors.

To achieve a large speed-up, all possibilities of parallelization should be utilized. The amount of serial work is actually the limiting factor for the speed-up. Let X be the fraction of serial work that can be parallelized. Then the speed-up is $S = \frac{t_s}{t_p} = \frac{t_s}{(1-X) t_s + Y} < \frac{1}{1-X}$, where Y is the total work done in parallel. Even if this work Y becomes negligibly small by using many parallel processors, the speed-up can never be larger than $\frac{1}{1-X}$. For example, in a case where 90% can be done in parallel, the speed-up is limited by a factor of 10.

As the MATLAB code made during the literature study [20] was written mainly to research the model interactions, it can be assumed that it was not the fastest possible sequential program. However, it can still give an

indication of the speed up. It was seen that the amount of work in the part where the strain energy densities where computed increased on longer simulations. This indicates that the work that can be parallelized increases as well. In Figure 6.1 the computation work is split up in the strain energy density part and the Finite Element part for a wound healing simulation of 8000 iterations with a time step of 0.09 seconds. The Finite Element computations increase barely with the number of cells in comparison to the strain energy density part. The part of cell migration without the concentration fields does increase greatly with more cells. The work in the FEM part has approximately the same amount of work for different numbers of cells. Therefore, it is not scalable according to Amdahl's law. The cell migration depends on the cell number and has an increasing percentage of the total work.

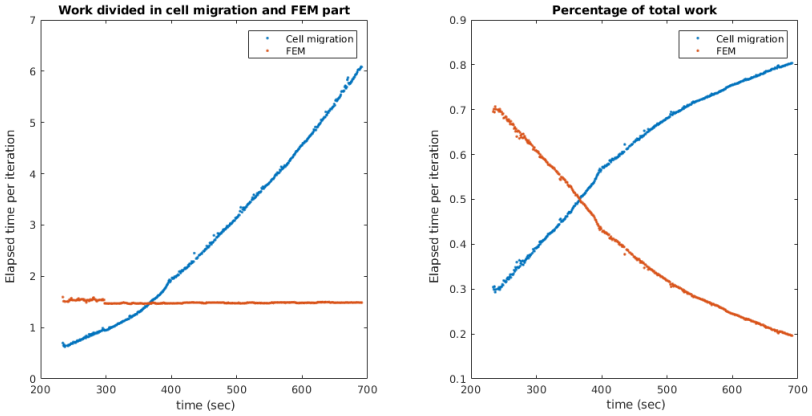


Figure 6.1: Computational work in MATLAB split up in the strain energy density part (cell migration without concentration influences) and the FEM element computations.

After converting the program to C++, the times of computation decreased greatly. Furthermore, since the matrices in the system do not change during the iterations, they have to be build only once. Removing the work for all the other loops sped up the work in the Finite Element part. The work for the strain energy density and the Finite Element program are closer to each other now (see Figure 6.2). For both parts, there is a strong dependency on the number of cells. For the strain energy density, this increases more than linear and will quickly become a problem for larger problem sizes. The modelling scenario is scalable as described by Gustafson's law, because the work increases with the problem size. Moreover, the speed of the Finite Element program depends on the chosen grid: if the number of nodes in the mesh is chosen larger, this will require more computational power as well.

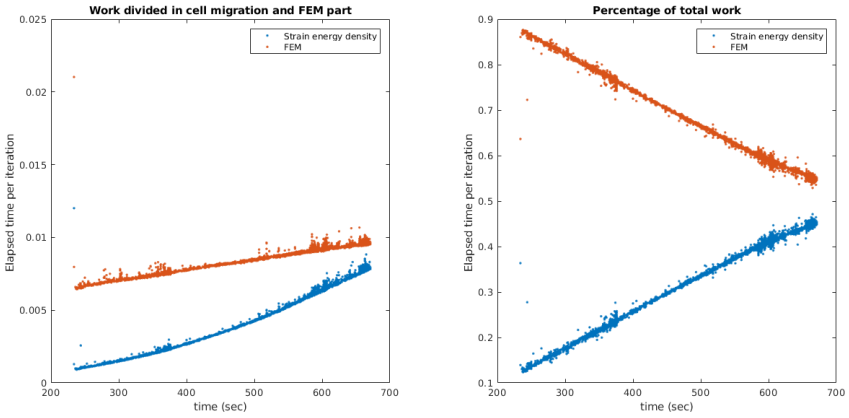


Figure 6.2: Computational work in C++ split up in the strain energy density part (cell migration without concentration influences) and the FEM element computations, where the stiffness and mass matrix are only computed in the first iteration. The domain size is [-50 μm, 50 μm] x [-40 μm, 40 μm].

7

Programming of wound healing model

In the development of the code for a model that uses the GPU and has possibilities to visualize the results, many sub-steps had to be taken. This chapter gives an overview of the choices made during the programming of the model. First of all, a C++ program had to be created, that could model the same as the previous MATLAB code [20]. In order to check for bugs, the implementation of the GPU was omitted initially. In Section 7.1.1, the results from MATLAB and C++ are compared and seem to be qualitatively the same, as far as we can conclude that based on stochastic data. Thereafter, the model was not further extended in MATLAB, but only in C++. In the C++ code, there are two implementations. One implementation is a CPU program and the other one uses the GPU in certain parts to speed up computations.

7.1. From MATLAB to C++

After the construction of a model in MATLAB during the literature study [20], a similar model was made in C++. As some pieces of code were rewritten and direct conversion from MATLAB to C++ was not possible for some parts, the results were compared in order to make sure it was simulating the same process. The cell dynamics depend on stochastic processes, therefore we need to compare the results statistically, as described in Section 7.1.1.

7.1.1. Interpretation of results

Consider a family X_j of independent identically distributed (iid) stochastic variables. The central limit theorem gives us $\sqrt{n}(X_g - \mu) \sim N(0, \sigma^2)$, where n is the sample size, X_g the sample average, μ the expected value and σ the standard deviation. With this information, the Monte Carlo error can be predicted.
We can construct confidence intervals, based on a certain number of simulations. If we have a sample size n with data X_i, then we can define:

$$\text{the sample mean } \bar{X} = \frac{\sum X_i}{n},$$

$$\text{the sample standard deviation } s_x = \sqrt{\frac{\sum (X_i - \bar{X})^2}{n-1}}.$$

Suppose that we are interested in estimating the mean μ_x of our output variable x. The sample mean can be used as an (unbiased) estimator, however, we are interested in the variability of this estimate. The variability is quantified by the variance of the error $\bar{X} - \mu_x$. As the data X_i are independent, random samples, the variance of the sum of these samples is the sum of the variances, thus we obtain

$$\text{Var}(\bar{X} - \mu_x) = \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} n\sigma_x^2 = \frac{\sigma_x^2}{n}, \text{ see [41].} \tag{7.1}$$

We use the sample variance s_x^2 as an estimator for σ_x^2. The Monte Carlo error is the square root of Equation 7.1, thus becomes $\frac{s_x}{\sqrt{n}}$. The error decreases with a factor $\sqrt{n}$ when increasing the number of Monte Carlo simulations [41].

For a chosen confidence interval of $100(1-\alpha)\%$ (usually 90, 95 or 99 %), the corresponding Z-value can be found for the number $1-\frac{\alpha}{2}$. The desired confidence interval is $\left[\bar{X} - Z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}, \bar{X} + Z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}\right]$.
Two confidence intervals that overlap, are not necessarily an indication for the two processes to be significantly similar. However, if two confidence intervals do not overlap, this indicates that the two processes are significantly different.

In Figure 7.1 eight MATLAB simulations and fifteen C++ simulations are displayed. The model includes a concentration field and was run for 4000 iterations with a time step of 0.09 seconds. The confidence intervals overlap for these data sets. Furthermore, it is not possible to distinguish the MATLAB runs from the C++ runs without the color indication. It seems therefore plausible that they are simulating processes according to the same underlying dynamics.

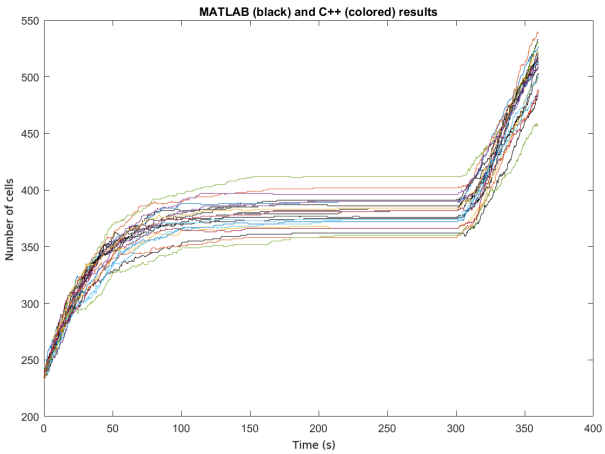


Figure 7.1: Different simulations in C++ (colored) and MATLAB (black) tracked over time in one figure.

7.1.2. Comparison of simulation time

The program as described in the literature study [20] has been implemented in both MATLAB and C++, of which some example iterations are displayed in the previous section. To perform the wound healing simulation for 4000 iterations MATLAB needs about 3.5 hours to complete the computations. In the C++ program, this takes only about three minutes. This is approximately 70 times faster than the MATLAB program.

Of course, the comparison is not entirely fair. The MATLAB program was a first program to investigate the behavior and was not very sophisticated in terms of efficiency. When making the C++ program, the knowledge about the model was used to write a more efficient code. The simulation time of C++ mentioned above is the run-time only. On the other hand, MATLAB uses grid division to speed up computation, while C++ simply checks all interactions with all other cells.

The C++ code uses different libraries to model the wound healing. For the Finite Element computations (in particular building sparse matrices and solving the system) the library `Eigen` is used [15]. To generate the initial mesh, the `Triangle` software is used [33]. The Standard Library vector is used whenever possible, as this is easy to use and optimized to work very well.

Another measure that speeds up the C++ code is that the mass and stiffness matrix are computed only in the first iteration, as they are constant in this problem. Furthermore, the system $Ax = b$ is solved by a Sparse supernodal LU factorization. This happens in two steps: compute() and solve(). During the compute() step, the matrix $A = M + S * dt$ is factorized. This factorization also only needs to be done in the first iteration, as M, S and dt will not change. The right-hand side will change, thus the solve() step needs to be executed every iteration.

For a domain of size 200 x 160 μm, the effect of the GPU is graphically shown in the Figures 7.2-7.4. The base scenario is the code that is optimized as described above (only computing part of FEM once), which runs

solely on the CPU. In this scenario, shown in Figure 7.2, we can see a strong dependency of the computational work on the number of cells. During the literature study, the exponential behavior of the computational work in relation to the number of cells indicated to become problematic. Therefore, the use of the GPU was proposed as a measure to make the program more efficient.

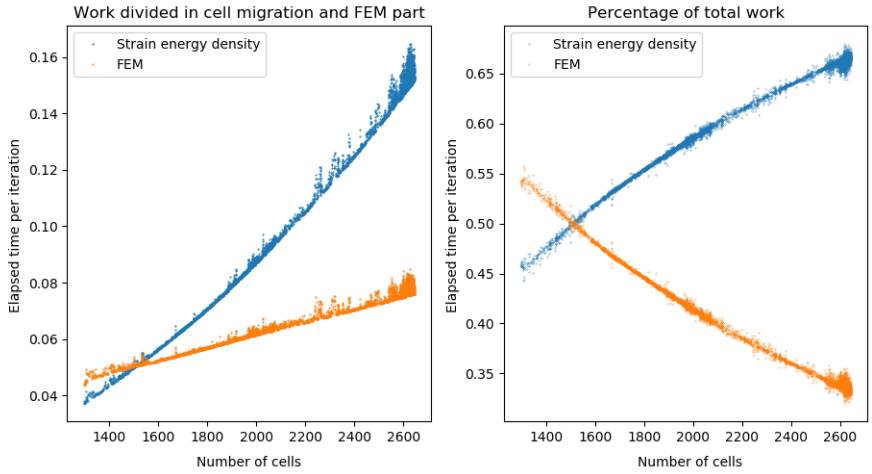


Figure 7.2: Elapsed time per iteration for a domain of 200 x 160 μm, with a time step of 0.09 s and run for 10,000 iterations performed on the CPU.

In Figure 7.3 the increase of the computational work is reduced by using the GPU to parallelize the computations of the strain energy densities. The computation time for the Finite Element part is the same as in the base scenario (Figure 7.2). The elapsed time per iteration for the strain energy densities keeps on increasing with the number of cells, but this change is much smaller than before. The Finite Element computations are now the bottleneck with respect to efficiency. Within this part, the mapping of the concentration gradient to the cells was parallelized. In Figure 7.4 the improved elapsed times can be seen for the FEM part. The elapsed times for the cell energy densities are the same with respect to the previous scenario. It is noteworthy that the dependency on the number of cells is almost removed for this domain. This is caused by the fact that the computation of the right-hand side vector and solving the FEM system depend on the concentration mesh, hence does not depend on the number of cells. The actual computation of the concentration gradient does naturally depend on the cells. Shifting this part to the GPU, however, prevents the large increase with the number of cells.

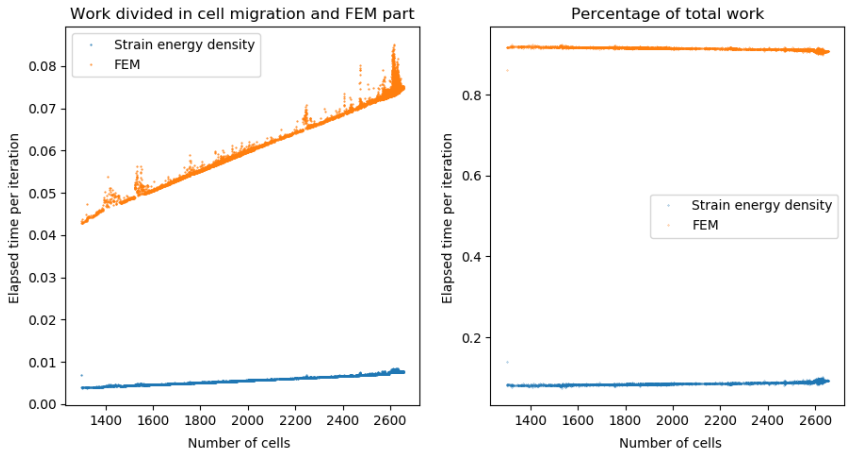


Figure 7.3: Elapsed time per iteration for a domain of 200 x 160 μm, with a time step of 0.09 s and run for 10,000 iterations where the computations of the strain energy densities is performed on the GPU.

With the current usage of the GPU, we see in Figure 7.4 that the strain energy density computations depend a bit more on the number of cells than the FEM computations. If the domain is extended (while keeping the same coarseness of the grid), the iteration time spend in the cell energy density part might become larger than the time in the FEM part.

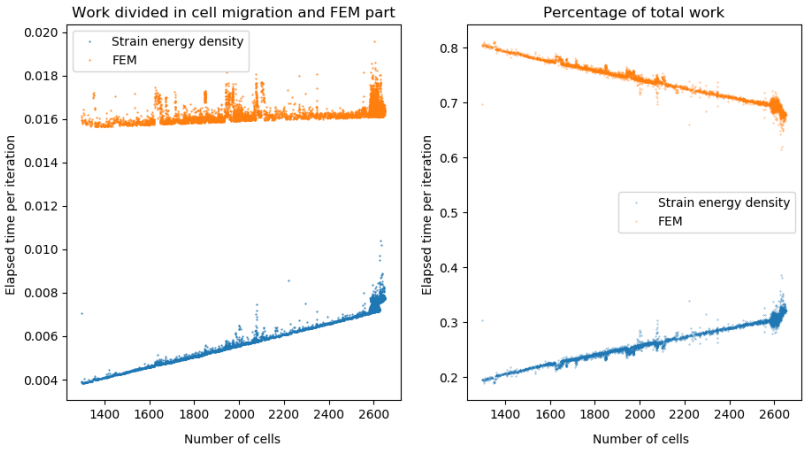


Figure 7.4: Elapsed time per iteration for a domain of 200 x 160 μm, with a time step of 0.09 s and run for 10,000 iterations, where the strain energy densities and the concentration gradient mapping are computed on the GPU.

To investigate the behavior of the computational work in relation with the number of cells, we consider different domain sizes. For all simulations 10,000 iterations with a time step $dt = 0.09$ sec are run and the initial lifespan of the cells is determined by a uniform random distribution between $[0, 600]$ sec. In the Table 7.1 some runs are displayed to indicate the differences between using the CPU only and using the GPU on the strain energy densities computations or the concentration mappings or both. Apart from the total simulation time and the average time per iteration, the range of the number of cells that occur during the iteration are given as well.

Domain dimensions	GPU usage	Total time	Average time	Range of cell counts
100 x 80 μm	None	2.68 min	0.0160 s	300 - 650
100 x 80 μm	Only cell part	1.58 min	0.0095 s	300 - 650
100 x 80 μm	Cell and FEM	1.14 min	0.0068 s	300 - 650
200 x 160 μm	None	31.73 min	0.1904 s	1,300 - 2,700
200 x 160 μm	Only cell part	12.4 min	0.0744 s	1,300 - 2,700
200 x 160 μm	Cell and FEM	3.84 min	0.0231 s	1,300 - 2,700
400 x 320 μm	Only cell part	142.3 min	0.854 s	5,000 - 11,000
400 x 320 μm	Cell and FEM	19.09 min	0.1146 s	5,000 - 11,000
600 x 480 μm	Cell and FEM	63.31 min	0.3799 s	12,000 - 23,000
800 x 640 μm	Cell and FEM	153.28 min	0.9197 s	20,000 - 39,000

Table 7.1: Overview of wound healing simulations for different domain sizes, all simulations with 10,000 iterations and a time step of 0.09s.

On the smaller domains, 100 x 80 μm and 200 x 160 μm, the FEM part occupies a much larger part of the computation time (more than 70%). On the somewhat larger domain, 400 x 320 μm, it can be seen that the cell energy density computations start to increase with significant jumps, such that percentage of the computation work of the FEM part decreases towards 55%. Finally, for the 600 x 480 μm domain, the cell energy density computations become more expensive than the FEM computations (with more than 20,000 cells). For the modelling scenarios investigated here (as was described in the literature report [20]), it is assumed that the initial domain is very dense. Furthermore, proliferation will only halt if the mechanical energy between cells becomes too large. If a lower density of cells is assumed, then the ratio between the work of the strain energy densities and the FEM will change such that the FEM computations are dominating again.

7.2. Description of mmobi

For the choice of program, the efficiency was a key factor for the model, but the user-friendliness was also kept in mind. The wound healing model is made as a C++ program with a Python API, that can be executed as a library. The library has been given the name mmobi, which is an acronym for Mathematical Modelling of Burn Injuries.

