Modelling Angiogenesis in Three Dimensions

Louis Parsonson, BSc

Thesis submitted to the University of Nottingham for the degree of

Doctor of Philosophy

July 2015

Abstract

The process through which new blood vessels are formed within the body is known as *angiogenesis*. An essential part of our survival, it has also been implicated more recently in many diseases both in terms of induced growth, and abnormal vascular structure.

Angiogenesis is characterized as two processes, the development of a vascular network during embryonic growth and the production of new blood vessels. This work focuses on the latter, and seeks to develop a robust, three-dimensional model for simulating blood vessel growth and the attendant processes of blood flow and mass transfer within the simulated system. A system was developed which utilises medical imaging scan data (specifically, MicroCT) as the initial conditions from which a network of vessels is grown. This is combined with GPU accelerated simulations of fluid dynamics, with the intention of providing a technique for future use in predictive medicine and therapeutic simulation.

Acknowledgements

The work presented here is the result of three and a half years at the University of Nottingham Computer Science department, funded by an Industrial CASE award from the Engineering & Physical Sciences Research Council (EPSRC). The work on cloud computing was completed in collaboration with Biotronics3D, with the use of their 3dNet Medical cloud-based medical imaging solution.

I would like to express my sincerest thanks to Dr. Bai Li, my supervisor, for all of her advice, support and patience throughout the process, without which this would not have been possible.

I would like to dedicate this to my father, for giving me his love of science, my mother for teaching me to believe in myself, and to Anna for always pushing me to be my best.

Contents

1 Introduction **7**

1.1 Angiogenesis in health and diseases 9

1.1.1 Mechanisms of angiogenesis 9

1.1.2 Angiogenesis in cancer 10

1.1.3 Anti-angiogenic therapy 11

1.1.4 Angiogenesis in neurodegenerative diseases 13

1.2 Angiogenesis in tissue engineering 14

1.2.1 Cell-based vascularization 14

1.2.2 Scaffold-based strategies 15

1.3 Challenges of vascular modelling 16

1.4 Thesis roadmap 18

2 Literature review **19**

2.1 Modelling angiogenesis 19

2.1.1 Anderson and Chaplain's model 20

2.1.2 Other modelling approaches 23

2.2 Simulating blood flow 27

3 Cloud medical imaging and visualisation **31**

3.1 Cloud computing 32

3.2 Medical imaging 37

3.3 Volume rendering 39

3.3.1 Texture-based volume rendering 39

3.3.2 Ray casting 40

3.3.3 Rendering depth 41

4 Modelling angiogenesis in three dimensions **44**

4.1 Endothelial tip cell movement 46

4.2 Vessel formation 50

4.3 Physical constraints and boundary conditions 52

5 Fluid dynamics simulation **55**

5.1 Growth factor diffusion 55

5.2 Flow and pressure 58

6 Implementing complex computation on the GPU **61**

6.1 The parallel programming model 62

6.2 Diffusion 64

6.2.1 Diffusion equation 64

6.2.2 Memory requirements 65

6.3 Pressure 66

6.4 Mass transfer 73

6.4.1 Mass transfer by advection 74

6.4.2 Delivery of mass to the body 78

6.5 Programming for the GPU 80
6.6 Unifying a modular system 82

7 Individualised modelling **86**
7.1 Introduction 86
7.2 Personalised medicine 87
7.3 Imaging 88
7.4 Limitations of image based methods 92

8 Modular Analysis on the Cloud **94**
8.1 3DNet 94
8.2 A flexible pipeline for medical image analysis 95
8.2.1 Event-driven workflow 97
8.2.2 Logic engine 98
8.2.3 Module management 101
8.2.4 Module wrapper 103
8.2.5 Exploiting the DICOM standard for binary file transfer . . . 106
8.2.6 Inter-component communication 108
8.2.7 Individualised modelling in the cloud 108

9 Conclusions and Future Work **110**
9.1 Future work 115
9.2 Tissue engineering 117
9.3 Cloud computing 118
9.4 A more general scheme for modular development 119

Publications **125**

List of Figures **126**

References **128**

Chapter 1

Introduction

Angiogenesis is the formation of new blood vessels in the human body. With ongoing research, evidence continues to mount up to implicate vascular structure and function, in particular the angiogenic process, in a long list of human diseases. This includes cancer, diabetes, cardiovascular diseases, and in recent years neurodegenerative diseases [1], [2]. Study of the formation of microvasculature, therefore, has important implications in diagnosis and treatment of diseases. It is no surprise, then, that recent years have seen an increase in research into the mechanisms behind the angiogenic process, as well as attempts to predict how various therapies affect it.

Simulations of angiogenesis in two dimensions are fairly common now, with the model developed by Anderson and Chaplain [3] being of particular note. These models are designed to replicate growth patterns observed *in vitro*; thus, their application is limited in real-world situations. Continued interest in vascular growth in cancer, in particular, has resulted in interest in simulations of the effectiveness of anti-angiogenic therapy [4] as well as chemotherapy [5].