After reproducing the same basic model behavior in C++, the model was extended with a second type of cell, the macrophages, and two other concentration fields, PDGF and TGF-β. The modelling dynamics also changed by setting the parameters and implementing extra features as described in Chapter 5. In Figure 7.5 a schematic overview is given of the most important functionalities of the core of the wound healing program. The main function, runIteration, computes the displacement of all the cells every time step. Before the first iteration however, the work of constructing an initial setting of the domain is done. This involves the placement of cells and the determination of their state. Moreover, the Finite Element mesh is constructed and the initial concentrations and constant parts are computed.

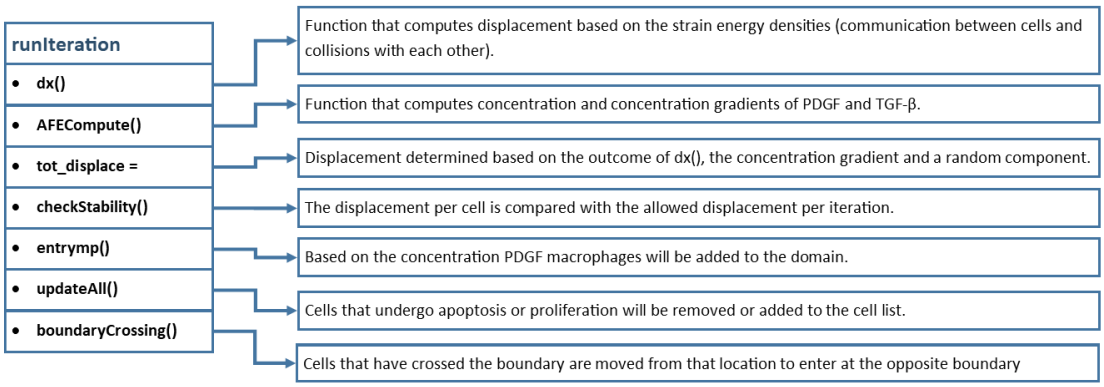


Figure 7.5: Schematic overview of the main components of the runIteration function.

Apart from this, there are many other small help functions that are used within the important functions mentioned here. Additionally, there are many functions that for example can be used to measure the elapsed time in a specific part of the code, to access members from other classes or to print information.

7.2.1. C++ particularities

Two parts that could not easily be implemented with standard C++ libraries only were the generation of the Finite Element Mesh and the solving of linear systems. The concentration field is found by solving a linear system that is built up during the Finite Element computations. The linear algebra library `Eigen` [15] is used in this part of the code. With this library, the matrices can be built up as sparse matrices and different direct and iterative solvers can be chosen from.

A triangular grid is used in combination with Finite Element methods, to compute the concentrations PDGF and TGF-β over time. The generation of this mesh is done by the Triangle mesh generator [33]. In the documentation of this software, it is explained that the mesh created with conforming Delaunay triangulation is suitable for Finite Element analysis. A Delaunay triangulation assures that within the circumcircle (circle that passes through all three vertices) of any triangle, no other vertex is located. Moreover, the angles of the triangles are made in such a way that it is never smaller than twenty-five degrees [33]. The advantage of using a mesh generator that makes an unstructured grid, is that the errors that occur have different directions.

Different options for the mesh generation can be chosen. In the simulations we use the following switches:

- -q : Makes sure the mesh triangles have no angles smaller than 20 degrees. A number can be added after 'q' to have a different criterion on the angle.
- -a : To fill in the maximum triangle area, it is put to 10^{-11} m^2 as default.

- -e : To have the mesh generator give a list with edges back.
- -z : To set items counting from zero (instead of one).

For most of the MATLAB simulations during the literature study [20], a mesh with 4448 triangles was used for a domain of size 100 x 80 μm.

7.2.2. Random numbers

There are two types of random number generators: PRNGs and TRNGs, which are pseudo and true random generators, respectively. The advantage of the PRNG, is that it is fast, however it is deterministic and has a period (sufficiently long for most problems). The TRNG on the other hand is more expensive from a computational point of view, but is non-deterministic and does not have a period [16]. Often, we are dealing with pseudo-randomness numbers. This will lead to the same sequence of random numbers every time the program is executed. For debugging purposes, this can be a desired option. In order to randomize the generation of random numbers, we need to give a 'seed' as input. For example, we can read the CPU time at a certain moment and use the value as a seed.

The uniform distribution in C++ can be declared with the line: uniform_real_distribution<double> dist (0.0,1.0). Then, the function call operator () will return the next random number in the distribution. Without usages of a different seed, the output will be the same. A non-deterministic uniform random number generator in C++ is std::random_device. The random number can be used as a seed for the uniform distribution. The std::random_device can be used as a seed for the Mersenne Twister. This is the most widely used PRNG, it is used as default PRNG by MATLAB, Python and the CUDA library among others. In order for this to work, it is required that the CPU can offer the support for this.

On the GPU, the Curand library is used to generate random numbers. These random numbers are used to determine if a cell dies or divides. As it is supposedly faster to generate many numbers at once, the amount of random numbers generated each iteration is two times the number of cells. Not all random numbers are needed, because if the division criterion is satisfied, the death criterion is skipped. Similarly, both checks are redundant for the case that a cell has a life span shorter than 300 seconds.

Seeding the C++ random generators happens by std::random_device() while calling the program from Python. This is also the default setting for C++, but within the C++ code it can be easily changed by giving a specific seed to the scenario_builder. In this way, scenarios can be repeated when executing the CPU scenario (the Curand library used in the GPU program has a build-in seeding).

7.2.3. Python API

After constructing a model with all the mathematical functionalities, a user interface was made. This was done in Python in order to be able to easily construct the modelling scenario and visualize the results of the simulation. From within the Python and C++ code, it is not possible to access each other's parts directly. To make this possible a C program is put in place to convert function calls and queries to variables. This C-wrapper links the C++ program to the Python program, in order to have an information transfer between the C++ and Python program.

The Python interface starts with a part where a basic scenario can be constructed (time step, number of iterations, domain size). Per scenario parameter, a value can be chosen or the user can refrain from this. In the latter case, a default parameter is set. In C++ these parameters can similarly be chosen or left to default by usage of the Scenario_builder class. In both Python and C++, the user can choose to perform all computation on the CPU or to use the GPU for the parts that have a GPU implementation. The default option would be to use the GPU, as this is much faster. Moreover, the computations on the GPU are done in double precision, just as on the CPU.

Furthermore, some Python tests were implemented. The tests can be executed by running the command nosetests (where the -s switch is needed to show all output). Within this test environment, we can quickly check if the requirements are satisfied for the current settings. For example, the sizes of the arrays should correspond to the number of cells and the stability criterion should be satisfied.

7.3. GPU usage for speed up

For the largest bottlenecks in the CPU code, alternative implementations using the GPU were made. The CPUscenario and GPUscenario class are derived classes from the base class Scenario. This means that they share the same fundamental basis, with most of the member variables and general member functions. The specific functions for the CPU and GPU are then overwritten in their own class and additional members can be added.

7.3.1. Strain energy density and mechanical energy

For every cell the same computations are performed in order to find the displacement. This advocates for a parallelization of the computations. The part that is performed on the GPU involves the influences of the strain energy density and collisions with other cells. This part is relatively easy to parallelize, as the computation for a cell does only depend on information of the previous iteration. The cell locations and the lifespan of the cell are send to the GPU, and after the computation the displacement, the updated state and lifespans are send back to the CPU. The decisions of cells undergoing proliferation and apoptosis are also performed on the GPU. These decisions are then stored in the state vector. On the GPU, we cannot use the random generator from the CPU. Pseudo random numbers are therefore generated by the Curand library. These numbers are double precision numbers from the uniform distribution that can be generated on the GPU.

7.3.2. Finite Element computations

After using the GPU to do the cell strain energy density computations efficiently, the FEM computations are the limiting factor. This part can be divided into three parts: the building, the solving and the mapping parts. The building and solving parts have more or less constant computation time during the simulations, while the mapping computations increase with the number of cells.

In the end, two parts of the Finite Element computations were speeded up by using the GPU. This implementation was more difficult than for the cell migration described in section 7.3.1. In principle, the Eigen library should be able to work with CUDA implementations. However, the latest version of CUDA is not compatible with the latest version of Eigen, which we both use in this project. The Eigen library is only used in the FEM class, however this class is part of the Scenario class. In order to use the GPU on certain parts, we need to make these parts as separate files and load those into the Scenario class. Moreover, in the usage of the Eigen library, special vectors and matrices are used. With the standard library vector in C++, it is easy to convert information to arrays, such that the GPU can work with it. For the Eigen members, we need to use a mapping function. Besides that, the matrices need to be converted to a one-dimensional array.

The largest part of the work during the mapping is performed in the trianglecheck function. This function performs the computations as described in Section 4.3.2. With a structured grid, it would be relatively easy to determine the division of cells over the elements in the grid. With the used unstructured grid however, we need to check per cell if it is located in a certain triangle. This is a time-consuming operation, as we loop through the whole cell list and check for every cell if it is located in the current triangle. In a sequential program, we could think of more efficient variations, such as leaving out cells that where located in a previously checked triangle. To speed up the computations using the GPU, the loop over all elements stays in place and the outer loop, over all the triangular elements, is parallelized.

Finally, the computation of the right-hand side vector of the system for the computation of the concentration TGF-β, as was given by Equation 4.7, was performed on the GPU as well. Again, the most time-consuming part here is the trianglecheck function. This time the function is used to check if a macrophage is located in a certain triangle. The influences of the macrophages are stored in the right-hand side vector, however, as multiple triangles can have a contribution to the same node, we need to be careful doing additions in parallel. The atomic functions make sure that only one thread can read and write at a certain location. In the case of AtomicAdd(), it is guaranteed to be able to read the value stored in the vector, add a value to it and store the new value back in memory, without interference of any other threads. Many of the additions to the element vector are zero, as there are no macrophages, and thus no secretion of TGF-β in those elements. Before initiating the AtomicAdd() function, it is therefore checked if the additions are non-zero.

7.3.3. Final optimizations

To reach an even more efficient code, some changes are made to achieve more speed-up. First of all are the computations involving the tPA concentration field (which were used in the literature study [20]) were made non-active. Secondly, all dynamic arrays from the Eigen library were replaced by fixed sizes ones, if possible. This sets the number of nodes per element to three and the number of nodes for a boundary element to two. Furthermore, the contents of elmat and elmatbnd are stored differently. It is made sure that all the inner loops in the FEM part iterate over the fast (first) index. Eigen is column major by default, therefore we prefer to loop through the first index, instead of the second.

Within the Eigen library, we need to choose our own matrix solver. There are some direct solvers to choose from, as well as iterative solvers. Furthermore, it is possible to use wrappers in order to have an external solver. For small systems, most direct solvers perform quite well. However, for larger systems (more than 1,000,000 unknowns) the solving time becomes a bottleneck. Iterative solvers are usually the better choice when the system reaches a certain large size, however, this is not an issue for this specific model yet. Although, the iterative solvers cope better with limited memory than the direct solvers, the latter is much faster for now. The Sparse supernodal LU solver is chosen for this project. After conversion of the model to 3D, the iterative solvers will probably become a better choice.

The mesh does not change during the simulation and quantities like the element area and the basis functions per element can therefore be computed once and used for the rest of the simulation. This already causes speed-up by removing unnecessary work. In order make this worthwhile on especially the GPU, the mesh information is located on the device once, instead of sending it to the device every kernel invocation. The speed-up of locating this information on the device is discussed in more detail in Section 11.1.1.

8

Wound healing quantification

In order to be able to assess the dynamics of the wound healing simulation, it is convenient to have some measure that gives information about this process. Furthermore, some quantity is needed to be able to compare different simulations to each other. In this chapter, different quantities for these purposes are introduced. In Section 8.3, the two-sample Kolmogorov-Smirnov test is described, which uses the wound quantifiers to compare different simulations.

8.1. Measuring wound healing

In [6] several methods are described to quantify the wound healing rate. Most wound healing quantities use the area of the wound over time, either the absolute value or relatively to the initial wound area.
The absolute wound size is just the wound area in m^2 over time. The relative wound is defined as $W = \frac{A_t - A_{t=0}}{A_{t=0}}$, where A_t indicates the area of the wound in m^2 at time t. The wound healing measurement of these quantities is very dependent on the size and form of the wound. In [6] the disadvantage of the incomparability of wounds with different sizes is overcome by including the wound perimeter in the wound healing rate as well. The wound margin advance is defined as

$$d_i = \frac{A_i - A_0}{\frac{1}{2}(p_0 + p_i)}, \text{ with } A_i \text{ the wound area and } p_i \text{ the wound perimeter at time } i.$$

The main point that remains is to choose a method to measure the wound area and perimeter. In practice the maximal diameter of the wound is measured (a_i), and after this the diameter perpendicular to this one is measured (b_i). Subsequently, the required quantities are approximated by an ellipse. Hence, the wound area and perimeter are

$$A_i = \frac{\pi}{4} a_i b_i, \tag{8.1}$$

$$p_i = \pi \left(\frac{3}{4}(a_i + b_i) - \frac{1}{2}\sqrt{a_i b_i} \right) \quad [6]. \tag{8.2}$$

A more convenient measure, regarding the parameters in the model, is to use the fibroblast density. A definition could be the restoration of normal fibroblast density: $R_f = \frac{\rho_{wound}}{\rho_{normal}}$, where ρ is the density of fibroblasts, thus the number of fibroblasts per m^2. The normal density ρ_{normal} has the value that was used to initialize the amount of fibroblasts in the uninjured area.

Another measure that is easy to keep track of is the relative wound density (RWD). This involves the total number of cells and their distribution over the wound and the undamaged part. This quantity can be used to measure cell migration [14]. It is defined as

$$\%\text{RWD}(t) = \frac{w_t - w_0}{c_t - w_0} \times 100, \tag{8.3}$$

where w_t and c_t are the cell densities in the wounded and undamaged area respectively at time t.

8.1.1. Polygonal estimation

As the grid used in the model is rectangular, a circular or elliptic approximation of the wound might not be very accurate. Especially in the initial phase of the migration into the wound, the cell gap will have a more rectangular shape. The shape is therefore approximated by a polygon, which allows us to cover the corners of the domain better than would be possible with an elliptic approach.

The domain is divided into twelve segments, where the sizes are based on a 30 degree angle. The middle of the domain, the (0,0) point, is taken as the origin. The first segment is located right below the origin. The twelve segments have their mean at the angles $-\frac{\pi}{2}, -\frac{\pi}{3}, -\frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}, \pi, -\frac{5\pi}{6}, -\frac{2\pi}{3}$. Within the 30 degrees segments, the distance from the cell that is closest to the origin is written down. In case there are no cells in the segment, the maximal distance from the origin to the wound boundary within that segment is taken.

Per segment, two quantities are computed in order to approximate the wound gap and the wound perimeter. The area in between two segments is computed as the triangle area for a triangle with a 30 degrees angle (C) and the two connecting sides (a, b) with length from the origin to the location of the closest cell.

$$\text{Area triangle } = \frac{1}{2} a\ b\ \sin(C), \text{ as follows from the SAS-rule for triangles.}$$

The wound gap is found by summing up the results of the twelve computations. The perimeter of the wound is found by adding up the third sides that make the triangles with a and b complete. It can be found by solving for c in the law of cosines:

$$c^2 = a^2 + b^2 - 2\ a\ b\ \cos(C).$$

In order to compare this method with the elliptic approximation, a simple approximation based on Equations 8.1 and 8.2 is done as well. With the minimal distances computed for the polygon estimation, we can find wound diameters in six directions. As 30 degree angles were taken, we can find three pairs of orthogonal wound diameters. With these pairs, three elliptic wound areas and perimeters are computed. The largest wound area and perimeter are taken as final approximation for the elliptic approach.

While doing these computations, all cells are located in their segments and thus after this procedure the number of fibroblasts in the wound and outside the wound are known as well. This allows us to compute the restoration factor of fibroblasts R_f and the relative wound density (RWD) as described in Section 8.1 with a very small amount of extra computations.

8.2. Assessment of wound quantifiers

The approximations for the area and perimeter overshoot the actual value a bit on purpose, such that it is the maximal value that can occur for the approximation method of the wound. With this choice the approximated area can only shrink when a fibroblast enters the wound. The overshoot for the polygonal approach is less drastic than it is for the elliptic approach, as can be seen in Table 8.1.

	Wound area	Wound perimeter
Exact value	0.8 mm^2	3.6 mm
Polygonal approximation	0.9279 mm^2	3.7418 mm
Elliptic approximation	1.1379 mm^2	3.7924 mm

Table 8.1: The actual wound area and perimeter at $t = 0$ in comparison with the estimated values for the polygonal and elliptic approach.

In order to check the applicability of the wound quantifiers described in the previous Section 8.1, a test scenario is run. This scenario simulated the initial four days of wound healing, thus existed of 57600 iterations with a time step of 6 seconds. The domain size was [2000 μm x 1600 μm] and the wound is located in the middle of the domain with dimensions [1000 μm x 800 μm]. Every 10 iterations, the wound healing quantifiers are computed and plotted.

As in the current model, contraction of the wound is not implemented, there cannot be said much about the actual closure of the wound. During these simulations, the wound boundary will have no displacement. The current implementation models the pathways of the fibroblasts. Therefore, the wound area and perimeter

are now tracking the entry of the fibroblasts in the wound and the part of the wound bed that they occupy.

Two wound quantities are based on the approximated wound area and perimeter. In Figure 8.1 these quantities are displayed for both the elliptic approximations and the polygonal approximations. In the left figure it is visible that the elliptic approach has larger deviations between different iterations, especially in the initial period. This is caused by the fact that the area is based on the diameter in two directions only, where for the polygonal approach it is based on six directions. If a cell moves from one segment to another, this makes a larger difference for the elliptic approximation, than it would for the polygonal approximation. Furthermore, the initial wound shape is rectangular, which is easier to approximate with a polygon than an ellipse.

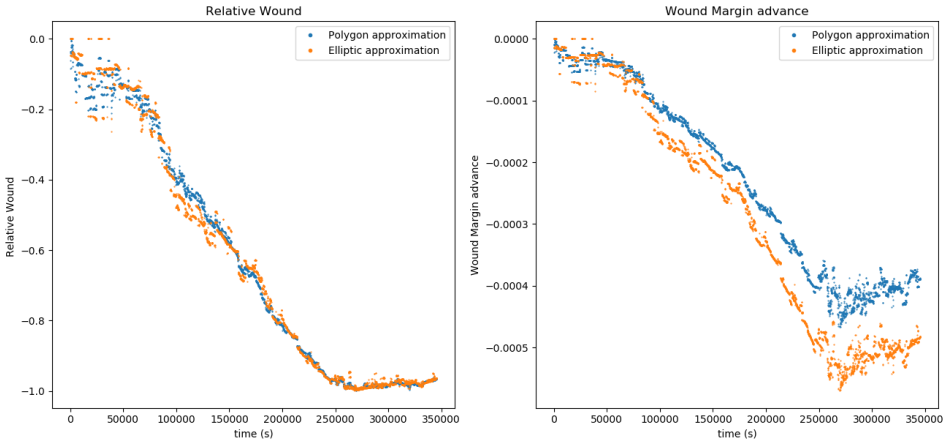


Figure 8.1: Polygonal and elliptic measurement of the relative wound area and the wound margin advance.

The right figure of 8.1, gives more diverging results for the two approaches. This is mainly caused by the fact that the wound margin advance is a ratio depending on the initial wound area and perimeter approximations. For the case that the wound is completely closed, thus the wound area and perimeter are zero, then the wound margin advance are 0.496 mm and 0.600 mm for the polygonal and elliptic approach, respectively. In the figure, we see that the lowest points of both approximations go towards these values. The behavior of the approaches are similar, however scaled differently. Again, the polygonal approach seems to have a somewhat smoother course than the elliptic approach.

After the wound area reached zero, the relative wound and wound margin advance are not very informative anymore. Therefore, the quantifiers that describe the fibroblast density over time are useful as well. The restoration of the fibroblast density and the relative wound density can be tracked over a longer period. Moreover, these quantifiers give information about the global rate of healing, in addition to the specific information about the wound lay-out. The slope of the RWD graph gives information about the migration between the undamaged area and the wound. Thus this gives a visualization of the incentives of the macrophages to migrate towards the wound bed. The restoration of fibroblasts gives an overview of how far the process towards a normal fibroblast density is.

Based on the findings described in this section, we decide that the polygonal approach of the relative wound and wound margin advance is a better choice than the elliptic approximation. This quantity is informative until the moment that the wound area reaches zero. Both restoration of fibroblasts and the relative wound density will be used as quantifiers to reflect the global state of the wound healing process.

8.3. Comparing simulations

The quantities that measure the wound healing will be used as the main components to compare between simulations. Different cases that can be investigated are the following:

1. Simulations in double precision versus single precision.

2. Simulations with varying time step.
3. Simulations with varying grid size for the finite element computations.
4. Checking the influence of a certain parameter.

By investigating the items 1, 2 and 3, the fastest method can be chosen while still having the same qualitative results as for a more accurate and computationally more expensive method. The fourth item can be investigated to gain more insight into the mathematical model. For example, to find out which parameter variations have a small or large impact on the outcome of the model.

Given two or more scenarios that will be investigated, a few time instants $t_1, \ldots, t_n$ will be chosen. Then for a large number of simulations, the wound healing quantifiers will be noted at t_i. With these results, a wound healing distribution can be made for every time instant. The distributions can be made for all the different scenarios. We would like to find out if two (or more) scenarios behave similarly. This can be done by performing a Kolmogorov-Smirnov test on the data samples from the scenarios. It allows us to test if two data sets belong to the same, continuous probability density function or to different ones. Moreover, this test is non-parametric, which means that no assumptions are needed regarding the distribution of the variables and the error between the actual density and the samples [43].

Given two sample distributions f_{n_1} and f_{n_2} and their underlying distributions f_1 and f_2, we can define

$$\text{the null hypothesis } H_0 := f_1(x) = f_2(x)$$
$$\text{and the alternate hypothesis } H_1 := f_1(x) \neq f_2(x).$$

First, the cumulative distributions, F_{n_1} and F_{n_2}, are made. Subsequently, we need to find the maximum vertical distance between these cumulative density functions. This is defined by the quantity D, where

$$D = \max_x |F_{n_1}(x) - F_{n_2}(x)|.$$

The test statistic D is compared with a critical value, based on the chosen confidence level and the number of samples, in order to either reject or accept the null hypothesis [43]. The critical values for different confidence levels α are given in Table 8.2, on the condition that both sample sizes are larger than 15. Based on the sample sizes, it is defined that $s(n) = \sqrt{(n_1 + n_2)/n_1 n_2}$. With $s(n)$ computed and a chosen confidence level, the value of D_{crit} can be computed from the Table 8.2. Finally, the null hypothesis H_0 is rejected if $D > D_{\text{crit}}$ and is accepted otherwise.

α	$D_{crit}/s(n)$
0.10	1.22
0.05	1.36
0.01	1.63
0.005	1.73
0.001	1.95

Table 8.2: Critical values and confidence levels for K-S Two-Sample Test [43].

9

Simulation results

In this chapter the results of our baseline scenario will be discussed. This scenario is small enough to perform multiple runs within reasonable time and large enough to have a nice wound healing response. It consists of a domain with dimensions [-1mm, 1 mm] × [-0.8mm, 0.8 mm] and uses a time step of six seconds. For the mesh generation the maximum area constraint for the triangles is set to 10^{-11} m^2. For solving the systems of the concentration fields, the Sparse supernodal LU solver is used. The initial fibroblast density was set to 10^9 fibroblasts per m^2 and their initial lifespan was set by a uniform distribution between 0 and 600 seconds. The density was used to place the initial fibroblasts in the undamaged area. This density corresponds to an initialization of 2400 fibroblasts in the domain. In the first section, the modelling results will be described. In the second section, the computational work will be discussed. In the third section, a comparison is made between the CPU and GPU program in C++.

9.1. Wound healing dynamics

In Figure 9.1 several snapshots at consecutive times of the simulation are given. Initially there is only PDGF in the wound bed, which diffuses over time through the domain until the concentration diminishes to a small amount that is not visible on the fixed scale anymore. The concentration TGF-β increases during the period of time where macrophages enter the wound and secrete TGF-β.

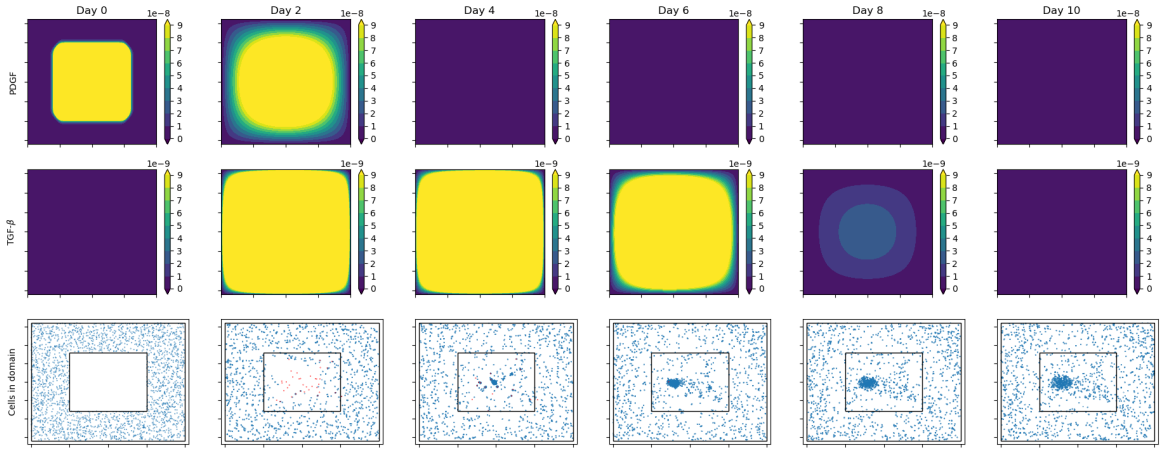


Figure 9.1: Overview of concentration fields PDGF and TGF-β and the cells in the domain. The blue cells are fibroblasts and the red cells are macrophages.

The macrophages (the red cells) are diffusing toward the center of the domain and by secreting TGF-β, they guide the fibroblasts (the blue cells) towards the center as well. This is visible in the snapshot of the second day. At certain locations the TGF-β concentration is a bit higher than in the surrounding tissue, although this

is not visible for the concentration scale in Figure 9.1. This is where the fibroblasts eventually group, as is the case around the sixth day. Hereafter, the fibroblasts will diffuse over the domain and some will undergo apoptosis at overcrowded areas.

The ingress into the wound site of macrophages starts quickly as can be seen in the third graph of Figure 9.2 and stagnates after a day. At that point the concentration PDGF has decreased below the threshold that allows macrophages to enter the wound. The macrophages keep on migrating towards the center of the wound, in the direction of the gradient of PDGF. The apoptosis rate of the macrophages increases with the decreasing concentration of PDGF and with no proliferation or entry of new cells, the population will finally deplete (as is the case at day six). Meanwhile, the fibroblast population keeps proliferating as there is enough space left to fill. The limit of available nutrients is not a problem in the domain yet, thus by the proliferation of fibroblasts the population increases steadily, as can be seen in the second graph of Figure 9.2.

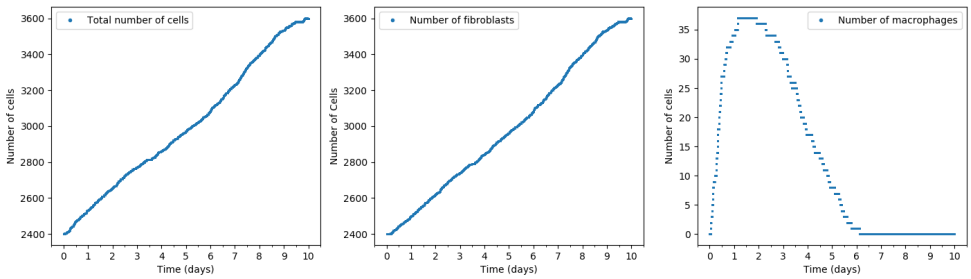


Figure 9.2: Growth of the total cell population, fibroblasts and macrophages over time.

The time evolution of the populations is not a very informative indicator for the wound healing. The healing status is easier to derive via the created wound quantifiers. First of all, we have the absolute wound area over time in Figure 9.3. The overall behavior of the wound area is decreasing over time, however, some increases in wound area occur due to the approximation we perform, as was described in Section 8.1.1. At the point where the wound is almost closed, the gained wound area is smaller as the gap becomes smaller as well. After four days, the fibroblasts are spread out over the complete injured area. However, the fibroblasts group at locations where the TGF-β concentration is highest, which might leave some small areas in the wound without any fibroblasts. This leads to small increases in wound area as is seen around the fourth, fifth and sixth day. After the first 4 days, this quantity becomes approximately zero and does not give us useful information about the wound healing anymore.

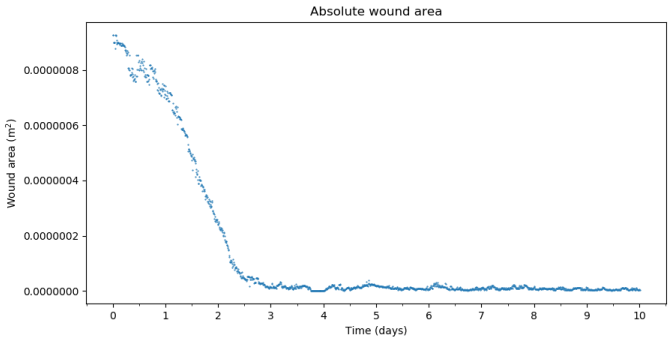


Figure 9.3: The absolute wound area over time.

In Figure 9.4, there are four more wound quantifiers displayed. The relative wound density in Figure 9.4a gives the same information as the absolute wound area. However, as it is scaled it is easier to compare to wounds that do not have the same size. The wound margin advance (Figure 9.4b) indicates a steady shrinking rate of the wound during the first days. After 2 days, the wound margin advance becomes smaller and more fluctuating. The wound margin advance depends on the approximated area as well as on the approximated wound perimeter and therefore, is very dependent on the specific locations of the fibroblasts.

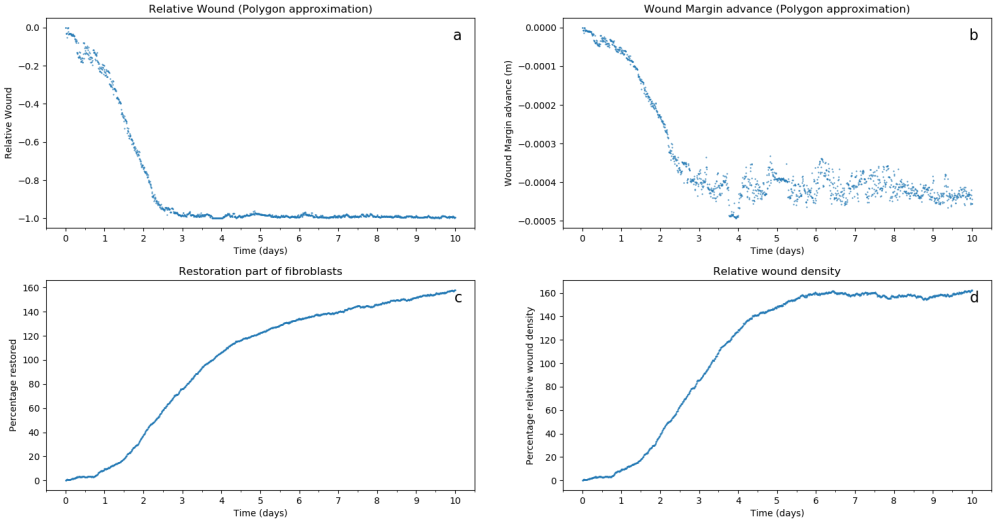


Figure 9.4: Overview of quantifiers of the wound healing process.

Initially, the area decreases as many fibroblast from the surrounding undamaged tissue migrate into the wound. Subsequently, the fibroblasts will be driven by the gradient of TGF-β once being secreted by the macrophages. The relative wound density, as was defined in Equation 8.3, demonstrates the effect of the gradient of TGF-β on the migration rate of the fibroblasts (Figure 9.4d). The steep increase in the graph corresponds with the high concentration of TGF-β during the period between the second and fourth day. Thereafter, the relative wound density does still increase as proliferation of fibroblasts occurs without the restriction of needing to have enough space.

The restoration of the fibroblasts (Figure 9.4c) indicates the progression towards a 'normal' amount of fibroblasts. The normal amount is defined as the amount that corresponds to density that was set for the initialization of the fibroblasts. After 10 days, the density has reached a level that is larger than the assumed amount of fibroblasts for normal scenarios. After the large initial increase, the growth stagnates. This is caused by the fact that the TGF-β concentration has decreased, such that the proliferation is halted in case of overcrowding. Furthermore, the fibroblasts that originate from surrounding tissue have for some part already migrated and therewith the density is lower in the undamaged region, as can be seen from Figure 9.5. At the seventh day, the density over the whole domain reaches the 'normal' density of 10^9 fibroblasts per m^2 and subsequently overshoots this value.

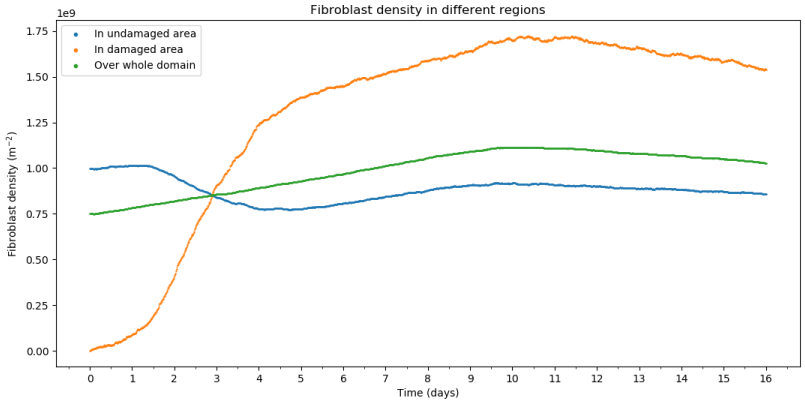


Figure 9.5: Overview of the fibroblast density over a time of 16 days in the wound, the undamaged area and total domain.

After the wound healing response, the domain should go back towards a normal state eventually. On a longer simulation of 16 days, it is visible that the decrease of fibroblasts starts around the ninth day (Figure 9.6). With a high concentration of TGF-β, the apoptosis is suppressed and proliferation is allowed regardless of the available space. At some point, apoptosis is not suppressed anymore, although cells can still undergo proliferation. This happens when the growth curve becomes flat and population does not either grow or shrink much. When the concentration TGF-β decreases even more, proliferation is halted due to limiting space. The migration of fibroblasts from the wound area back to the undamaged area will not occur very fast. The speed of movement of fibroblasts depends on the receptors bound to TGF-β molecules according to Equations (4.10). Hence, the speed keeps on decreasing on the longer run.

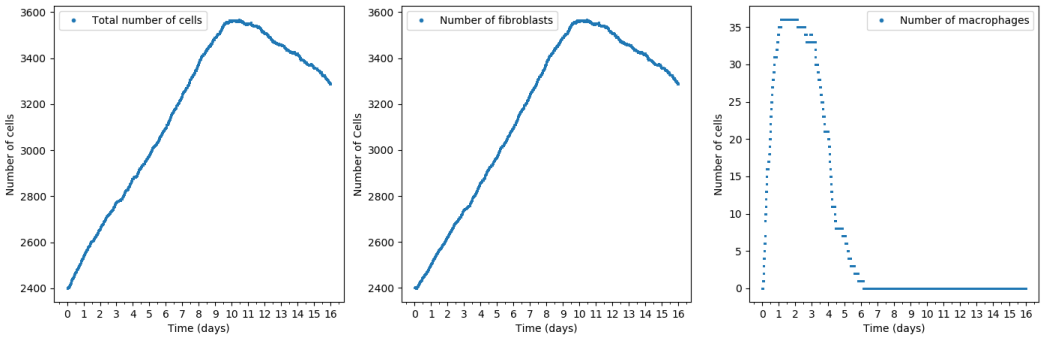


Figure 9.6: Growth of the total cell population, fibroblasts and macrophages over time.

9.2. Computational work load

Before going into detail about the computational work, the used hardware is stated for reproducibility. The Intel® Core™ i5-7400 CPU is used, with 8 GB of memory. The used GPU is the GeForce GTX 1080, which has a total global memory of 8114 MB. The gcc 5.4.0 compiler is used, where the optimization flags '-O3' and '-march=skylake' are enabled for the simulations in this chapter.

During the simulation of the baseline scenario, different parts of the program were timed. The different timed parts are displayed in Figure 9.7. The strain energy densities are computed on the GPU and occupy a small part of the total iteration time. This part of the computations last between 0.007 and 0.011 seconds per iteration, which is less than 1/20 of the time that the Finite Element computations occupy. There are three jumps where the computational works suddenly increases. This can possibly be caused by needing extra memory space that is slower to access. If more actions are performed in an iteration, such as computation of the wound quantifiers, the iteration time increases somewhat as well.

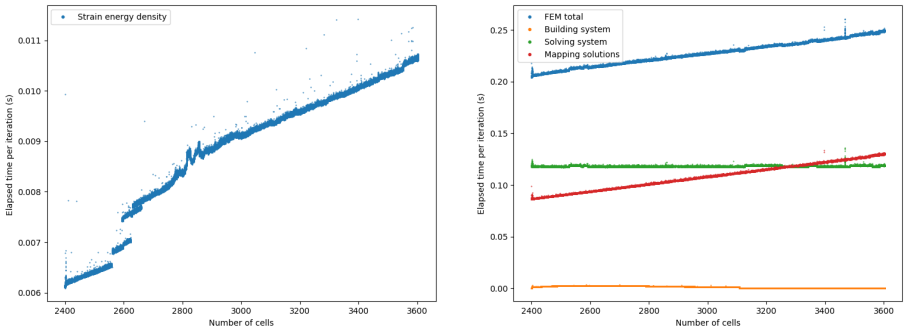


Figure 9.7: Division of work load in different part of the wound healing simulation.