This thesis describes how a CPU-based Cellular Potts model [6] of sprouting angiogenesis in three dimensions is combined with medical imaging techniques and fluid dynamics equations to create an individual-based GPU accelerated angiogenesis simulation. The use of the GPU, optimized for fast, highly parallel mathematical operations, provides an increase in simulation speed and balances resource requirements across hardware. Primarily, this work is concerned with modelling vascular growth at the microscopic level. It is becoming clearer that the structure and function of microvasculature, that is the smallest vessels within the human body, are implied in many diseases. However, the need for studying microvasculature is not limited to clinical diagnosis of disease. In tissue engineering, creating vascularized tissue is a considerable problem, limiting the size to which effective engineered tissues can be grown [7].

The system described here attempts to personalise simulations of angiogenesis using medical scan data. Specifically, microCT scans of resin cast rat cerebral vasculature are segmented to remove reconstruction artifacts and imported to instantiate nascent endothelial cells in a homogeneous three-dimensional grid representing the area over which the simulation is performed. A growth factor source is added, and simulation of steady production and diffusion of the vascular endothelial growth factor (VEGF) is performed on the GPU using the NVidia Compute Unified Device Architecture (CUDA) programming platform. Motion of individual endothelial cells is then tracked over the lifetime of the simulation towards the source of growth factor, incorporating sprouting and anastamosis (looping) events. The network generated is used for simulations of blood flow and mass transfer, also accelerated using graphics hardware.

1.1 Angiogenesis in health and diseases

Since the discovery of angiogenesis, research has continued to highlight the role of this process in disease, from implications of vascular abnomalies in Diabetic Retinopathy [8], Cardiovascular Diseases [9], Alzheimer's Disease[10], through to the uncontrolled angiogenic response initiated by tumors in cancer patients [1].

1.1.1 Mechanisms of angiogenesis

Angiogenesis is triggered by the release of angiogenic factors, such as Fibroblast Growth Factors, a family of proteins which stimulate fibroblast and endothelial cell growth. The binding of angiogenic factors to receptors in endothelial cells stimulates the cells into migrating towards the source of the factor. The cells are induced to release enzymes, which degrades the basement membrane of the parent vessel, thus enabling them to move towards the source of the growth factor.

Angiogenesis occurs in multiple stages. Sprouts are formed as cells migrate towards the source of the growth factor, led by the tip cell, which recruits more endothelial cells from the parent vessel. As the tip moves, the sprout has a tendency to split, thus creating the well-known branching structure associated with the vasculature.

From time to time, migrating sprouts come into contact with other sprouts and can join with them to form the characteristic looping of the microvasculature. These events are known as *anastamoses* and are a key factor of the vascular geometry, enabling bi-directional flow of blood through the system.

During embryo growth and development, the network of vasculature develops from a field of endothelial cells, during which the cells group together to form

the primitive vascular network which later develops into the functional systems our bodies rely upon. This process is also known as *vasculogenesis*. In developed bodies, however, angiogenesis occurs during wound healing in a process known as *neoangiogensis*. This form of angiogenesis is preceded by the release of chemical factors by the body when in need of vascularisation, for example when areas of the body exist in a hypoxic state. Of the triggers of neoangiogenesis, the most common are chemical factors released by various parts of the body. Initially isolated in the 1970s from tumor mass [11], new factors have been identified since then. Of note is Vascular Endothelial Growth Factor (VEGF), identified as a key element in the mechanism of tumor-induced angiogenesis [12].

1.1.2 Angiogenesis in cancer

The risk of developing some form of cancer of a lifetime is currently rated on average to be approximately 40%, or a 2 in 5 chance. This means that the majority of people will experience cancer either through developing it, or having a friend or relative who does. The chances of said cancer being fatal are 1 in 5, and it is estimated that more than 1.6 million new cases of cancer will be diagnosed in the US alone, with nearly 600 000 deaths, in 2014 [13].

Angiogenesis is of key interest in the study and treatment of cancer. Initially, tumors in the body are unable to grow to sizes greater than one or two cubic millimetres, a size limited by the diffusion limits of oxygen within the body. Tumors which outgrow this size begin to die on the inside as they are unable to secure adequate nutrients. However, if a tumor switches to the angiogenic phenotype the cells begin to secret angiogenic factors, such as VEGF. As these factors move

through the surrounding tissue, they encounter existing blood vessels and encourage the growth of new vasculature towards the tumor, thus securing a nutrient supply and creating the possibility of metastatis - in some cases, tumor cells can detatch and enter the bloodstream, spreading the cancer across the body, often with fatal consequences.