The Finite Element computations are divided into three parts: the building of the system, the solver part and the mapping of the concentrations to the cells. Both the mapping and building part use the GPU to speed up

computations. The solver part is the slowest part in this set-up, although this will presumably change for a larger number of cells or for a less fine mesh. The solver part only depends on the number of nodes in the chosen mesh. Therefore, the computational work is constant for a non-varying mesh during the simulation.

The linear increase of the work load with the number of cells for most parts is also visible in Figure 9.8. The dependency of the computational work load is clearly related to the number of cells, which follows from the similarity of the shapes of the two right figures with the left figure displaying the cell growth. Only in the initial period the computation times are higher. This is caused by doing a lot of extra work at the start and storing the outcomes, such that it speeds up the rest of the iterations.

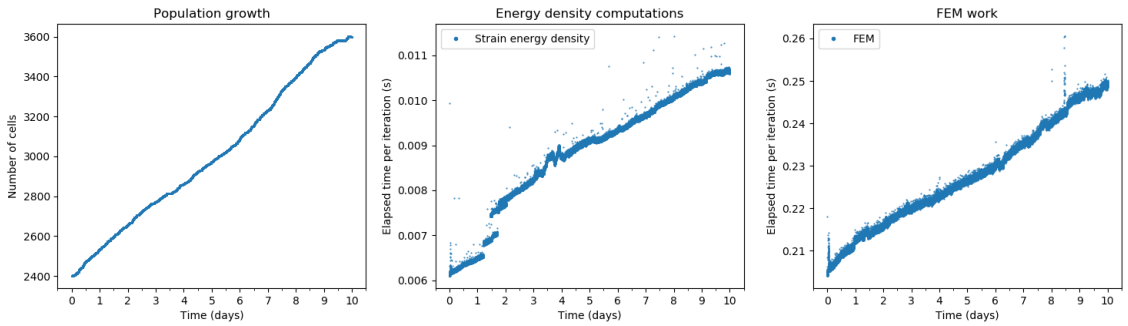


Figure 9.8: Cell growth (left) and computational work in two parts over time.

Finally, the scalability of the model is checked. An increase in the domain size, leads to an increase in the required computational power, as there are more cells and more nodes in the Finite Element mesh. For all scenarios, 2000 iterations are run with a time step of 6 seconds. The mesh area constraint was set to $5 \cdot 10^{-11}$ m^2 and a direct solver is used for the FEM system. In Table 9.1 the increase in work is given in relation to the smallest domain. For the three smallest domains, the computation times are quite reasonable. Furthermore, the factor of increase in computation time is smaller than the increase in domain area.

The turning point is the 2.5 x 2 mm domain, where the computation factor is larger than the domain increase. For scenarios of this size or larger, the mapping part becomes the largest bottleneck. This is caused by the increase of triangles on the one hand and on the increase of cells on the other hand. We parallelize over the triangles, however, the work per triangle (as was described in Section 4.3.2) increases with the number of cells.

Domain size	Initial cells	Vertices	Triangles	Computation time	Factor area	Factor time
0.5 x 0.4 mm	150	3199	6239	0.11 min	1 x	1 x
1.0 x 0.8 mm	600	12,544	24764	0.31 min	4 x	2.8 x
2.0 x 1.6 mm	2400	50,069	99473	1.58 min	16 x	15 x
2.5 x 2.0 mm	3750	78,043	155224	5.07 min	25 x	27 x
3.0 x 2.4 mm	5400	112,383	223785	3.01 min	36 x	50 x
4.0 x 3.2 mm	9600	199,370	397423	14.57 min	64 x	132 x
5.0 x 4.0 mm	15000	311,379	621029	33.03 min	100x	300 x

Table 9.1: Computation timings for different domain sizes for a simulation of 2000 iterations with a time step of six seconds and a maximal triangle area of $5 \cdot 10^{-11}$ m^2.

In order to make a ten-day simulation, we need 144,000 iterations for a time step of six seconds. Doing this for the domain with size 2 x 1.6 mm takes already two hours. For the domain with size 4 x 3.2 mm this increases to 18 hours. With the goal of Monte Carlo simulations in mind, the maximal feasible domain size is around 4 x 3.2 mm with the current resources.

9.3. Comparison of CPU and GPU program

We consider some simulations consisting of 5,000 iterations with a time step of 6 seconds. The maximal mesh area is set to 10^{-10} m^2, which leads to the construction of a mesh with 25,148 vertices and 49,821 triangles.

The length of this run is chosen to be quite short, as the CPU scenario already takes more than an hour to complete. The long run-time is the reason that in this analysis only a few CPU runs are performed.

We apply the methods as described in Section 7.1.1 for data obtained from the CPU and GPU implementations of the model. For both the data sets, we can make an estimate of the mean at every time point. For this mean, we can then estimate the Monte Carlo Error. This error indicates the variability of the mean. For increasing sample size, the error will also decrease. The means and the corresponding confidence intervals are plotted in Figure 9.9, along with the data of both simulations. The 95% confidence interval for the CPU scenario (displayed as the black lines), is quite wide, as it is only based on 12 CPU runs. The confidence interval for the GPU scenario (displayed as the blue lines) is much smaller as it depends on 50 runs. The confidence intervals overlap at every time point. Therefore, the hypothesis that the two implementations simulate similar behavior is not rejected.

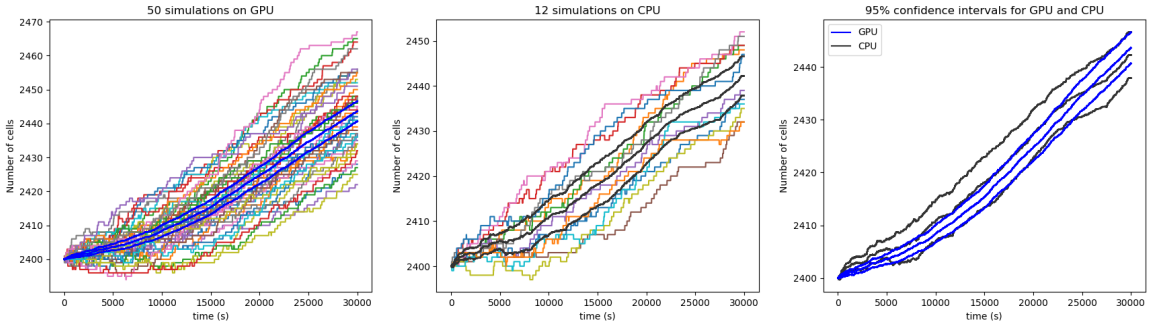


Figure 9.9: Different runs of the GPU and CPU program with 5,000 iterations and time step of 6 seconds. There are 50 GPU runs and 12 CPU runs. The 95% confidence interval for the mean of the set of GPU runs is given in blue and for the set of CPU runs in black.

The difference in run-time between the CPU and GPU implementation is very apparent. For 10,000 iterations with a time step of six seconds and a maximal mesh area of $5 \cdot 10^{-11}$ m^2, the CPU program lasts about 7.15 hours, while the GPU program needs less than 8 minutes. In Figure 9.10 the computational work for a CPU simulation of one day in different parts is displayed. Towards the end of the simulation, the time spent in computing the strain energy densities (SED) is about 0.13 seconds and the time spent in performing the Finite Element computations is about 2.54 seconds. For a simulation of one day on the GPU with the same Finite Element mesh, the elapsed times per iteration are 0.0065 seconds and 0.039 seconds respectively. This means that the SED is computed 20 times faster and the FEM part is computed about 65 times faster in the GPU implementation. The over-all gained efficiency of the GPU is a bit more than 58 times faster, as the FEM computation accounts for the largest portion of the overall work. Whereas in the CPU implementation the solving of the concentration systems is relatively fast, this changes to be one of the bottlenecks as the other parts use the GPU for speed-up.

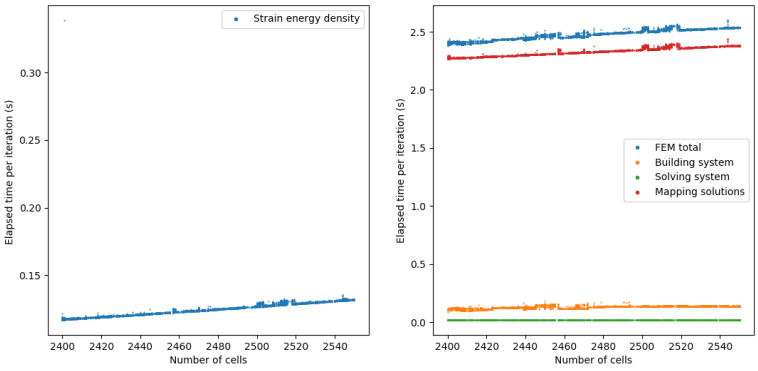


Figure 9.10: Computational work of the strain energy density computations and different parts of the Finite Element computations in the CPU program.

10

Accuracy of concentration fields

In this chapter the concentration fields are studied in more detail. In the first section, we have attempted to estimate the accuracy of the PDGF field by Richardson's Extrapolation. However, different methods did not lead to feasible results. In the second section, an alternate approximation of the TGF-β field by means of Green's function solutions is studied.

10.1. Richardson's Extrapolation for estimating the error order

In order to be able to say something about the order of the errors being made during the computation of the concentration fields, Richardson's Extrapolation is used. Suppose the correct value is M and the approximation we make is $Q(h)$, where h indicates the grid size. Then the error is $E(h) = M - Q(h)$. For a small enough grid size, this error can be approximated as $E(h) = c_p h^p$, where p is the order of the error and c_p a non-zero constant. The order of the error can be estimated by computing three approximations of the solution for grid sizes h, $\frac{h}{2}$ and $\frac{h}{4}$. With the formula

$$\frac{Q(h/2) - Q(h)}{Q(h/4) - Q(h/2)} = 2^p, \tag{10.1}$$

we can find the estimated order p [39].
In order to compare the approximations of the concentration field, the L^2-norm is computed for three different meshes. The L^2-norm is a function norm and is defined as

$$\|c\|_{L^2(\Omega)} = \left[\int_\Omega c^2 d\Omega\right]^{1/2}.$$

The function c^2 is approximated per triangle element and added up afterwards as follows:

$$\left[\int_\Omega c^2 d\Omega\right]^{1/2} = \left[\sum_{k=1}^{N_{\text{tri}}} \int_{e_k} c^2 d\Omega\right]^{1/2} \approx \left[\sum_{k=1}^{N_{\text{tri}}} \left(\frac{|\Delta e_k|}{6} \sum_{p=1}^{3} c(x_p)^2\right)\right]^{1/2}. \tag{10.2}$$

For the Richardson Error estimation, we want to halve the grid size. This is a straightforward measure to do in one- or two-dimensional rectangular grids. For a triangular unstructured mesh it is not clear how this mesh should be refined. We could think of halving the length of the sides of the triangle, although another option would be to halve the triangle area. The latter is implemented in the triangle software, where an input parameter is the maximal allowed area. Within the triangle software, we can refine an input mesh. This means that the nodes of the previous mesh will be present in the refined mesh and extra nodes will be added in order to comply with the maximal area constraint.

10.1.1. Precise refinement factor

Previously, it was assumed that by halving the area constraint for a triangular element, a refinement factor of two can be used. In [23] an alternate definition for the refinement ratio is used. Here $r = \frac{h_{g-1}}{h_g}$, where h_g is finer than h_{g-1} and $h = \sqrt{\frac{A}{N}}$, the square root of the ratio between the total area and the number of elements.

In order to be safe with regard to the refinement factor, we consider the case where the two refinement ratios are not equal. Three grids are considered with respectively h_1, h_2, h_3, where $h_1 > h_2 > h_3$. For sufficiently small h, the error $M - Q(h)$ is approximated by $c_p h^p$, with $c_p \neq 0, p \in \mathbb{N}$. We can write for the three grids:

$$M - Q(h_1) = c_p (h_1)^p, \tag{10.3a}$$
$$M - Q(h_2) = c_p (h_2)^p, \tag{10.3b}$$
$$M - Q(h_3) = c_p (h_3)^p. \tag{10.3c}$$

And after subtracting (10.3b) from (10.3a) and (10.3c) from (10.3b), we obtain the equations

$$Q(h_2) - Q(h_1) = c_p((h_1)^p - (h_2)^p) = c_p (h_2)^p (r_{12}^p - 1), \tag{10.4a}$$
$$Q(h_3) - Q(h_2) = c_p((h_2)^p - (h_3)^p) = c_p (h_3)^p (r_{23}^p - 1). \tag{10.4b}$$

Here, the ratios are defined as $r_{12} = \frac{h_1}{h_2}$ and $r_{23} = \frac{h_2}{h_3}$. Division of Equation (10.4a) by (10.4b) gives the equation

$$\frac{Q(h_2) - Q(h_1)}{Q(h_3) - Q(h_2)} = \left(\frac{h_2}{h^3}\right)^p \frac{(r_{12}^p - 1)}{(r_{23}^p - 1)} = (r_{23})^p \frac{(r_{12}^p - 1)}{(r_{23}^p - 1)},$$

that needs to be solved for the order p.

This equation can be solved by the iterative Newton-Raphson method. We define

$$f(p) = (r_{23}^p - 1)\frac{Q(h_2) - Q(h_1)}{Q(h_3) - Q(h_2)} - (r_{23} r_{12})^p + r_{23}$$

and want to find the zero of this equation by iteratively solving $p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$. Alternatively, other root-finding algorithms, such as Halley's method can be used.

10.1.2. Limitations

In practice, the estimation of the order with Richardson's Extrapolation does not always give reasonable results. There can be several causes for this; higher-order derivatives do not need to exist, mistakes can be made in the code or the final result is a combination of multiple approximations that remove the direct relation with the estimated order p [39]. Furthermore, the mesh refinement needs to be systematic, which means that two conditions, uniform and consistent refinement, need to be complied with [31]. Uniform refinement means that the rate of refinement should be the same over the whole domain, and consistent refinement means that the mesh quality must improve or stay constant with refinement. Mesh quality involves things like cell skewness and aspect ratio.

With the mesh generator, an unstructured grid is generated. It is not possible to enforce some criteria such that the grid surely improves on all points. Even with the precise refinement factor of the grid, the approximation of the error does not give reasonable results. Therefore, a simple structured triangular grid (as displayed in Figures 10.1 and 10.2) is used to find the estimated order of the error.

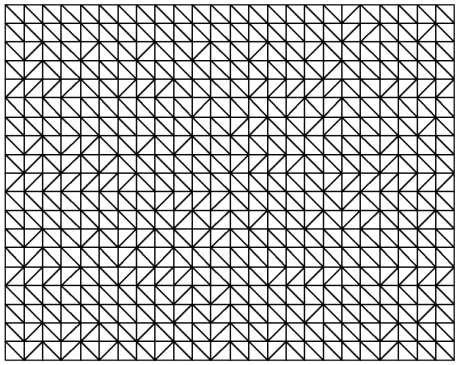


Figure 10.1: Base grid, with grid size h.

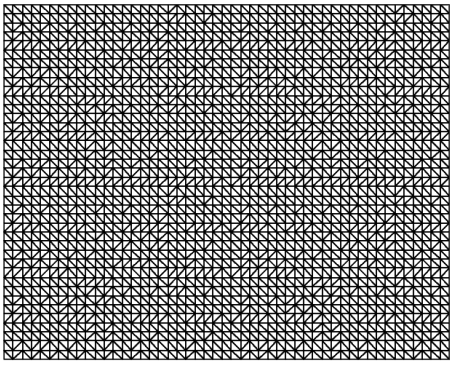


Figure 10.2: After one refinement, with grid size $\frac{h}{2}$.

During every iteration, some approximation errors are made. This error depends on both the grid size h and the time step Δt. In the process of making these quantities smaller and smaller, the error will converge to zero as well. For sufficiently small h and Δt, higher order dependencies do not have much influence on the total error. For the backward Euler method, the total error would then depend quadratically on the grid size and linearly on the time step. When halving the grid size, the time step needs to be divided by four to prevent the time integration errors to dominate the total error.

The procedure to find the estimated order starts with choosing the coarsest grid and the corresponding time step. The grid size h and time step Δt will be chosen such that $\frac{D\Delta t}{h^2} \leq \frac{1}{2}$ is satisfied. The diffusion coefficient of PDGF, D_{cp}, is $3.33 \cdot 10^{-12}$ m^2/s and with an initial grid size of $2 \cdot 10^{-5}$ m, the time step should be smaller than 60 seconds. As the initial setting of the PDGF field has a quite sharp transition from high to zero concentration, we need to run the simulation long enough in order to let the concentration diffuse through the medium. The simulations will have a total length of one hour to assure this. In Table 10.1 the estimated L^2-norms are given for three different grid sizes.

h	Δt	m	$Q(h)$
$32 \cdot 10^{-7}$	0.9375	3840	1.04096e-09
$16 \cdot 10^{-7}$	0.234375	15360	3.89459e-09
$8 \cdot 10^{-7}$	0.05859375	61440	6.57464e-09

Table 10.1: One hour simulations for estimating the order of the error by halving the grid size h.

With the formula (10.1), we get $\frac{Q(h/2)-Q(h)}{Q(h/4)-Q(h/2)} = 1.064767$. That would result in an order $p = 0.091$ which does not seem to be a correct outcome. However, the most precise computation is already approaching the limit of memory. Therefore, halving the grid size again is not possible. Alternatively, we attempt to find a solution between $h = 16 \cdot 10^{-7}$ and $h = 8 \cdot 10^{-7}$. The refinement factor for the grid size then becomes $\sqrt{2}$ and for the time step it becomes two. We reuse the results from Table 10.1 and add an extra simulation. We need to solve $\frac{Q(h_2)-Q(h_1)}{Q(h_3)-Q(h_2)} = (\sqrt{2})^p$, for the results displayed in Table 10.2. This gives us the estimated order $p = 2.3$.

h	# triangles	Δt	m	$Q(h)$
$16 \cdot 10^{-7}$	624000	0.234375	15360	3.89459e-09
$\sqrt{2} \cdot 8 \cdot 10^{-7}$	1248562	0.1171875	30720	5.74212e-09
$8 \cdot 10^{-7}$	2498000	0.05859375	61440	6.57464e-09

Table 10.2: One hour simulations for estimating the order of the error by reducing the grid size h with factor $\sqrt{2}$.

In the procedure of making a structured mesh, the number of triangles is determined by rounding up the value of the domain length or width divided by the grid size h. This means that the displayed grid size in Table 10.2 is not the actual used grid size and furthermore, the grid size can differ in the x and y direction. We investigate if applying the method as described in Section 10.1.1 gives different results for the order. The total domain area is $8 \cdot 10^{-7}$ m^2, thus the following grid sizes are obtained:

$$h_1 = \sqrt{\frac{8 \cdot 10^{-7}}{624000}}, \quad h_2 = \sqrt{\frac{8 \cdot 10^{-7}}{1248562}} \text{ and } h_3 = \sqrt{\frac{8 \cdot 10^{-7}}{2498000}}.$$

The refinements factors become $r_{12} = 1.41453$ and $r_{23} = 1.41446$, which are both fairly close to $\sqrt{2} \approx 1.4142$. Solving with the Newton-Raphson method, gives an order of $p = 2.2983$, whereas taking the refinement factor constant gave an order of $p = 2.3001$. This difference is not significant, hence as long as the refinements are approximately the same, using a constant refinement factor suffices.

Repeating the Richardson Estimation with the L^2-norm gives different and unrealistic results again. Specific information about the estimations are stated in Appendix B. Next to the L^2-norm approximation, we have attempted to look at specific locations in the middle of the domain and perform Richardson's Error estimation on this. This did not lead to convincing results either and therefore, we can draw no conclusions about the accuracy. What the problem is for approximating the error is not clear and needs to be studied further (see Chapter 13).

10.2. Approximate solution with Green's functions

The reaction-transport equations have the general form $\frac{\partial c}{\partial t} + D\nabla^2 c = Q(t,x)$, where D is the diffusion coefficient and $Q(t,x)$ a source term. In the case of the PDGF concentration field, the source is zero, whereas for the TGF-β concentration field there are multiple sources depending on the number of macrophages.

We use the derivations for the heat equation for the diffusion of the concentrations from [17]. The solution for the concentration can be expressed in terms of Green's function $G(\mathbf{x}, t; \mathbf{x_0}, s)$ as

$$c(\mathbf{x}, t) = \int_0^t \iint G(\mathbf{x}, t; \mathbf{x_0}, s) Q(\mathbf{x_0}, s)\, d^2x_0\, ds + \iint G(\mathbf{x}, t; \mathbf{x_0}, 0) c(\mathbf{x_0}, \mathbf{0})\, d^2x_0 \tag{10.5}$$
$$+ D\int_0^t \oint [G(\mathbf{x}, t; \mathbf{x_0}, s)\nabla_{\mathbf{x_0}} c - c(\mathbf{x_0}, \mathbf{s})\nabla_{\mathbf{x_0}} G(\mathbf{x}, t; \mathbf{x_0}, s)] \cdot \mathbf{n}\, dS_0\, ds,$$

where Q represents a source and the last line takes the boundary conditions into account.

First, we consider an infinite space, with no boundaries and no sources. Setting these quantities to zero, simplifies the above Equation (10.5) to $c(\mathbf{x}, t) = \iint G(\mathbf{x}, t; \mathbf{x_0}, 0) c(\mathbf{x_0}, \mathbf{0})\, d^2x_0$. For this case the Green's function can be derived by analyzing it with the Fourier transform, which gives us a general solution for dimension $n = 1, 2, 3$:

$$G(\mathbf{x}, t; \mathbf{x_0}, t_0) = \left[\frac{1}{4\pi D(t-s)}\right]^{\frac{n}{2}} e^{-|\mathbf{x}-\mathbf{x_0}|^2/4D(t-t_0)} \quad [17].$$

Insertion of this Green's function in Equation 10.5, allows us to find a solution for a problem with sources. This function is valid on an infinite space, however, the domain is bounded in the wound healing model. If the time difference $t - t_0$ is very small in relation to the domain length divided by the diffusion parameter, the solution can be approximated by the infinite space solution. This solution is valid in regions far away from the boundary, if the condition $\frac{L^2}{D(t-t_0)} \gg 1$ is satisfied.

The used domain length is usually around a millimeter and the diffusion speed is of the order 10^{-12}. The condition in this case becomes $(t - s) \ll 10^6$, which should be all right for a few time steps of 6 seconds. The approximations (for locations far from the boundary) that will be used are therefore:

$$c(\mathbf{x}, t) = \int_0^t \iint \frac{1}{4\pi D(t-s)} e^{-|\mathbf{x}-\mathbf{x_0}|^2/4D(t-s)} Q(\mathbf{x_0}, s)\, d^2x_0\, ds + \iint \frac{1}{4\pi Dt} e^{-|\mathbf{x}-\mathbf{x_0}|^2/4Dt} c(\mathbf{x_0}, \mathbf{0})\, d^2x_0.$$

For symmetric domains (which is the case for our model), the method of images can be used to construct a solution on a finite domain that satisfies the boundary conditions [22]. The main idea of this method is to create non-existent point sources outside the domain in order to comply with the boundary conditions.

We are interested in the approximation for the TGF-β field. The initial concentration is zero, thus the second integral vanishes. The sources are caused by the macrophages and have the form:

$$Q(\mathbf{x}, s) = \sum_{m=1}^{M} Q_m(\mathbf{x}, s) = \sum_{m=1}^{M} \kappa_{c_\beta} \delta(\mathbf{x} - \mathbf{x_m}),$$

where M is the number of macrophages and Q_m is the concentration that results from the secretion of one macrophage. By summing up the secretion of all individual macrophages, we obtain an approximation for the concentration field over the domain. The integration over area disappears with the Dirac Delta function, as it is only non-zero at the location of the macrophage, thus only the integration of time needs to be taken care of. The concentration field is then described by

$$c(\mathbf{x}, t) = \int_0^t \iint \frac{1}{4\pi D(t-s)} e^{-|\mathbf{x}-\mathbf{x_0}|^2/4D(t-s)} Q(\mathbf{x_0}, s)\, d^2x_0\, ds, \tag{10.6}$$
$$= \sum_{m=1}^{M} \int_0^t \frac{\kappa_{c_\beta}}{4\pi D(t-s)} e^{-|\mathbf{x}-\mathbf{x_m}|^2/4D(t-s)}\, ds, \tag{10.7}$$

where x_m is the location of a macrophage.

10.2.1. Computation with numerical approximation

In contrast to the computation of the TGF-β field by the Finite Element Method, we can now compute the concentration directly on a desired location $\mathbf{x}$.
Define the integrand as

$$f_m^c(s) = \frac{\kappa_{c_\beta}}{4\pi D(t-s)} e^{-|\mathbf{x}-\mathbf{x_m}|^2/4D(t-s)} \tag{10.8}$$

where the current time t , the location $\mathbf{x}$ and the macrophage location $\mathbf{x_m}$ have known constant values. Then, the concentration at time t and location $\mathbf{x}$ will be given by

$$I = \sum_{m=1}^{M} \int_0^t f(s)ds = \sum_{m=1}^{M} \left(\int_0^{dt} f(s)\, ds + \int_{dt}^{2dt} f(s)\, ds + \ldots + \int_{t-dt}^{t} f(s)\, ds \right).$$

After the second equality sign the integral is split in sub-integrals with the length of the chosen time step dt. For the approximation of the integral of $f_m(s)$ the trapezoidal rule [39] will be applied, this rule is defined by

$$\int_{t_L}^{t_R} f(t)dt \approx \frac{t_R - t_L}{2} \big(f(x_L) - f(x_R) \big).$$

Applying this rule for every sub-integral leads to

$$I \approx \sum_{m=1}^{M} \frac{dt}{2} \sum_{k=1}^{T} \big(f_m(t_{k-1}) + f_m(t_k) \big),$$

on a time interval $[t_{\text{start}}, t_{\text{end}}]$ with $t_{\text{start}} = t_0 < t_1 < \cdots < t_T = t_{end}$.

For the chemotaxis of the fibroblasts, the gradient of TGF-β is required. Equation 10.6 can be differentiated to x or y to obtain the gradients in the x and y directions respectively. The gradient for the concentration becomes

$$\nabla c(\mathbf{x}, t) = -\sum_{m=1}^{M} \int_0^t \frac{\kappa_{c_\beta}}{4\pi D(t-s)} e^{-|\mathbf{x}-\mathbf{x_m}|^2/4D(t-s)} \frac{2}{4D(t-s)} \begin{bmatrix} x - x_m \\ y - y_m \end{bmatrix} ds.$$

This integral will be approximated numerically, as was the case for the computation of the concentration. Now, the integrands that need to be approximated are

$$f_m^x(s) = \frac{-\kappa_{c_\beta}}{8\pi D^2(t-s)^2} e^{-|\mathbf{x}-\mathbf{x_m}|^2/4D(t-s)} (x - x_m), \tag{10.9}$$

and

$$f_m^y(s) = \frac{-\kappa_{c_\beta}}{8\pi D^2(t-s)^2} e^{-|\mathbf{x}-\mathbf{x_m}|^2/4D(t-s)} (y - y_m). \tag{10.10}$$

During the simulations, we need to keep track of the locations of the macrophages and the times these macrophages were at those locations. This information needs to be stored in the memory and is used to compute the concentration or gradient at a certain time and place for all the subsequent iterations.

Two vectors are used to memorize the locations, one for the x-coordinates and one for the y-coordinates. The third vector saves the time stamp of a macrophage being at that location. Every iteration, the locations of the moved macrophages and the newly entered macrophages are appended to the vectors.
In order to find the concentration or gradient at time t, we need to loop through all locations and integrate f_m as described in Equation 10.8, 10.9 or 10.10 from the time stamp saved in the vector to the current time t.

10.2.2. Results

The approximation of the concentration and gradient for TGF-β with Green's functions seems to give similar results as for the previous method of solving the Finite Element system. That is, the cell dynamics based on the concentration gradient seem to work similarly. For both methods, different assumptions are made, thus the approximations can differ as a result of this.

A straightforward implementation, as described in the previous section, is not very feasible in terms of computation time. In Figure 10.3 only the first 500 iterations are run (with a time step of 6 seconds). Here the computation time is already becoming large. At first, the computation time is zero as there are no macrophages.

In this simulation, two macrophages entered the domain, which happened at iterations 68 and 366. At every time iteration, the number of integrals that need to be approximated increases with the number of macrophages present in the domain. The quadratic fit seems to be a good indication of the growth of work. Therefore, a very large time per iteration can be expected on longer time scales.

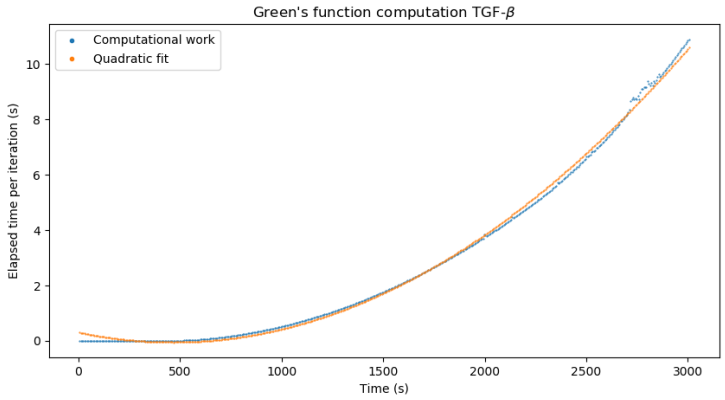


Figure 10.3: The elapsed time per iteration spent for the computation of the concentration gradient of TGF-β by using Green's function approximation.

It seems that this method is not very efficient, especially for simulations consisting of many small time steps. However, in order to make a fair comparison with the Finite Element approximation, the GPU is deployed for the computations of this part as well. The GPU program computes the gradient of TGF-β in a given cell center by parallelizing over the cell locations list. Although this speeds up the program, per cell location the amount of work that is left takes a long time. This increases as long as there are macrophages present in the domain.

In Figure 10.4 the work performed during the approximation with Green's Functions by using the GPU is displayed. In this specific simulation, macrophages were created at iterations 99 and 486. We cannot compare this simulation exactly with the one displayed in Figure 10.3, as the macrophages were created at different moments. The more macrophages and the earlier they enter, the more work will need to be performed within this frame of 500 iterations. In both simulations, we deal with two macrophages entering during the simulation. However, they enter at later moments in the GPU simulation.

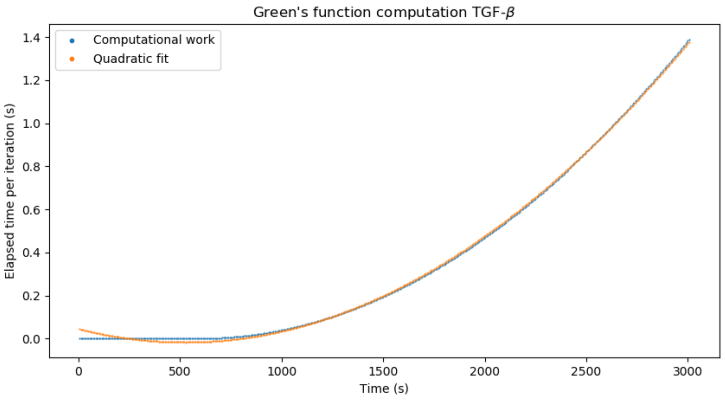


Figure 10.4: The elapsed time per iteration spent for the computation of the concentration gradient of TGF-β by using Green's function approximation with usage of the GPU.

From the two figures, we can conclude that the GPU improves the computation time of the Green's function approximation. The downside is that we are still dealing with rapidly increasing computational work with increasing cell counts. For the simulation in Figure 10.4 the behavior is again increasing quadratically. The

simulation results discussed in Chapter 9 use the Finite Element method to find the concentrations. Solving the systems for both PDGF and TGF-β and mapping the solutions in order to find the gradient takes less than 0.25 seconds. For this approach the elapsed time per iteration already reaches 1.4 seconds per iteration. Therefore, we conclude that the Green's function method is not performing well with respect to computational work for the current set-up of the model.

11

Analysis model behavior

This chapter consists of the results of analyzing some specific parts of the model. In the first section, the communication times that are necessary for the GPU usage are measured. Thereafter, the impact of changing certain settings in the model is studied. The Kolmogorov–Smirnov test is performed for the time step, the precision and the choice of a normalization parameter in the model.

11.1. Timing GPU communication

The usage of the GPU speeds up our basic CPU code as is described in Section 7.1.2. However, with three separate GPU kernels every iteration, there is a lot of communication necessary between the CPU and the GPU. In this section, the part that is copying data back and forth to the GPU and the part that the GPU is actually working are timed. These timings are done by using `std::chrono::high_resolution_clock` and `cudaDeviceSynchronize()`, which are described in Section 6.4. This allows us to time the communication part and GPU work, although the simulations are slower, as the progress is stalled until the GPU computations are completely done.

The communication with the GPU can be divided into two parts: the latency and the bandwidth. The latency involves everything that needs to happen before the actual data transfer takes place. The bandwidth is the time involved with copying something to the device. We can describe this by the formula for communication time $T = \alpha + \beta n$, where α is the latency, β the bandwidth and n the size of the data transfer [38].

Analyzing the latency and bandwidth provides insight into what speed-up would still be possible, if we could omit the data transfer multiple times per iteration, or even simulate the whole scenario on the GPU. Ideally, the GPU is working a more significant part of the time, than is used by transferring data between the CPU and GPU.

11.1.1. Reducing data transfer

We divide the timings into three different parts: the displacement computations based on the strain energy densities, the computation of the mapping and the computation of the right-hand side vector for the system of TGF-β.
For cell energy density we need some previously generated data. In the function, we need the cell list, the lifespans, a list with random numbers, the concentration PDGF in the cell centers and the parameter list. The displacement vector, the states of the cells and the updated lifespan are send back. For the mapping, we need a lot of information from the CPU. As this function maps the concentrations to the cells, we need the values of the concentrations and their gradients, the coordinates of the cells and the mesh specifics. After computation on the GPU, the values with the concentrations and gradients in the cell centers are sent back. For the generation of the element vector, we mainly need to find the locations of the macrophages in the mesh. Therefore, the list with macrophages is sent, together with the mesh specifics. The resulting right-hand side vector f_b is sent back.

Putting the parameter list, with values as in Section 5.2 and scenario specific input data, on the GPU is relatively easy, as these are all constants. To send over the specifics of the mesh, such as the coordinates of the nodes, we are dealing with arrays. This involves more work, as we need to allocate the arrays on the device and save the pointers to these locations. However, when the transfer times were investigated for the program where only the parameter list is allocated on the device, it became clear that the memory transfer of the mesh information takes up a considerable part, as is shown in Table 11.1.

For all the timed simulations in this section, the area constraint for the Finite Element Mesh is set to 10^{-11} and a time step of six seconds is used. In the Table 11.1 it is shown that for the Generate element vector part, almost the complete computation time is spent copying data to and from the device.

Part	Transfer	GPU computation	Percentage transfer
Strain energy densities	0.0002998 s	0.003837 s	7.25 %
Mapping kernel	0.02129 s	0.04704 s	31.16 %
Generate element vector	0.02107 s	0.0004415 s	97.95 %

Table 11.1: Percentages of transfer times from the total simulation with 2,000 iterations (given values are average of two runs).

After implementing the class that stores all constant mesh information and sending this to the GPU only once, the transfer times decrease considerably. For the same number of iterations and domain, the transfer time is about 20 times faster for the mapping kernel and about 29 times faster for the generate element vector part. The work conducted on the GPU increases slightly, as the access of the arrays is done via the object that stores the locations of the arrays. The total time for the mapping kernel is only 1.3 times faster, as the transfer part was not the major work. On the contrary, the transfer for the generate element vector part covered almost the total time and therefore improved to a 20 times faster total time. The specific times can be seen in the next subsection, for the 2,000 iterations scenario on a 1.0 mm x 0.8 mm domain.

11.1.2. Transfer time in relation to simulation size

Both the transfer time and the computation time on the GPU will increase with the increasing number of cells. This makes sense, as with more cells, more data needs to be copied and more computations need to be performed. On the other hand, the latency will have less impact in comparison to the bandwidth and the computation on the GPU.

In the tables below, three simulations are compared with each other for each of the GPU parts separately. In Table 11.2 the timings for computing the strain energy densities and in Table 11.3 the timings for mapping the concentrations are given. For these two kernels, the average transfer time and computation time are found by dividing the sum of the timings during the whole simulation by the number of iterations. For the generation of the element vector of the TGF-β system (Table 11.4), the vector is only non-zero if there are macrophages present. In the iterations where no macrophages are present, no computations will be performed in this part. Therefore, only the iterations where the GPU is actually used are taken into account.

Domain size	iterations	Transfer	GPU computation	Ratio
1 mm x 0.8 mm	2,000	0.0002746 s	0.0043407 s	1 : 15.8
1 mm x 0.8 mm	10,000	0.000272 s	0.004460 s	1 : 16.4
1.2 mm x 0.96 mm	2,000	0.0002761 s	0.005173 s	1 : 18.7

Table 11.2: Timings of the data transfer in relation to the GPU computation time for the GPU part that computes cell migration based on the strain energy densities.

Domain size	iterations	Transfer	GPU computation	Ratio
1 mm x 0.8 mm	2,000	0.0009412 s	0.05318 s	1 : 56.5
1 mm x 0.8 mm	10,000	0.0009345 s	0.05477 s	1 : 58.6
1.2 mm x 0.96 mm	2,000	0.001174 s	0.09030 s	1 : 76.9

Table 11.3: Timings of the data transfer in relation to the GPU computation time for the GPU part that maps the concentration from the nodes to the cell centers, in order to obtain the concentration and its gradient in these locations.