1.1.3 Anti-angiogenic therapy

Anti-angiogenic therapies are an area of recent interest in terms of treating cancer, particularly in malignant glioma where angiogenesis is considered to be the key event in tumor progression [14]. Recent evidence has shown that antiangiogenic therapy can work to normalize tumor vasculature (i.e. making it more regular and efficient) which in turn causes a more effective platform for targeted cytotoxic therapies such as chemo- and radiotherapy, both of which require adequate functional vasculature to maintain efficacy [15]. However, application of antiangiogenics carries an inherent risk since giving to high a dose can lead to destruction of too much of the tumor vasculature, making cytotoxic therapies ineffective and leading to toxicity in normal tissues. Anti-angiogenic therapy has shown promise in terms of reducing the abilities of tumours to initiate vasculaturization, however recent research has shown that tumors can become resistant to the therapy through upregulation of alternative pro-angiogenic signals [16]. As such, anti-angiogenic therapies as they stand provide a respite, but not a cure.

Modelling of such therapies is of key interest in the continuing struggle against cancer. Arakelyan et al [4] developed a numerical model which was comprised of a set of formulae describing each of the interactions in their system. The results

were in the form of numerical values of measured quantities, such as growth factor, tumor size, effective vascular density and mature vessel density.

Ledzewicz et al [17] used a numerical method to solve a mathematical description of cancer treatment, that is solving for a maximum reduction in tumour volume, to provide dosages of anti-angiogenic drugs. More recently, Stefanini et al [18] detailed a model of VEGF distribution and interactions in tissue with the goal of aiding in the design of optimal strategies for VEGF inhibition. This is purely a numerical model which represents the tissue as a collection of concentrations. Similarly, Ledzewicz et al [17] used a numerical method to solve a mathematical description of cancer treatment, solving for a maximum reduction in tumor volume, to provide dosage sizes of anti-angiogenic drugs. Benzekry et al [19] used a similar method to model the effectiveness of combining anti-angiogenic therapies with cytotoxic therapies. This model focuses primarily on the effect of the delay between treatment types, rather than the dose given, and is concerned with optimizing this. It explicitly states, however ,that this is a model in which parameters would need to be adjusted to a particular patient.

Also of note is the model developed by Owen et al [5], a two-dimensional grid-based model for simulation of combining chemotherapy with a macrophage-based gene therapy under influence of a magnetic field, alongside a vessel and tumor growth system. While not an antiangiogenic model, it uses a similar approach to generate and simulate the vascular growth and drug movement, primarily through expression of VEGF. The model is based on a regular grid, across which drug and oxygen concentrations are stored for each cell in the grid.

1.1.4 Angiogenesis in neurodegenerative diseases

Our research interest in the microvasculature was brought upon by a new direction in research of Alzheimer's Disease (AD). This is based on a new theory on the aetiology of AD - the neurovascular pathway to neurodegeneration. Amyloid beta (ABeta) deposition in the walls of cerebral blood vessels had generally been attributed to the cause and development of AD. A growing body of evidence suggests that there is a strong association between the structure of the cerebral vasculature and AD, especially the microvasculature/capillaries that are responsible for the exchange of nutrients across the blood-brain barrier. This raises the possibility that the abnormalities in the structure may be a decisive early indicator of dementia. There is also evidence that vascular changes occur early in the course of the disease, and may even precede ABeta deposition. Studies on the brains of mice show that before ABeta and other symptoms of AD appear, these mice already have altered cerebral vasculature. In fact, abnormal (e.g. holed, kinked or distorted) microvascular networks in AD brains at the arteriole, capillary, and venule level were reported a long time ago and there was no difference between relatively large arteries in AD and normal brains, but it is only in recent years that the neurovascular factor has become the focus of research into AD. Similarly, the neurovascular factor in other neurological diseases such as Multiple Sclerosis (MS) have been receiving much interest in recent years as many MS patients have been found to have vascular abnormalities. Thus the cerebral vasculature may be a target for diagnosis and treatment of these neurological diseases.

1.2 Angiogenesis in tissue engineering

In their 2011 paper [7], Novosel et al. describe the challenge of vascularization in tissue engineering as the key limitation in engineering large tissue structures. *In vivo*, tissues are supplied with nutrients via naturally formed vascular systems, but *in vitro* tissues are generally submerged in a nutrient fluid. Typically, the maximum distance between capillary vessels in vivo is 200μm, which correlates with the diffusion limit of oxygen, however some tissues such as skin and cartilage are able to perfuse from a greater distance without dying, and hence these tissues are typically the result of tissue engineering efforts. There are two main approaches to vascularizing tissues: cell-based strategies and scaffold-based strategies. The former is based on the ability of endothelial cells to form new vessels, the latter on the actual vessels themselves.

1.2.1 Cell-based vascularization

Cell-based strategies for tissue vascularization rely on the twin processes of neoangiogenesis and vasculogenesis. Within this particular subset of methods are two main strategies of encouraging the growth of functional vasculature, prevascularization and induction of neoangiogenesis. The former deals with growing a vascular network within the engineered tissue before implantation. The latter involves using chemical growth factors to encourage neoangiogenesis once the tissue has been placed in situ.