In Table 11.2 it can be seen that the relative amount of work on the GPU increases with longer iterations and larger domains. The ratio improvement is not very large, as both the bandwidth and GPU time will increase, however the increase in GPU time is larger. For the mapping kernel (Table 11.3) the same effect is seen while prolonging the iteration time. For the 1.2mm x 0.96 mm domain the ratio improves quite a lot. This behavior is explained by the fact that the number of triangles in the mesh increases and that a check of which cells are located in it is performed for every triangle. The GPU computations increase for a larger mesh, while there is no extra data transfer involved, because the mesh specifics were transferred once at the start of the simulation.

Domain size	iterations	Transfer	GPU computation	Ratio
1 mm x 0.8 mm	2,000	0.00066024 s	0.0003916 s	1 : 0.59
1 mm x 0.8 mm	10,000	0.0006581 s	0.001100 s	1 : 1.67
1.2 mm x 0.96 mm	2,000	0.0009070 s	0.0004186 s	1 : 0.46

Table 11.4: Timings of the data transfer in relation to the GPU computation time for the GPU part that computes the right-hand side vector of the TFG-β system.

The generation of the element vector involves a rather small amount of work. Small changes in the scenario have therefore a relatively large effect. Especially when the amount of macrophages is small, there is not much that needs to be computed. Therefore, the ratio is not very good for the short simulations (2,000 iterations) where the number of macrophages is generally below ten. The timings of the three scenarios are displayed in Table 11.4. With the mesh specifics being stored on the device once, the main copying involves a zero vector f_b to the device and the updated vector f_b back. This vector has the length of the number of nodes in the FEM mesh and causes the increase of the transfer time in the larger domain.

11.2. Varying the time step

The time step is of large influence on the performance of the computation time. One iteration takes less than half a second, however, in order to simulate a day, 28,800 iterations are needed with a time step of three seconds. When choosing the time step small enough, the collisions between cells are modelled explicitly. Cells that come into contact will exert repelling forces and move away from each other. In this case, cells cannot move through each other in an iteration.
The criterion that was imposed during the literature study [20] was:

$$||dX_i(t)|| = \|v_i\|\Delta t \leq \frac{R}{2} \quad \forall i \qquad \Rightarrow \qquad \Delta t \leq \frac{R}{2\max\|v_i\|}. \tag{11.1}$$

This criterion will guarantee, that the displacement is never larger than half the radius. However, it might be possible to have a less strict criterion. This will allow cells to have a larger displacement and even move partly through each other (which corresponds to moving over each other in biology), as long as this does not cause any problems. After computation of the displacement, new repelling forces will be computed to find the displacement in the subsequent iteration. The only check that needs to be implemented is one that checks if no unrealistic displacement (multiple cell radius lengths for example) takes place.

Since we are not specifically interested in the small scale behavior of the model, it might be possible to use a larger time step. The important point is that this results in similar dynamics of the wound healing. We perform an analysis on two time steps: a scenario with a time step of three seconds and one with six seconds. With a time step of three seconds, we generally comply with the criterion (11.1). For a time step of six seconds, this is not the case and therefore we allow the displacement to be twice as large as in Equation 11.1.

Since our model contains several stochastic processes, the impact of the time step needs a statistical assessment. In Figures 11.1 and 11.2, the distributions of the simulations with time steps of three seconds (in orange) and six seconds (in blue) are displayed for the absolute wound area and the density of fibroblasts in the wound. In these figures, the histograms en the kernel density estimates are plotted. The y-axis is scaled to obtain an area of one, thus does not give important information. In the distributions, we see some differences, although these do not seem to be very large or to have structural behavior. The largest difference between the distributions occurs at day two for the fibroblast density (first graph of Figure 11.2). Further in the simulations, at days four and six, there is much more agreement between the distributions.

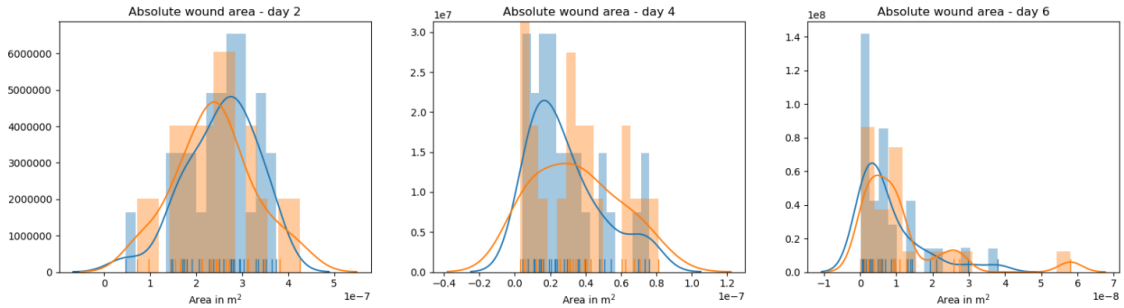


Figure 11.1: Distributions of the absolute wound area measured at day 2, 4 and 6. It consists of 28 simulations with time step of 6 seconds (in blue) and 21 simulations with a time step of 3 seconds (in orange).

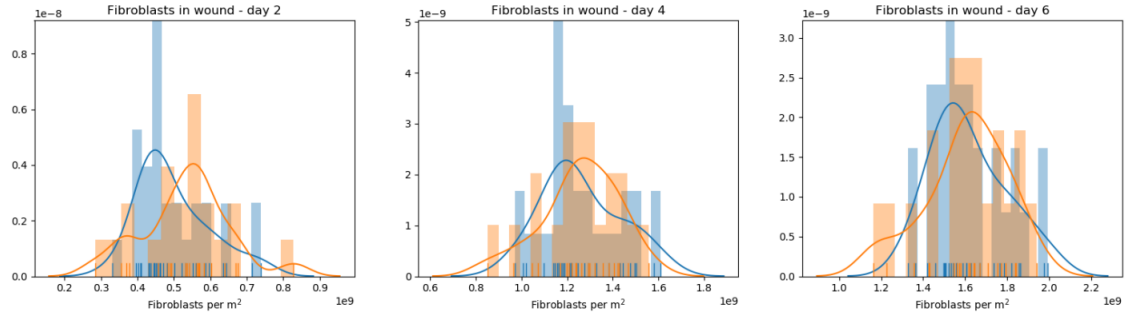


Figure 11.2: Distributions of the fibroblast density measured at day 2, 4 and 6. It consists of 28 simulations with time step of 6 seconds (in blue) and 21 simulations with a time step of 3 seconds (in orange).

A more precise comparison is made by using the Kolmogorov–Smirnov Two-Sample test. In Table 11.5 the results are displayed for some of the wound quantifiers. The K-S statistic D gives the maximal vertical distance between the two cumulative distributions, as was defined in Section 8.3. For two sample distributions f_{n_1} and f_{n_2} and their underlying distributions f_1 and f_2, the null hypothesis is that $f_1(x) = f_2(x)$. We want to reject the null hypothesis if the statistic D is large. From the p-value, we can derive the level of confidence for which we will reject the null hypothesis.
We choose a confidence interval of 95%. For this level of confidence, the null hypothesis is not rejected for any of the wound quantifiers. The largest value of D occurs at day two for the quantifier that measures the fibroblasts in the wound. Here the null hypothesis would have been rejected for a confidence level of 90%, although we do not for a level of 95% or higher. Based on these results, we do not reject the hypothesis that the underlying probability densities are different. It can further be investigated how the behavior is for even larger time steps.

Wound quantifier	Time	K-S statistic	p-value
Wound margin advance	Day 2	0.250	0.387
Wound margin advance	Day 4	0.131	0.978
Wound margin advance	Day 6	0.262	0.331
Relative wound density	Day 2	0.333	0.109
Relative wound density	Day 4	0.155	0.913
Relative wound density	Day 6	0.167	0.861
Fibroblasts in wound	Day 2	0.369	0.0568
Fibroblasts in wound	Day 4	0.2028	0.658
Fibroblasts in wound	Day 6	0.190	0.730
Absolute wound area	Day 2	0.262	0.331
Absolute wound area	Day 4	0.262	0.331
Absolute wound area	Day 6	0.238	0.449

Table 11.5: Results of the Kolmogorov–Smirnov Two-Sample test for scenarios with a time step three (21 samples) and six seconds (28 samples).

11.3. Single versus double precision simulation

In the code of the model, all variable types that are real numbers are defined as 'real_t'. This allows us to easily set this type to either 'float' or 'double' and run the simulations in single or double precision respectively. The expectation is that the model runs faster in single precision. Most parts of the wound healing model are indeed computed faster or equally fast in single precision compared to how this was in double precision. Moreover, less memory is required for storing all numbers in single precision.

The exception is the part where the FEM systems are solved. The division of work is given in Figure 11.3 for a simulation in single precision. For a comparable simulation in double precision, the timings for the building and mapping parts are comparable. The solving part is below 0.02 seconds in double precision. Hence, in single precision this part performs about four times slower. The strain energy density computations on the other hand are faster in single precision, although the difference is not very large. The double precision implementation takes about 0.008 seconds to compute the strain energy densities.

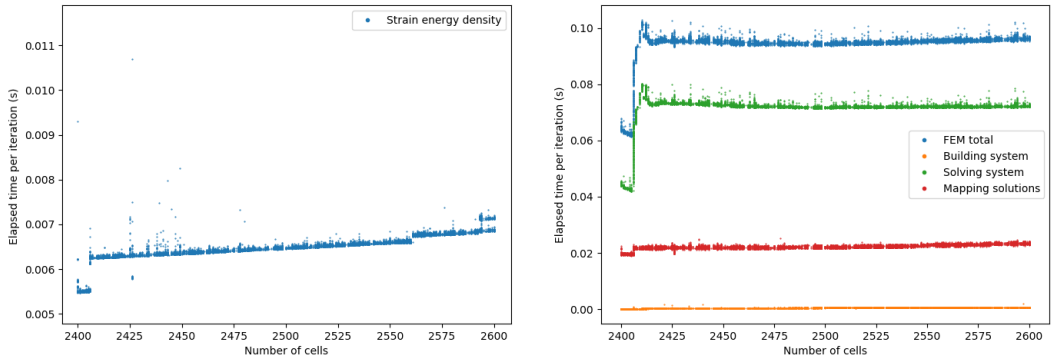


Figure 11.3: Division of work for a single precision simulation with 20,000 iterations, a time step of 6 seconds and a domain of size 2mm x 1.6mm

While trying out other iterative solvers, BiCGstab and CGleastsquare, we observe wrong results for the concentration field. Apparently, the entries of the matrix are so small that the iterative solvers give a vector with zeros as a valid result. Multiplying both the matrix A and the right-hand side vector b with a factor 10^9 seems to give more reasonable results for solving the system $Ac = b$.

The direct solver might take longer in single precision, as rounding errors make the system more difficult to solve. For the Bi Conjugate Gradient stabilized solver, where the system is multiplied by 10^9 first, there is a speed-up (about 1.5 times faster) of the solving time when performing this in single precision. We can guess a concentration that should be close to the solution or use the zero vector as start solution. Starting with the concentration from the previous iteration results in a faster solving time, as we need fewer iterations to find a solution.

Probably, some problems are occurring, which were not occurring for the double precision implementation. The exact problem has not been identified yet. Therefore, no further comparison is made between the wound quantifiers of both models. In single precision, small values are rounded to zero faster than in double precision. This might be a possible cause for the different behavior in single precision. As the parameters are expressed in standard units, many parameters have very small values. Converting the parameters to different units that lead to larger values might help (see Chapter 13).

11.4. Parameter variation of gradient scaling

For the computation of the displacement caused by chemotaxis, as in Equation 4.8, special care had to be taken to prevent division by zero. It was chosen that the norm was approximated as $\|\nabla c\|_2 \approx \sqrt{2}\max\left(\left|\frac{\partial c}{\partial x}\right|, \left|\frac{\partial c}{\partial y}\right|\right)$ are rounded to zero due to precision. If the gradients are zero, then the displacement was set to zero as well. We will call this method the base approach.

Another option would be adding a small value ϵ to the denominator to prevent division by zero [21]. For now, we will call this option the ϵ-approach. The displacement caused by chemotaxis then becomes

$$dx_{\text{conc}} = v_{\text{cell}} \left(\frac{\nabla c}{\|\nabla c\|_2 + \epsilon} \right) dt \ , \quad \text{with } \epsilon \text{ a small constant .}$$

The value of ϵ should not be chosen too small, as then the displacement can become excessively large. In the case that either $\left(\frac{\partial c}{\partial x}\right)$ or $\left(\frac{\partial c}{\partial x}\right)$ or both are non-zero and $\|\nabla c\|_2$ becomes zero, we still want to guarantee that $\left|\frac{\nabla c}{\epsilon}\right| \leq \mathbf{1}$. The minimal value of the gradient to satisfy this is $\sqrt{1.17549 \cdot 10^{-38}}$. This value is based on single precision, as the smallest positive number in C++ is `std::numeric_limits<float>::min()` = $1.17549 \cdot 10^{-38}$. We set $\epsilon = 10^{-18}$ to comply with criterion $\left|\frac{\nabla c}{\epsilon}\right| \leq \mathbf{1}$. The advantage of this definition of the displacement is that we use it for all cells regardless of the value of the concentration gradient. On the other hand, we add ϵ to the denominator every computation, in contrast to only adjusting the displacement definition for problematic cases.

The Kolmogorov–Smirnov Two-Sample test is used to compare the two cases described in this section. Again, we have the null hypothesis that the underlying distributions of the two cases are the same and the alternate hypothesis that they are different. The results are displayed in Table 11.6. The sample size for the base approach was 35 and the sample size for the ϵ-approach was 25. For a confidence level of 95% we do not reject the hypothesis that the two approaches simulate results according to the same underlying densities. The sensitivity of the parameter ϵ is apparently not very large. Hence, we can choose one of the two methods without changing the outcomes significantly.

Wound quantifier	Time	K-S statistic	p-value
Wound margin advance	Day 2	0.171	0.744
Wound margin advance	Day 4	0.154	0.848
Wound margin advance	Day 6	0.206	0.518
Relative wound density	Day 2	0.12	0.977
Relative wound density	Day 4	0.160	0.815
Relative wound density	Day 6	0.269	0.206
Fibroblasts in wound	Day 2	0.12	0.977
Fibroblasts in wound	Day 4	0.160	0.815
Fibroblasts in wound	Day 6	0.269	0.206
Absolute wound area	Day 2	0.171	0.744
Absolute wound area	Day 4	0.194	0.592
Absolute wound area	Day 6	0.206	0.518

Table 11.6: Results of the Kolmogorov–Smirnov Two-Sample test for two different approaches of computing the displacement caused by chemotaxis. The sample size of the base approach is 35 and the sample size of the ϵ-approach is 25.

12

Conclusion

During this thesis project, a cell-based model is made to simulate the healing of burn injuries. In order to cope with the large work load that is involved with this, the GPU is used to attain an efficient program. Previous research (described in Chapter 3) indicated that simplifying assumptions can be done. The most important assumption is that the cell interactions can be modelled statistically, instead of taking the full cell history into account. Furthermore, specific representations of the cells and implementations of the some processes are not important for the qualitative behavior of the model on larger scales. The chosen cell-based approach in this research models the cells as discrete entities that can be tracked during the simulation. The concentrations in the domain, PDGF and TGF-β, are continuous entities.

In this project, we have combined two models to form a basis for the modelling of the wound healing. This model includes the influence of the strain energy densities and chemotaxis. There are two types of cells: fibroblasts and macrophages. Their displacement is composed of a component based on the strain energy density, a component based on the chemical decomposition of the domain and a random component. Moreover, a mechanical energy occurs by collision of cells as well. Proliferation and apoptosis are modelled by an exponential distribution. The concentration PDGF is present in the wound bed initially and TGF-β is secreted by macrophages. In every iteration, the concentrations are computed by the Finite Element Method and the gradients are found by mapping the concentrations to the cell centers.

A straightforward implementation of the model in MATLAB has shown a rapidly increasing work load for the computation of the strain energy densities for an increasing number of cells. Converting the MATLAB code to C++ has already given a considerable speed-up. However, the work load for the computation of the strain energy density still increased more than linearly with the number of cells. For more efficiency, the GPU has been employed on the largest bottlenecks. By performing the strain energy density computation and the mapping on the GPU for a domain with size 200 μm by 160 μm, there was a speed-up of a factor 2.35. On a four times larger domain, the speed-up already increases to a factor 8.3 times faster.

For the interpretations of the simulated results, we can run multiple simulations and subsequently compute confidence intervals based on the sample mean and standard deviation. To gain more insight into the wound healing process, we have introduced several wound healing quantifiers. With these quantifiers, we can have an overview of the wound area or amount of fibroblasts in different regions over time. The wound perimeter and area are approximated by a polygon, which turned out to be more suitable than the elliptic approach. The Kolmogorov-Smirnov two-sample tests are used to compare different scenarios. It is a non-parametric method to test if two scenarios have the same underlying density or a different one.

The model developed in this project simulates the fibroblast entry in the wound. This happens by incentives of the TGF-β concentration field that the macrophages secrete. The macrophages enter the domain when the concentration PDGF is high enough and will survive in the domain as long as this concentration is available sufficiently. During the simulation, the cell population grows steadily to fill up the damaged area. Where normally the cell growth would halt due to the restriction of space, it continues in the presence of TGF-β. After the depletion of the macrophages, the concentration TGF-β diminishes and fibroblasts undergo apoptosis in

the overcrowded areas.

The final implementation with the GPU is more than 58 times faster than the CPU implementation, for a simulation of one day with a time step of six seconds on a domain of size 2 mm by 1.6 mm. Here the GPU is used for the strain energy density computations, the concentration mappings and the computation of the right-hand side vector of the TGF-β system. The mesh is constant in this scenario, which allows us to store this information only once on the device. The largest speed-up occurs in the part of the Finite Element concentrations. The three aforementioned parts were very suitable to be parallelized as the individual cell computations are independent of each other. In other words, all information needed for the computation is available beforehand. Employment of the GPU causes significant speed-up, although with the high number of iterations the run-time is still quite large.

For larger domain sizes, the computational work load increases rapidly, as the computation times depends on the cell count and number of nodes in the Finite Element Mesh. For a domain with size 4 mm x 3.2 mm, a ten-day simulation increases to have a simulation time of 18 hours. With the goal of Monte Carlo simulations in mind, the domain size cannot increase much further. More efficiency measures need to be considered in order to reach a sufficiently fast model. The slowest part now is the solving of the concentration fields. After finding a suitable implementation for this part on the GPU, we could consider transferring to a complete program on the GPU.