1.2.2 Scaffold-based strategies

While still limited in its clinical applications, building artificial vasculature is an area of continuing research. Scaffolds can be either biological in nature, derived from decellularized mammalian sections, or artificial in nature, using directed design of vasculature-like structures. Decellularized sections reveal natural 3D vascular structures; the matrix can then be seeded with human cells, introducing the possibility of building perfusable constructs. In artificial approaches, scaffolds are typically produced from hydrogels, made of either biopolymers or synthetic polymers [7]. Even more recently, modified three-dimensional printing techniques have been utilized to create a framework of a sugar-based substance which, when hardened, is submerged in a matrix substance and seeded with cells. The framework is dissolved, revealing a network of tubular structures which can be used as a vascular system with which to perfuse an engineered tissue structure [20]. While initially limited to a cuboid structure, this raises interesting implications regarding the applicability of designing efficient vascular structures for arbitrary three-dimensional volumes.

In their 2009 paper, Lafayette et al [21] described their method of designing vascular scaffolds through reconstruction of micro-CT data. This process involves perfusing the vasculature with a resin to produce a cast, scanning the cast at high resolution, reconstructing and segmenting the images acquired, and then correcting errors manually with a CAD software program. The advantage to this method is that it provides a naturally formed, functional vascular structure, which can be re-used since essentially a "pattern" is created. The disadvantage to this method is that the structure obtained is designed for a singular purpose, and of a fixed

shape. Therefore, a more flexible method of scaffold design would be desirable.

1.3 Challenges of vascular modelling

In silico simulations of biological processes present complex challenges to researchers. First and foremost is the complexity of the biological process being simulated. Even simple processes within the body are comprised of many constituent mechanisms, each subtly tuned to each other. Identifying these mechanisms requires an in depth knowledge of the body on a microscopic level.

Because of the complexity inherent in such systems, the need for computing resources is very high. It is of paramount importance that a simulation of a biological system can run in reasonable time. This is especially important when developing a model, since parameters may need to be modified repeatedly in order to adjust the model to accurately reflect real biological processes. While the term *reasonable time* can mean different things in different situations, in context of the possible need to repeat the simulation multiple times, *reasonable time* is considered here to be anything between ten minutes and two hours.

Typically, simulations of biological processes have been limited to a two-dimensional domain. These simulations are often based on observations of *in vitro* experiments, where cell movements are sometimes constrained to two dimensions, such as cell migration within a petri dish, and often in a uniform environment. Such a simulation is not an appropriate indicator of biological processes *in vivo*, where the domain is three-dimensional and variable.

This research has been performed in a three-dimensional domain. Such an increase in dimensions has had the effect of causing an exponential increase in

complexity. Consider the immediate neighborhood of a single cell in a regular grid: in two dimensions, this neighborhood is eight other cells ($3^2 - 1$); in three dimensions, however, this is increased to twenty-six ($3^3 - 1$), more than three times as many. Applying this increase in complexity to each element causes simulation resource requirements to increase by an order of magnitude.

The nature of a biological system can be described as a collection of individual agents with differing roles, the emergent behaviour being that which we observe as life. This work focuses on one particular system, the growth of microvasculature, and thus the agents in question are endothelial cells. Around this are several physical and molecular phenomena, such as blood flow and respiration, reactions or sub-systems which all contribute to the overall function of the system. The relevance of this is the applicability of GPGPU (General Purpose Graphics Processing Unit) hardware to such a system, as the parallel nature of computation on a GPGPU device is ideal for simulations of complex systems. Having the GPU as a candidate for such simulations gives a possible method for spreading the load across all available hardware resources.

An alternative mode of parallel computing is that of grid computing, and bears mentioning at this point. Grid computing involves sharing the resources of multiple distinct computers, sometimes distributed geographically, communicating via a network. This gives the immediate benefit of being very easily scalable, with the potential of a huge amount of processing power being dedicated to a single task. However, this comes with the increased overhead of data transfer between machines, and increased cost involved in both the initial investment and running costs. A GPU solution mitigates both the cost and data transfer issues, while sacrificing overall power. It is important to note, as well, that grid computing

is typically for computationally more complex tasks utilizing the CPU, although more recently GPU grids have been used [22]. An attractive middle-ground could be found in the ability to combine multiple graphics cards in a single machine, through the use of such technology as NVidia's SLI bridge [23].

1.4 Thesis roadmap

The rest of the thesis is organised as follows. Chapter 2 provides background to the thesis, introducing the key concepts in computerised modelling of angiogensis, and simulating blood flow. Chapter 3 introduces cloud computing and medical imaging, in particular an application of the former using the latter. Chapter 4 describes in detail the angiogenesis model developed during the course of this work. Chapter 5 describes the blood flow and diffusion simulation work of the thesis; Chapter 6 details how these were implemented using parallel processing on the GPU. Chapter 7 explains how individual vasculature from microCT images are integrated with the modelling and simulation frameworks to form a personalised simulation system. Chapter 8 comes back to cloud computing in the context of a modular analysis framework, work completed in the first year of the project, and how the model described in previous chapters can be implemented as a unique analysis module. Conclusions and future work are given in Chapter 9.1.

Chapter 2

Literature review

2.1 Modelling angiogenesis

In the past decades, considerable effort has been made in the development of mathematical models of angiogenesis [18, 24, 25]. Mathematical models rely on the phenomenon of hypoxic regions of tissue producing growth factors, which then diffuse across the tissue towards existing vessels causing endothelial cell migration and the creation of de novo vasculature. This is often combined with hydrodynamic models to assess the effectiveness of the vasculature, and direct remodelling efforts. These models rely on a partially stochastic process, which can produce inefficient or unrealistic results, and are generally bound to a discrete grid resulting in sharp corners. While such attempts to 'grow' simulated vascular networks *in silico* have been the most popular approach, some efforts have been made to generate vasculature following tree-based or fractal methods. This has, however, had limited success.

2.1.1 Anderson and Chaplain's model

Anderson and Chaplain [3] developed a model based on motion of migrating endothelial tip cells governed by three functions; the chemotactic function, or tendency to climb a growth factor gradient; the haptotactic function, relating to the requirement of fibronectin; and cell motion.

Motion of migrating endothelial tip cells is defined as a partial differential equation with three key components as follows:

$$\frac{\partial n}{\partial t} = D_{\mathrm{n}}\nabla^2 n - \chi\nabla\cdot(n\nabla c) - \rho\nabla\cdot(n\nabla f) \tag{2.1}$$

where χ represents a function of the angiogenic factor (VEGF-A) density c, ρ represents a function of fibronectin density f, and D_n represents a random-walk component, based on endothelial cell density n .

In this system, the function *chi* represents uptake and binding of the angiogenic factor, and is described as follows:

$$\frac{\partial c}{\partial t} = -\lambda n c \tag{2.2}$$

where λ is a positive constant. Finally, uptake and binding/degradation of fibronectin, the function rho, is modelled as:

$$\frac{\partial f}{\partial t} = \omega n - \mu n f \tag{2.3}$$

where ω and μ are both positive constants [3].

This system of equations was then solved using finite difference methods to provide a discretization of the model which could be applied to a regular grid de-

scribing a unit square (i.e. a numerical domain of [0, 1] $\times$ [0, 1]) representing an area of width 2mm. The square was divided into a grid of 200 x 200 cells, offering a resolution of 0.01mm per unit. This system was expanded to include random branching events and looping due to fusion of cells which met (anastomosis). Simulation efforts were shown to give results visually comparable to those of in vivo experiments.

This led the way for further improvements and modifications of the scheme and blood flow modelling using fluid dynamics. Arakelyan et al [4] developed a numerical model for tumor vasculature which was comprised of a set of formulae describing each of the interactions in their system. The results were in the form of numerical values of measured quantities including growth factor density, tumour size, effective vascular density, concentration of pericytes, and mature vessel density. Their model suggested that the joint administration of two agents, an anti-angiogenic agent and an anti-maturation agent, provided prolonged suppression of tumour growth and a decrease in average tumour size, irrespective of initial conditions.

Preziosi and Astanin showed a model of vasculogenesis and angiogenesis which created realistic models of vascular networks, comparable to real networks [26]. Linninger and Vaicaitis showed examples of artificially generated cerebral vascular trees based on actual patient data which show similarity to the real images [27]. The model incorporates natural flow laws to generate trees with high congruence to actual vasculature in terms of appearance and statistical properties. Moore and David developed a method for modelling the autoregulation function of the cerebral vasculature in such a way as to simulate the effects of stenosis and occlusion [28]. Perfahl et al developed a multiscale model of angiogenesis in tumours in three

dimensions, which was shown to be able to predict the spatio-temporal evolution of vascular tumours [29].

Maturation of vessels is an important factor in the growth of blood vessels which relates to the formation and regression of mature vessels, (that is, vessels which have stabilized). Previous models have focused mainly on the formation of immature vessels (here categorized as angiogenesis). The model developed by Arakelyan et al [4] includes the concept of vessel maturation, however the algorithm is described numerically in terms of more abstract concepts such as effective vascular volume and concentration of pericytes. As such, its application is less direct in terms of predicting vascular growth initiated by tumor presence.

Godde and Kurtz [30] developed a model which works on a hexagonal grid and considers shear-stress and pressure dependent systems. Their model showed that shear-stress dependent modelling led to a more homogeneous distribution of capillaries connecting terminal arterial and venous vessels than that of pressure dependent modelling. Owen et al. [31] developed a multi-scale model of angiogenesis and vascular remodelling which accounts for vessel pruning due to low wall shear stress amongst other factors.