In order to assess the accuracy of the concentration fields, it was attempted to estimate the order with Richardson's Extrapolation. The L^2-norm was computed for meshes with different grid sizes after a simulated time of one hour. The order estimation was attempted on both an unstructured grid as well as a structured grid, however, neither gave reasonable results. Even the estimation of the order by looking at specific locations did not give good results. The memory capacity on the used machine limits more precise computations. It remains unclear why the approximations do not work.

We have investigated an alternate method for computing the TGF-β field. The field is found by using Green's functions for the concentration. In order to approximate this, we sum up all sources that were exerted by macrophages. This needs to be done for all macrophages and for all past iterations during the lifetime of the macrophages. The field seemed to be comparable with the field obtained by the Finite Element approach. However, the computational work increases very rapidly. Even by speeding up computations on the GPU, this method did not turn out to be feasible in terms of computational work.

By having three separate kernels that communicate with the GPU, we are transferring data three times as well. For relatively small tasks, the communication can take more computational effort than the actual computations on the GPU. Before transferring the mesh information to the device only once, the GPU implementation of generating the right-hand side vector of the TGF-β system consisted of communication for almost 97%. The communication times are a significant part of the implementations for the right-hand side vector and to a lesser extent of the strain energy density computation. On the other hand, the work in these parts occupies only a small fraction of the total work load. As other parts in the program take more time, the communication times are not problematic for the current model.

With a strict criterion that does not allow cells to move more than half of their radius, the maximal time step is about three seconds. For a time step of six seconds, we allow the maximal displacement to be twice as large. By the Kolmogorov-Smirnov test, the scenarios with time steps of three and six seconds are compared. With a confidence level of 95%, the hypothesis that the underlying densities are the same is accepted and therefore a time step of six seconds is used.

The real type parameter in the model can be set to doubles or floats, and thus the program can be run in double precision or in single precision, respectively. The single precision implementation caused a slowdown of the program in the part where the concentration systems are solved. A cause for the problems might be that many values in the matrix and in the model in general are very small and are rounded to zero faster in single precision. Converting the parameters to different units such that the values become larger can be attempted to solve these problems. Furthermore, an implementation that runs partially in single precision and partially in double precision can be considered. Solving the concentration systems can then occur in double preci-

sion, while the other parts are run in single precision.

Finally, we have tested the influence of the approach that is chosen to scale the concentration gradients. The first method normalizes differently if the x-gradient and y-gradient are very small such that it will lead to division by zero, while the second method always adds a constant to the denominator to prevent division by zero. By the results of the Kolmogorov-Smirnov test, we conclude that the choice of the specific implementation of the normalization does not influence the wound healing process.

This research presents a framework that can be used for future testing of an extended model, with respect to precision and efficiency. The Python user interface makes it easy to run Monte Carlo simulations and display the results graphically. It is attempted to model the fibroblast migration during wound healing as realistic as possible. To overcome the mismatch between the model and reality, the parameters can be tuned according to specific experiments. The challenge is to link experiments to the model, while having different parameters and model output than measured in the experiments. Whenever more data becomes available, the model parameters can easily be adjusted. In the next chapter, improvements and ideas for future research are discussed.

13 Future work

In this chapter some interesting topics for future work will be discussed. Next to possible extensions of the model in this project, the current implementations will be reviewed and alternate methods will be described.

13.1. Extensions of current model

The final model that has been developed during this project needs to be extended with more features, in order to be able to realistically model the wound healing. The most important aspect is the addition of the mechanical field, as was already mentioned in Section 1.1 as one of the three main components. With this extension, the grid will not be fixed anymore, as it will deform as the result of mechanical influences. This will allow us to track the wound boundary explicitly.

Another important extension will be the transition from the two-dimensional model to a three-dimensional model. This will again make the model more realistic and will give results different from the 2D models. The interactions between cells will also change, many more cells can be in contact in a 3D setting and more cells can be within the communication range of a certain cell. An advantage of a model in 3D is that it is easier to link to experiments, as the quantities in experiments are often given as a volume. For example, the fibroblast density is usually expressed in volume instead of surface and concentrations are often given per milliliter.

In this report, the strain energy density signal is modelled by exponential decay. The most simple, more realistic choice would be to model the strain energy as $E_s = \frac{1}{2}\underline{\underline{\sigma}} : \underline{\underline{\epsilon}}$, where : denotes the dyadic product. Here $\underline{\underline{\epsilon}} \approx \frac{1}{2}(\nabla \underline{u} + \nabla \underline{u}^T)$, under the assumption of infinitesimal strain, with $\underline{u}$ the displacement vector. From Hooke's law, it can then be found that $\underline{\underline{\sigma}} = 2\mu\underline{\underline{\epsilon}} + \lambda \mathrm{Tr}(\underline{\underline{\epsilon}})\underline{\underline{I}}$, where μ and λ are the Lamé constants that are linked to the Young's modulus and the Poisson ratio and $\underline{\underline{I}}$ is the second-rank identity tensor. The final goal would be to use morpho-elasticity. This will allow for permanent deformation, which will occur for tissue growth. To implement this, again different equations from the earlier mentioned ones would have to be used.

13.2. Efficiency considerations

A large part of this thesis is focused on the performance of the model. The use of the GPU improves the computational work load significantly, although the total computation time remains still large. This is mainly caused by the fact that for a simulation of ten days, we need 144,000 iterations for a time step of six seconds. Furthermore, by doing Monte Carlo analysis, we need a lot of repetitions of this simulation. For increasing problem size and extensions of the model, the work load will only increase and extra measures will have to be taken to preserve the efficiency.

In a three-dimensional model, we need to re-evaluate the choices made for the current model. Whereas in the current model a direct solver is the fastest option, this will probably be an iterative solver in 3D (especially with a non-constant grid). Furthermore, we could consider computing a stress field over the membrane by using a Finite Element solution for the membrane equation. This would take away the increasing computational work that is needed for the intracellular communication through the current approach of cell migration

for larger cell populations.

Another approach to obtain speed-up would be the multiscale fast summation for dipoles [32]. This method is based on an N-body particle system for which we have to compute pairwise interactions. Instead of the order $\mathcal{O}(N^2)$ that occurs for simply evaluating every pair, this algorithm has an order $\mathcal{O}(N)$ complexity. If there are several particles in a grid, then these are combined to super-particles on a coarser grid. The super-particles agree with the original particles except for some aggregation error. In order to compute something at a certain location, we are interpolating and thus having an interpolation error.

In this research, we have used the GPU at three parts of the program, to speed up the computations in that specific part. For most scenario set-ups (with fine meshes that have a maximal triangle area of 10^{-11} m^2) the solving of the concentration field systems becomes the bottleneck. The next logical step would therefore be to implement a GPU version of the solver. Possible methods for this are the Mixed Precision Techniques for the GPU [38]. With these techniques, some part of an iterative method is performed on the GPU in single precision and the other part is performed on the CPU in double precision. Hereby, we still guarantee enough precision, while speeding up computations.

Another part that covers a significant part of the computations are the checks that are performed to determine if a cell lives in a certain mesh triangle. The straightforward implementation can possibly be improved by a smarter algorithm. As we are modelling the cell interactions explicitly, the maximal time step is limited to prevent unrealistic behavior. In order to make the simulation more efficient, we can put more parts of the program on the GPU. At some point, the transfer times of all separate parts will add up to be a considerable amount. The next step would then be to compute everything on the GPU to reach sufficient speed-up. In that case, no communication is needed anymore.

Lastly, if we want to run the program on multiple different platforms, it is possible that the code needs to be adjusted to run it on the respective platform. In this case, it can be beneficial to reconsider OpenCL, as it supposed to be better for portability. The transition from CUDA to OpenCL should be doable without too drastic changes.

13.3. Accuracy considerations

Now the simple implementation as described in Section 4.3 is used to find the gradient. The error estimate should be $\mathcal{O}(h)$. Finding the estimated order of the error by using Richardson's Extrapolation did not give convincing results. What the exact problem is by determining this order can be further investigated.
For more accurate approximations of the gradient we can do the following. The gradient can be computed on the nodes first and with that information determine the gradient in the cell centers.
Let $p = \frac{\partial c}{\partial x}$, then $\int_\Omega p\,\phi\,d\Omega = \int_\Omega \frac{\partial c}{\partial x}\,\phi\,d\Omega$.
Approximate with $p(x,y) \approx p^n(x,y) = \sum_{j=1}^n p_j(t)\phi_j(x)$ and let $\phi = \phi_i$, where $\phi_i = \alpha_i + \beta_i x + \gamma_i y$ is a basis function. It follows that $\sum_{j=1}^n p_j \int_\Omega \phi_i\phi_j d\Omega = \sum_{j=1}^n c(t,x_j)\int_\Omega \frac{\partial \phi_j}{\partial x}\phi_i d\Omega$.
In a similar manner, we let $q = \frac{\partial c}{\partial y}$ and end up with the equations :

$$\sum_{j=1}^n q_j \int_\Omega \phi_i\phi_j d\Omega = \sum_{j=1}^n c(t,x_j)\int_\Omega \frac{\partial \phi_j}{\partial y}\phi_i d\Omega.$$

These systems are of the form $\begin{cases} \sum_{j=1}^n S_{ij}^x p_j = \sum_{j=1}^n C_{ij}^x c(t,x_j), \\ \sum_{j=1}^n S_{ij}^y p_j = \sum_{j=1}^n C_{ij}^y c(t,x_j), \end{cases}$, where

$$S_{ij}^x = S_{ij}^y = \int_\Omega \phi_i\phi_j d\Omega = \frac{|\Delta e_k|}{24}(1+\delta_{ij}),$$
$$C_{ij}^x = \int_\Omega \frac{\partial \phi_j}{\partial x}\phi_i d\Omega = \frac{|\Delta e_k|}{6}\beta_j,$$
$$C_{ij}^y = \int_\Omega \frac{\partial \phi_j}{\partial y}\phi_i d\Omega = \frac{|\Delta e_k|}{6}\gamma_j.$$

After obtaining the gradient of c in all the nodes, we can determine the gradient at the cell centers x by:

$$\frac{\partial c}{\partial x}(t,x) = p(t,x) = \sum_{l \in \{p_1,p_2,p_3\}} p(t,x_l)\phi_l(x),$$
$$\frac{\partial c}{\partial y}(t,x) = q(t,x) = \sum_{l \in \{p_1,p_2,p_3\}} q(t,x_l)\phi_l(x).$$

The disadvantage of this approach is that we need to run a Finite Element program twice, once to find the concentration and once to find the concentration gradient. This might be computationally expensive and thus advocates for another approach, such as the use of mixed finite elements (with Raviart-Thomas elements).

13.4. On the practical side

For doctors it is above all important to know more about the complications caused by burn injuries (wound contraction and hypertrophic scars). Important results would be the magnitude of stress that occurs during contraction or the permanent reduction of the skin by contraction. After extension of the model, new wound quantifiers would need to be defined in order to give this information. In the end, we want to use the model to test different therapies for a specific burn injury. With Monte Carlo simulations, we can predict which treatment has the highest probability on recovery without complications.

To reduce the mismatch between the model and the reality, we need more accurate estimates for the parameters. Specific data is hard to obtain, therefore, we need to perform parameter sensitivity analysis. It is important to find out which parameters have a large influence on the rate of wound healing. In order to investigate this matter, we can choose one of the wound quantifiers and plot the behavior over time for different values of a certain parameter. As we are dealing with stochastic processes and with uncertainty in the input parameters (which are often patient-specific), we need to perform Monte Carlo simulations and use the estimated mean and confidence interval for comparison. Eventually, we would like to predict the likelihood that complications such as contractions and hypertrophic scarring occur. Furthermore, correlations between several input and output variables can be determined.

In this research only one parameter has been tested for its sensitivity, although ideally this is performed for all important parameters in the model. For some parameters, we can estimate the impact of deviations from the correct value. For example, the exact value of the maximal detectable range of cells to communicate with each other does not seem very important. The influences from cells outside this radius do barely contribute to the total strain energy density of a cell (in the wound healing set-up). Therefore, making the radius a bit smaller or larger will not have much impact. On the other hand, the parameter that determines the amount of TGF-β that a macrophage secretes will probably have a lot of impact.

Furthermore, we want to have a basic validation with experiments, in order to tune the parameters in the model. An experimental set-up can be quite specific and might be hard to translate to the mathematical model. For example, temperature and medium composition are no input parameters in the model, but are important during an experiment. More research can be done to find experiments that can be linked to the model. The other way around, we can consider tuning the model such that it can be linked to experiments as well.

13.5. Open questions

The computations in single precision did not result in the expected speed-up. For the strain energy parts the speed-up is only a small fraction, which is less than expected. Furthermore, the solve time of the FEM systems in the single precision implementation caused unexpected behaviour. The computation time increased with a factor two instead of being faster. It is interesting to further investigate the cause for this behavior. For simple test matrices, the SparseLU solver is faster in single precision compared to double precision. It is possible that the system is not well-defined, such that small perturbations in the right-hand side have a large effect on the solution. Most entries of the matrix are quite small and this might be the cause of different results in single precision. A recommendation would be to convert all parameters to a different scale, such that the values are larger. Now the values are given in the standard units (kilograms, meters and seconds),

but it would be better to express the parameters in grams, millimeters and hours. After this, we can again investigate if running the model in single precision gives reasonable results.

A

Finite Element Method derivations

A.1. Platelet Derived Growth Factor

We consider the following system for Platelet Derived Growth Factor (PDGF):

$$\begin{cases} \frac{\partial c_P}{\partial t} - D_{c_P}\Delta c_P = 0\ ,\ t>0\ ,\ x\in\Omega \\ D_{c_P}\frac{\partial c_P}{\partial n} + \kappa c_P = 0\ ,\ t>0\ ,\ x\in\partial\Omega, \\ c_P(0,x) = f(x). \end{cases}$$

It is assumed that there is an absence of sources and the initial concentration is $f(x)$, which was described in Section 4.3.1. The boundary condition is a homogeneous Robin boundary condition. Therefore, the function space in which the solution is searched, is a linear space. With this boundary condition, the concentration of PDGF vanishes far away from the wound.

The weak formulation for $(BVP1)$: $\begin{cases} \frac{\partial c_P}{\partial t} - D_{c_P}\Delta c_P = 0 & \text{in } \Omega, \\ D_{c_P}\frac{\partial c_P}{\partial n} + \kappa c_P = 0\ , & \text{on } \partial\Omega, \\ c_P(0,x) = f(x) & \text{in } \Omega, \end{cases}$
is found in the following steps (by multiplying with the test function ϕ and integrating over the domain):

$$\begin{aligned} \int_\Omega \phi\frac{\partial c_P}{\partial t} - \phi D_{c_P}\Delta c_P\ d\Omega &= \mathbf{0} \\ \int_\Omega \phi\frac{\partial c_P}{\partial t} d\Omega - \int_\Omega D_{c_P}(\nabla(\phi\nabla c_P) - \nabla\phi\nabla c_P) d\Omega &= \mathbf{0} \\ \int_\Omega \phi\frac{\partial c_P}{\partial t} d\Omega - \int_{\partial\Omega} D_{c_P}(\phi\frac{\partial c_P}{\partial n}) d\Gamma + \int_\Omega D_{c_P}\nabla\phi\nabla c_P d\Omega &= \mathbf{0} \\ \int_\Omega \phi\frac{\partial c_P}{\partial t} + D_{c_P}\nabla\phi\nabla c_P\ d\Omega + \int_{\partial\Omega}\phi\kappa c_P d\Gamma &= \mathbf{0} \end{aligned}$$

This leads to the weak formulation:
$(W1)$: Find $c_1 \in L^2((0,T), H^1(\Omega))$ such that $\int_\Omega \phi\frac{\partial c_P}{\partial t} + D_{c_P}\nabla\phi\nabla c_P\ d\Omega + \int_{\partial\Omega}\phi\kappa c_P d\Gamma = \mathbf{0}$ with $c_P(x,t) = f(x)$ holds true $\forall\phi\in H^1(\Omega)$.

Let $c_P(x,y) \approx c^n(x,y) = \sum_{j=1}^n d_j(t)\phi_j(x)$ and let $\phi = \phi_i$, where $\phi_i = \alpha_i + \beta_i x + \gamma_i y$ is a basis function. Filling in in the weak form gives:

$$\int_\Omega \phi_i\frac{\partial(\sum_{j=1}^n d_j(t)\phi_j(x))}{\partial t} + D_{c_P}\nabla\phi_i\nabla(\sum_{j=1}^n d_j(t)\phi_j(x))\ d\Omega + \int_{\partial\Omega}\kappa\phi_i\sum_{j=1}^n d_j(t)\phi_j(x)d\Gamma = \mathbf{0}.$$

The Galerkin equations become

$$\sum_{j=1}^{n} d_j'(t)\int_\Omega \phi_i\phi_j d\Omega + \sum_{j=1}^{n} d_j(t)\left[\int_\Omega D_{c_P}\nabla\phi_i\nabla\phi_j\, d\Omega + \int_{\partial\Omega}\kappa\phi_i\phi_j d\Gamma\right] = \mathbf{0}.$$

This is of the form $\sum_{j=1}^{n} M_{ij}d_j' + \sum_{j=1}^{n} S_{ij}d_j = 0$. Here, S_{ij} is called the stiffness matrix. The separate components are calculated and displayed below:

$$\begin{aligned}
M_{ij}^{e_k} &= \int_{e_k}\phi_i\phi_j\, d\Omega = \frac{|\Delta e_k|}{24}(1+\delta_{ij}),\\
S_{ij}^{e_k} &= \int_{e_k} D_{c_P}\nabla\phi_i\nabla\phi_j\, d\Omega = \int_{e_k} D_{c_P}(\beta_i\beta_j+\gamma_i\gamma_j)\, d\Omega = \frac{|\Delta e_k|}{2}D_{c_P}(\beta_i\beta_j+\gamma_i\gamma_j),\\
S_{ij}^{be} &= \int_{be}\kappa\phi_i\phi_j\, d\Gamma = \kappa\frac{\|x_{l1}-x_{l2}\|}{6}(1+\delta_{ij}).
\end{aligned}$$

The other terms are zero.

A.2. Transforming Growth Factor β

We consider the following system for Transforming Growth Factor β (TGF-β):

$$\begin{cases}
\frac{\partial c_\beta}{\partial t} - D_{c_\beta}\Delta c_\beta = \kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)\ ,\ t>0\ ,\ x\in\Omega,\\
D_{c_\beta}\frac{\partial c_\beta}{\partial n} + \kappa c_\beta = 0\ ,\ t>0\ ,\ x\in\partial\Omega,\\
c_\beta(x,t) = 0.
\end{cases}$$

The macrophages secrete TGF-β and thus does the source term exist of a summation over their locations. It is assumed that initially there were no macrophages present and therefore the concentration is set to zero.