A model of VEGF distribution and interactions in tissue was developed by Stefanini et al [18] which was designed to aid in determining optimal strategies for VEGF inhibition. The model describes kinetic ligand-receptor interactions between VEGF isoforms ($VEGF_{121}$, $VEGF_{165}$) and ECM binding sites for $VEGF_{165}$. The model does not explicitly describe tissue geometry, but rather represents the components of the tissue (capillaries, ECs, basement membranes) as numerical values of concentrations of molecular species.

Watson et al. [25] developed a model of vascular development in the murine

retinal plexus during neonatal development, involving modelling development of the astrocyte network which precedes vasculogenesis in neonatal mice. This approach employs similar modelling techniques to those outlined by Anderson and Chaplain in [3] by tracking individual astrocyte and endothelial tip cells. However, this model adds increased complexity through remodelling of vasculature. This involves simulation of blood flow through the network, and delivery of oxygen to surrounding tissue as a result, followed by pruning of the capillary network in light of relevant criteria. Simulation is started at embryonic day fifteen (E15), with an initial seeding of astrocyte cells around the location of the optic nerve. These cells grow outward, encouraged by an existing gradient of platelet-derived growth factor A (PDGF-A), a factor implicated in the development of astrocyte networks. Hypoxic astrocytes are modeled as producing VEGF-A; thus, as the astrocyte network grows, a gradient of VEGF-A is created across the retina. At the day of birth (E21.5) endothelial cells are seeded around the optic nerve and begin to migrate radially outward, motivated by the recently created VEGF-A gradient.

2.1.2 Other modelling approaches

Cellular Automata

An alternative to the typically developed models is that of a cellular automaton model for angiogenic processes. In a cellular automaton a region is, typically, divided into a regular grid, each of which constitutes one cell in the automaton. Each cell is assigned a state, and the state of each cell is derived iteratively from itself and its neighbors. Rules are established to determine whether a cell changes state after each iteration. Possibly the most well known cellular automaton is

Conway's Game of Life [32], in which a cell can be either alive or dead. Evolution of the system is described by three rules; if a cell has fewer than two living neighbors, or more than three living neighbours, it dies; if a cell has two or three living neighbors, it survives; if a dead cell has three living neighbours, it is brought to life [33]. Through this simple model, immensely complex systems have been designed, and it continues to be a subject of study today.

Topa [34] demonstrated a cellular automaton model developed to simulate tumor induced angiogenesis, emulating the results of Anderson and Chaplain's [3] model. The results, whilst comparable to those of Anderson and Chaplain, are in no way an improvement to the model, and thus considerable work would be required to improve this model to the point where it could be of any use. The model described is, in fact, a hybrid model based upon the concept of a cellular automaton; the automaton model is used to simulate the tissue and distribution of nutrients and other chemicals throughout, with the vessel network being a graph-based representation which sits atop the tissue automaton.

Tree-based networks

Comparisons between human vasculature and tree structures are often made. In fact, one of the processes crucial to formation of a vascular network in a developing embryo is known as intussusceptive arborization [35], that is formation of a branching structure (*arbor* meaning tree, a word whose origin is, curiously, unknown). Thus, it is reasonable to assume that tree systems could be used to generate visually similar structures to *in vivo* observations of human vasculature. The use of tree structures is attractive, as they appear to be a more efficient way of regularly filling a space than a vessel growth algorithm. Tree generation algorithms have

problems of their own, however. For one, they do not form anastamoses (looping structures), an essential characteristic of functional vasculature.

Space-filling trees are deserving of consideration when regarding the idea of filling an arbitrary space with vasculature [36]. Space-filling trees can be regular or random, and involve subdividing a space and branching into the new subdivisions from the nearest node. Regular trees fill the space in a symmetrical fashion, which leads to the property of having a short maximum distance from one node to the base node. Rapidly-exploring Random Trees are created from an already fully subdivided space. From a base node, a random point in the space is chosen and connected to the nearest node. This process is continued until the space is filled [37]. One problem with this initial model of formation is that the tree has a tendency to cross itself (see figure 2.1), which would result in low-efficiency of vasculature. Similarly, it does little in the way of emulating real vascular structures. In order to improve this, some considerations can be taken into account.

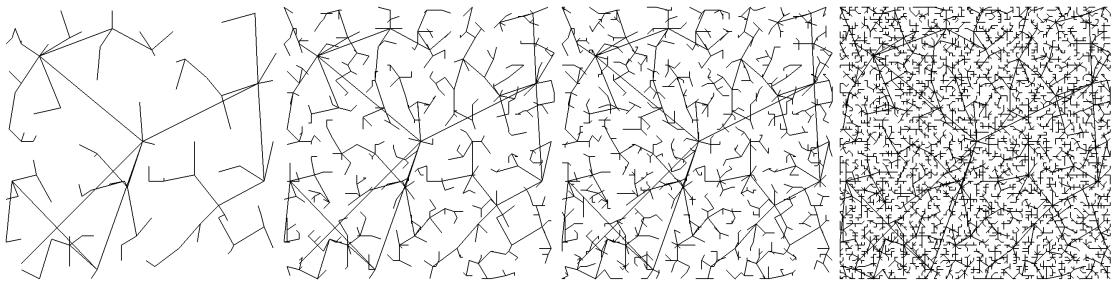


Figure 2.1: A Rapidly-exploring Random Tree at 100, 200, 500 and 1000 iterations.

Initially, the decision-making process behind choosing a new node to branch into can be modified to use only a region close to existing nodes. This effectively limits the size of each branch to a pre-defined maximum length. With appropriate conditions, this should effectively remove the possibility of self-crossing branches.

Second, the system can be modified to take into account a map of perfused areas, limiting the tree to branching only into areas of low oxygen content. Thus the process becomes more akin to biological growth in that it is driven by biological factors.

In considering tissue scaffold development, the idea of a space-filling tree is an appealing one. Possibly the largest barrier to engineering large tissue structures is the diffusion capability of oxygen. This diffusion characteristic limits the distance between vascular segments to a maximum of 200μm (0.2mm) [7], thus limiting the size of an engineered tissue quite severely without an effective vascular system in place. A space-filling tree appeals through its ability to fill an arbitrary space up to a desired threshold, for instance minimizing the distance between adjacent branch elements.

Unlike other models of angiogenesis, this approach does not model movements of individual cells, but instead attempts to imitate the formation of a vascular network by the assessment of various factors, such as oxygen diffusion level. Rapidly-exploring random trees were initially developed as a path planning tool, thus development of an RRT model of scaffold design could be considered a chemodynamic (that is, led by chemical factors) path-planning problem of sorts. In kinodynamic RRT development, a single location is given as a goal, and once a path to that goal is acquired the tree is considered complete [37]. In order to use a tree generation system for production of vascular scaffolds, the 'goal' would be two-fold: a) the space must be filled such that all vascular tree branches are a maximum of 200μm from any other branch, b) the the structure must produce a flow throughout the scaffold, presumably bridging the gap between opposing sides of the structure.

Like other trees, however, rapidly-exploring random trees do not taking looping

and blood flow into account, and thus are not a complete substitution for a vascular tree. Bearing this in mind, a possible solution would be to grow two opposing trees from opposite ends of the domain, and allow them to join where interactions occur. The advantage of this approach is the speed with which it could be generated since the decision making process is on a per branch basis, not per cell.

2.2 Simulating blood flow

As long ago as 1830 the movement blood in microvessels was of interest to scientists. The French physicist and physiologist Poiseuille took it upon himself to investigate the problem of resistance to blood flow posed by the narrowest parts of the vascular system. His work on the flow of liquids such as water and blood through narrow tubes led to him describing what is now known as Poiseuille's law (Eq 6.3). He noted that pressure across a tube was directly proportional to the length and radius of the tube, and the viscosity and flow-rate of the liquid within, and more specifically that it related to the *fourth* power of the radius.

For the next century, or so, Poisueille's law was generally considered the gospel for blood vessels. In 1954 the Microcirculatory Society was formed with the notion, in part, of promoting research into the form and function of microcirculation in health and disease. At this time, relatively little was known of the rheological behaviour of blood within microvessels [38], until the pioneering work of Fåhræus and Lidqvist discovered the curious effect of reducing capillary diameter on the relative viscosity of blood. Blood is a concentrated suspension of red blood cells in plasma [39], typically comprised of approximately 40-45% red blood cells (erythrocytes). A single red blood cell is, normally, a biconcave disc with a diameter

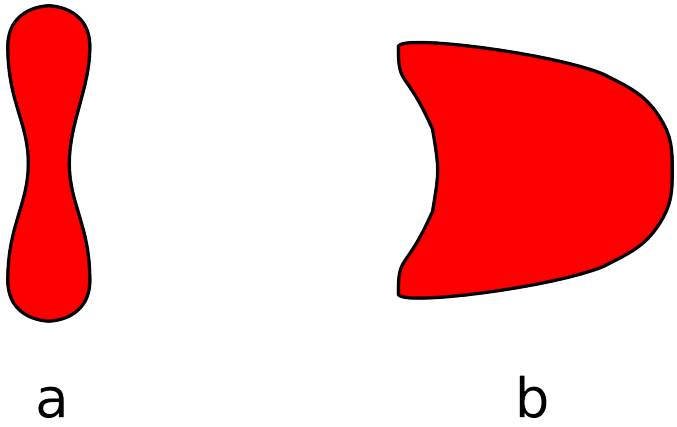


Figure 2.2: Red blood cells traveling in larger vessels are able to maintain their ordinary shape (a), however when passing through vessels smaller than the normal size of a single red blood cell, they are stressed in order to fit, forming a shape more akin to that of a bullet (b).

of 8 μm and a thickness of 2μm. During flow through the microvasculature, this is stressed to form a more bullet-like shape [39] by the smaller diameter of the capillaries themselves (Figure 2.2).