The weak formulation for $(BVP2)$: $\begin{cases}
\frac{\partial c_\beta}{\partial t} - D_{c_\beta}\Delta c_\beta = \kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t) & \text{in } \Omega,\\
D_{c_\beta}\frac{\partial c_\beta}{\partial n} + \kappa c_\beta = 0\,, & \text{on } \partial\Omega,\\
c_\beta(x,0) = 0 & \text{in } \Omega,
\end{cases}$
is found in the following steps (by multiplying with the test function ϕ and integrating over the domain):

$$\begin{aligned}
\int_\Omega \phi\frac{\partial c_\beta}{\partial t} - \phi D_{c_\beta}\Delta c_\beta\, d\Omega &= \int_\Omega \phi\kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)d\Omega\\
\int_\Omega \phi\frac{\partial c_\beta}{\partial t}d\Omega - \int_\Omega D_{c_\beta}(\nabla(\phi\nabla c_\beta) - \nabla\phi\nabla c_\beta)d\Omega &= \int_\Omega \phi\kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)d\Omega\\
\int_\Omega \phi\frac{\partial c_\beta}{\partial t}d\Omega - \int_{\partial\Omega} D_{c_\beta}(\phi\frac{\partial c_\beta}{\partial n})d\Gamma + \int_\Omega D_{c_\beta}\nabla\phi\nabla c_\beta d\Omega &= \int_\Omega \phi\kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)d\Omega\\
\int_\Omega \phi\frac{\partial c_\beta}{\partial t} + D_{c_\beta}\nabla\phi\nabla c_\beta\, d\Omega + \int_{\partial\Omega}\phi\kappa c_\beta d\Gamma &= \int_\Omega \phi\kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)d\Omega
\end{aligned}$$

This leads to the weak formulation
$(W2)$: Find $c_2\in L^2((0,T),H^1(\Omega))$ such that $\int_\Omega \phi\frac{\partial c_\beta}{\partial t} + D_{c_\beta}\nabla\phi\nabla c_\beta\, d\Omega + \int_{\partial\Omega}\phi\kappa c_\beta d\Gamma = \int_\Omega \phi\kappa_{c_\beta}\sum_{t=1}^{M}\delta(x-x_M^t)d\Omega$ with $c_\beta(x,t)=0$ holds true $\forall\phi\in H^1(\Omega)$.

Let $c_\beta(x,y)\approx c^n(x,y) = \sum_{j=1}^{n} d_j(t)\phi_j(x)$ and let $\phi=\phi_i$, where $\phi_i = \alpha_i+\beta_i x+\gamma_i y$ is a basis function. Filling in in the weak form gives:

$$\int_\Omega \phi_i\frac{\partial(\sum_{j=1}^{n} d_j(t)\phi_j(x))}{\partial t} + D_{c_\beta}\nabla\phi_i\nabla(\sum_{j=1}^{n} d_j(t)\phi_j(x))\, d\Omega + \int_{\partial\Omega}\kappa\phi_i\sum_{j=1}^{n} d_j(t)\phi_j(x)d\Gamma = \int_\Omega\sum_{t=1}^{M}\kappa_{c_\beta}\phi_i\delta(x-x_M^t)d\Omega.$$

The Galerkin equations become

$$\sum_{j=1}^{n} d_j'(t) \int_\Omega \phi_i \phi_j d\Omega + \sum_{j=1}^{n} d_j(t) \left[\int_\Omega D_{c_\beta} \nabla\phi_i \nabla\phi_j \, d\Omega + \int_{\partial\Omega} \kappa \phi_i \phi_j d\Gamma \right] = \int_\Omega \sum_{t=1}^{M} \kappa_{c_\beta} \phi_i(x_M^t) d\Omega.$$

This is of the form $\sum_{j=1}^{n} M_{ij} d_j' + \sum_{j=1}^{n} S_{ij} d_j = f_i$. Here, S_{ij} and f_i are called the stiffness matrix and vector respectively. The separate components are calculated and displayed below:

$$M_{ij}^{e_k} = \int_{e_k} \phi_i \phi_j \, d\Omega = \frac{|\Delta e_k|}{24}(1+\delta_{ij}),$$

$$S_{ij}^{e_k} = \int_{e_k} D_{c_\beta} \nabla\phi_i \nabla\phi_j \, d\Omega = \int_{e_k} D_{c_\beta}(\beta_i\beta_j + \gamma_i\gamma_j) \, d\Omega = \frac{|\Delta e_k|}{2} D_{c_\beta}(\beta_i\beta_j + \gamma_i\gamma_j),$$

$$S_{ij}^{be} = \int_{be} \kappa \phi_i \phi_j \, d\Gamma = \kappa \frac{\|x_{l1} - x_{l2}\|}{6}(1+\delta_{ij}),$$

$$f_i^{e_k} = \int_{e_k} \sum_{t=1}^{M} \kappa_{c_\beta} \phi_i(x_M^t) = \sum_{t=1}^{M} \kappa_{c_\beta} \phi_i(x_M^t), \quad i \in p_1, p_2, p_3 \qquad \Rightarrow \quad f^{e_k} = \begin{bmatrix} \sum_{t=1}^{M} \kappa_{c_\beta} \phi_{p_1}(x_M^t) \\ \sum_{t=1}^{M} \kappa_{c_\beta} \phi_{p_2}(x_M^t) \\ \sum_{t=1}^{M} \kappa_{c_\beta} \phi_{p_3}(x_M^t) \end{bmatrix}.$$

The other terms are zero.

B

Richardson's Extrapolation

B.1. Results order estimation

Repeating the experiment that was described in Section 10.1.2 with both refinement of factor two and $\sqrt{2}$ gives the results as displayed in Table B.1. Here $Q(h)$ is the approximation of the L^2-norm for the respective grid size h.

h	# triangles	Δt	m	$Q(h)$
$32 \cdot 10^{-7}$	78814	0.9375	3840	$1.57888 \cdot 10^{-9}$
$16 \cdot 10^{-7}$	625000	0.234375	15360	$8.81684 \cdot 10^{-9}$
$\sqrt{2} \cdot 8 \cdot 10^{-7}$	1249976	0.1171875	30720	$8.50564 \cdot 10^{-9}$
$8 \cdot 10^{-7}$	2500000	0.05859375	61440	$2.36812 \cdot 10^{-9}$

Table B.1: One hour simulations for estimating the order of the error by reducing the grid size h with factor two and factor $\sqrt{2}$.

Choosing $h_1 = 32 \cdot 10^{-7}$, $h_2 = 16 \cdot 10^{-7}$ and $h_3 = 8 \cdot 10^{-7}$, we obtain $\frac{Q(h_2)-Q(h_1)}{Q(h_3)-Q(h_2)} = \frac{8.81684 \cdot 10^{-9} - 1.57888 \cdot 10^{-9}}{2.36812 \cdot 10^{-9} - 8.81684 \cdot 10^{-9}} = -23.258$. Doing the same for refinements with factor $\sqrt{2}$ and thus having $h_1 = 16 \cdot 10^{-7}$, $h_2 = \sqrt{2} \cdot 8 \cdot 10^{-7}$ and $h_3 = 8 \cdot 10^{-7}$, we obtain $\frac{Q(h_2)-Q(h_1)}{Q(h_3)-Q(h_2)} = \frac{8.50564 \cdot 10^{-9} - 8.81684 \cdot 10^{-9}}{2.36812 \cdot 10^{-9} - 8.50564 \cdot 10^{-9}} = 0.0507$.

Both estimations of the order do not give reasonable results for these values. It is also attempted to find the order by looking at the concentrations at specific locations. Eight locations in the middle of the domain are chosen, with coordinates as follows: P1 = ($2.3 \cdot 10^{-6}$, $1.2 \cdot 10^{-6}$) , P2 = ($109 \cdot 10^{-6}$, $2.8 \cdot 10^{-6}$), P3 = ($11.3 \cdot 10^{-6}$, $209 \cdot 10^{-6}$), P4 = (-$254 \cdot 10^{-6}$, $210 \cdot 10^{-6}$), P5 = (-$212 \cdot 10^{-6}$, -$134 \cdot 10^{-6}$), P6 = ($56.2 \cdot 10^{-6}$, -$151 \cdot 10^{-6}$), P7 = ($301 \cdot 10^{-6}$, $157 \cdot 10^{-6}$) and P8 = (-$182 \cdot 10^{-6}$, $335 \cdot 10^{-6}$).
The results of this are displayed in Table B.2. Again, the estimations of the order give unfeasible solutions, as is displayed in the last two columns.

P	Q(h_0)	Q(h_1)	Q(h_2)	Q(h_3)	$\frac{Q(h_1)-Q(h_0)}{Q(h_2)-Q(h_1)}$	$\frac{Q(h_2)-Q(h_1)}{Q(h_3)-Q(h_2)}$
1	$7.13155131 \cdot 10^{-6}$	$7.13207305 \cdot 10^{-6}$	$7.13221148 \cdot 10^{-6}$	$7.1322245 \cdot 10^{-6}$	3.76898	10.6321
2	$6.45256581 \cdot 10^{-6}$	$6.45222084 \cdot 10^{-6}$	$6.4522938 \cdot 10^{-6}$	$6.45232088 \cdot 10^{-6}$	-4.72821	2.69424
3	$4.15053533 \cdot 10^{-6}$	$4.15092668 \cdot 10^{-6}$	$4.15096539 \cdot 10^{-6}$	$4.15099858 \cdot 10^{-6}$	10.1098	1.16632
4	$2.25876871 \cdot 10^{-6}$	$2.25853067 \cdot 10^{-6}$	$2.25851734 \cdot 10^{-6}$	$2.25848878 \cdot 10^{-6}$	17.8575	0.466737
5	$3.83937763 \cdot 10^{-6}$	$3.83981093 \cdot 10^{-6}$	$3.83977142 \cdot 10^{-6}$	$3.83978096 \cdot 10^{-6}$	-10.9668	-4.14151
6	$5.28696482 \cdot 10^{-6}$	$5.28727632 \cdot 10^{-6}$	$5.28729514 \cdot 10^{-6}$	$5.28732781 \cdot 10^{-6}$	16.5515	0.576064
7	$2.19654773 \cdot 10^{-6}$	$2.19681688 \cdot 10^{-6}$	$2.19689303 \cdot 10^{-6}$	$2.19692331 \cdot 10^{-6}$	3.53447	2.51486
8	$9.90517531 \cdot 10^{-7}$	$9.90548219 \cdot 10^{-7}$	$9.90500488 \cdot 10^{-7}$	$9.9050111 \cdot 10^{-7}$	-0.642936	-76.7379

Table B.2: One hour simulations for estimating the order of the error by comparing concentrations at specific locations.

Bibliography

[1] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. Extracellular Control of Cell Division, Cell Growth, and Apoptosis. 2002. URL https://www.ncbi.nlm.nih.gov/books/NBK26877/.

[2] P. Bainbridge. Wound healing and the role of fibroblasts. *Journal of Wound Care*, 22(8):407–412, 2013. doi: 10.12968/jowc.2013.22.8.407. URL https://doi.org/10.12968/jowc.2013.22.8.407.

[3] D. Brent Polk and Mark R. Frey. Mucosal Restitution and Repair. In *Physiology of the Gastrointestinal Tract*. 2012. ISBN 9780123820266. doi: 10.1016/B978-0-12-382026-6.00042-7.

[4] Jiao Chen, Daphne Weihs, and Fred J Vermolen. A model for cell migration in non-isotropic fibrin networks with an application to pancreatic tumor islets. *Biomechanics and modeling in mechanobiology*, 17(2):367–386, 2018.

[5] Steven Cook and Tamar Shinar. Enabling Simulation of High-Dimensional Micro-Macro Biophysical Models through Hybrid CPU and Multi-GPU Parallelism. 2017.

[6] D. Cukjati, S. Rebersek, and D. Miklavcic. A reliable method of determining wound healing rate. *Med. Biol. Eng. Comput*, 39:263–271, 2001.

[7] Lorenzo Dematté and Davide Prandi. GPU computing for systems biology. *Briefings in Bioinformatics*, 2010. ISSN 14675463. doi: 10.1093/bib/bbq006.

[8] Dirk Drasdo and Stefan Höhme. A single cell based-model of tumor growth in-vitro: monolayers and speroids. *Physical biology*, 2(3):133, 2005.

[9] M. Dudaie, D. Weihs, F. J. Vermolen, and A. Gefen. Modeling migration in cell colonies in two and three dimensional substrates with varying stiffnesses. *In Silico Cell and Tissue Science*, 2015. ISSN 2196-050X. doi: 10.1186/s40482-015-0005-9.

[10] Nicholas D. Evans, Richard O. C. Oreffo, Eugene Healy, Philipp J. Thurner, and Yu Hin Man. Epithelial mechanobiology, skin wound healing, and the stem cell niche. 2013. doi: 10.1016/j.jmbbm.2013.04.023. URL www.elsevier.com/locate/jmbbmwww.sciencedirect.comhttp://dx.doi.org/10.1016/j.jmbbm.2013.04.023.

[11] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A Comprehensive Performance Comparison of CUDA and OpenCL. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 216–225, 2011.

[12] J. Galle, G. Aust, G. Schaller, T. Beyer, and D. Drasdo. Individual Cell-Based Models of the Spatial- Temporal Organization of Multicellular Systems—Achievements and Limitations. *Cytometry Part A*, 69(7): 704–710, 2006.

[13] Alexander Golberg, Marianna Bei, Robert L Sheridan, and Martin L Yarmush. Regeneration and control of human fibroblast cell density by intermittently delivered pulsed electric fields. *Biotechnology and bioengineering*, 110(6):1759–1768, 2013.

[14] Ayman Grada, Marta Otero-Vinas, Francisco Prieto-Castrillo, Zaidal Obagi, and Vincent Falanga. Research techniques made simple: Analysis of collective cell migration using the wound healing assay. *Journal of Investigative Dermatology*, 137(2):e11–e16, 2017.

[15] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[16] Mads Haahr. Introduction to randomness and random numbers. *Random. org, June*, 1999.

[17] Richard Haberman. *Applied partial differential equations with Fourier series and boundary value problems*. Pearson Education Limited, 2014.

[18] Mark Harris. How to Implement Performance Metrics in CUDA C/C++, 2012. URL `https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/`.

[19] Cédric Jules. Accurate point in triangle test, 2014. URL `http://totologic.blogspot.nl/2014/01/accurate-point-in-triangle-test.html`.

[20] E.D. Kleimann. Literature report: Mathematical modelling of burn injuries. Master's thesis, TU Delft, 2018.

[21] D.C. Koppenol. *Biomedical implications from mathematical models for the simulation of dermal wound healing*. PhD thesis, TU Delft, 2017.

[22] Tara LaForce. PE281 Green's Functions Course Notes. 2006.

[23] C.H. Marchi, L.K. Araki, A.C. Alves, R. Suero, S.F.T. Gonçalves, and M.A.V. Pinto. Repeated Richardson extrapolation applied to the two-dimensional Laplace equation using triangular and square grids. *Applied Mathematical Modelling*, 37(7):4661–4675, 2013.

[24] Suejb Memeti, Lu Li, Sabri Pllana, Joanna Kolodziej, and Christoph Kessler. Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption. In *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, pages 1–6, 2017. ISBN 9781450351164.

[25] C.C. Miller, G. Godeau, C. Lebreton-DeCoster, A. Desmouliere, B. Pellat, L. Dubertret, and B. Coulomb. Validation of a morphometric method for evaluating fibroblast numbers in normal and pathologic tissues. *Experimental dermatology*, 12(4):403–411, 2003.

[26] Kelly E. Murphy, Cameron L. Hall, Philip K. Maini, Scott W. McCue, and D.L. Sean McElwain. A fibrocontractive mechanochemical model of dermal wound closure incorporating realistic growth factor kinetics. *Bulletin of mathematical biology*, 74(5):1143–1170, 2012.

[27] Marco S. Nobile, Paolo Cazzaniga, Andrea Tangherloni, and Daniela Besozzi. Graphics processing units in bioinformatics, computational biology and systems biology. *Briefings in Bioinformatics*, 2016. ISSN 1467-5463. doi: 10.1093/bib/bbw058.

[28] Luke Olsen, Jonathan A Sherratt, and Philip K Maini. A mechanochemical model for adult dermal wound contraction and the permanence of the contracted tissue displacement profile. *Journal of theoretical biology*, 177(2):113–128, 1995.

[29] OpenStax. Anatomy and Physiology, 2013. URL `http://cnx.org/contents/14fb4ad7-39a1-4eee-ab6e-3ef2482e3e22@8.24`.

[30] Cynthia A. Reinhart-King, Micah Dembo, and Daniel A. Hammer. Cell-cell mechanical communication through compliant substrates. *Biophysical Journal*, 2008. ISSN 15420086. doi: 10.1529/biophysj.107.127662.

[31] Christopher Roy. Review of discretization error estimators in scientific computing. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 126, 2010.

[32] Bilha Sandak. Multiscale fast summation of long-range charge and dipolar interactions. *Journal of Computational Chemistry*, 22(7):717–731, 2001.

[33] Jonathan Richard Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied computational geometry towards geometric engineering*, pages 203–222. Springer, 1996.

[34] Y. Song, S. Yang, and J. Lei. ParaCells: A GPU Architecture for Cell-Centered Models in Computational Biology. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PP(99):1–1, 2018. doi: 10.1109/TCBB.2018.2814570.

[35] José Juan Tapia and Roshan M. D'Souza. Parallelizing the Cellular Potts Model on graphics processing units. *Computer Physics Communications*, 2011. ISSN 00104655. doi: 10.1016/j.cpc.2010.12.011.

[36] F. J. Vermolen. Particle methods to solve modelling problems in wound healing and tumor growth. *Computational Particle Mechanics*, 2015. ISSN 21964386. doi: 10.1007/s40571-015-0055-6.

[37] F. J. Vermolen and A. Gefen. A semi-stochastic cell-based formalism to model the dynamics of migration of cells in colonies. *Biomechanics and Modeling in Mechanobiology*, 2012. ISSN 16177959. doi: 10.1007/s10237-011-0302-6.

[38] C. Vuik and C.W.J. Lemmens. Programming on the GPU with CUDA (version 6.5). 2015.

[39] C Vuik, F.J. Vermolen, M.B. van Gijzen, and M.J. Vuik. *Numerical Methods for Ordinary differential equations*. Delft Academic Press / VSSD, 2016.

[40] Nathan Whitehead and Alex Fit-Florea. Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs. *rn (A+ B)*, 21(1):18749–19424, 2011. URL `https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf`.

[41] Karen Willcox and Qiqi Wang. 16.90 Computational Methods in Aerospace Engineering, 2014. URL `(MassachusettsInstituteofTechnology:MITOpenCouseWare),http://ocw.mit.edu(AccessedSeptember6,2018)`. License: Creative Commons BY-NC-SA.

[42] Le Yang, Tarynn M. Witten, and Ramana M. Pidaparti. A biomechanical model of wound contraction and scar formation. *Journal of Theoretical Biology*, 2013. ISSN 00225193. doi: 10.1016/j.jtbi.2013.03.013.

[43] Ian T. Young. Proof without prejudice: use of the Kolmogorov-Smirnov test for the analysis of histograms from flow systems and other sources. *Journal of Histochemistry & Cytochemistry*, 25(7):935–941, 1977.