Fåhræus and Lidqvist reasoned that blood would not necessarily act like an ordinary fluid when passing through the smallest of human blood vessels, and thus began to investigate the effect of reduced vessel width on the viscosity of blood. In their experiments they were able to simulate blood flow through tubes of much narrower diameter than Poiseuille ever was, with values as low as 0.05mm. They discovered that at these decreasing widths, below approximate 0.3mm, the viscosity of blood appeared to decrease [40]. This phenomenon was termed the Farhæus-Lindqvist effect. Unfortunately, like Poiseuille, Farhæus and Lindqvist were limited by their equipment, and were unable to investigate anything smaller. If they had, they would have discovered something even more strange.

Pries et al. took this research further, and studied *in vivo* the effect of microvessels in the rat mesentery (a membrane which attaches many internal organs to the

abdominal wall), the results of which were observed through intravital microscopy. Pries et al. discovered that at radii of below approximately 20μm (0.02mm), there was a sharp increase in the apparent viscosity of blood moving through them. From their experimental data they were able to develop an equation, which they termed the "in vivo viscosity law" [41], describing the relative apparent viscosity of blood within these vessels.

More recently, computational fluid dynamics have evolved to be able to simulate complex flow patterns in non-uniform environments. Lattice Boltzmann (LB) methods have been used by many as an accurate method for simulating blood flow in vessels reconstructed from medical images. Due to the nature of the method, LB is able to accurately simulate fluid movement on a small scale, resulting in accurate representation of the flow of blood for such applications.

Lattice Boltzmann methods work by subdividing the simulation volume into a regular, axially aligned grid of nodes, each representing an area in the volume. Each node is affected by all of its neighbors, and similarly has an effect on each of its neighbors. In this way, the Lattice Boltzmann is a discretisation of the Boltzmann equation in three dimensions. When simulating flow in a vascular network, the grid created for a LB simulation is created by filling the convex space created by the shape of the vascular network. The resultant grid is updated by calculating the effect that each element has on each neighboring element, and the state that leaves the element in.

The advantage of this method is that it accurately simulates the flow of liquid within volumes of an arbitrary shape, and takes the shape into account when performing the simulation. However, microvasculature requires special concerns when being simulated, since the nature of blood on such a small scale is not that

of a uniform fluid. When considering blood on a macroscopic scale, it is easy to consider it as a uniform fluid, but in reality it consists of several constituent parts, including red blood cells (erythroctyes), white blood cells, and plasma. In humans, blood has a typical haematocrit, that is the percentage of red blood cells, of between forty and forty-five percent. So let us consider the effect this has on the movement of blood within the microvasculature.

Red blood cells are typically 8μm across, whereas the microvessels can be as small as 5μm in diameter. The result of this is that red blood cells are deformed as they travel through the vessels. With this consideration in mind, let us now consider the Lattice Boltzmann method of fluid simulation for such vessels. When traveling through capillaries red blood cells are deformed, but also lubricated by a thin layer of plasma on the vessel wall. This, then, is already a non-uniform fluid that is being considered. We must also consider that the elements of the grid which represent blood cells and those which represent plasma are undifferentiated, even though they have different physical properties. For example, the elements which represent a part of a blood cell should be physically connected to other elements representing another part of the same cell. As such, LB methods are unable to simulate the movement and deformation of red blood cells within capillaries. The assumption of uniformity on such a small scale combined with the high level of computational complexity suggests that a Lattice Boltzmann method would be a poor choice for simulations of blood flow on this scale.

Chapter 3

Cloud medical imaging and visualisation

The work detailed in this section was completed in the first year of study, in collaboration with Biotronics3D on their cloud-based medical imaging system 3DNet Medical. The work centers around the development of a medical imaging analysis platform, developed within a proprietary cloud-based 3D medical imaging solution. While focused primarily on the creation of the modular framework within the cloud system, it initially involved some work on volume rendering with the intention of providing a fully-featured SDK based on the core of the 3DNet rendering software. The result was a method for rendering depth values to an image based on the raycasting volume rending algorithm. The aim of the industrial project was initially categorized in two parts as follows:

1. To create a flexible modular framework for medical image analysis within a cloud environment.

2. To prove the framework by creating an analysis module with clinical relevance which can be integrated through use of the framework.

This chapter introduces concepts in cloud computing, medical imaging and volume rendering, as a basis for the first part. The second is covered later, in chapter 8.

3.1 Cloud computing

The concept of cloud computing has become commonplace in modern computer science. Ubiquitous are the adverts which claim that data is safer "in the cloud", or that business processes can be much improved by moving them "to the cloud". Of course, such a broad definition of cloud computing does little to demystify the term for the average user, and obscures the truth of the underlying technology involved.

In its simplest form, cloud computing can be described as moving functionality away from desktop computers and onto remote machines. The essence of a cloud is to have a collection of machines running as a cluster, offering an on-demand service which users are able to interact with from their personal computers via an internet connection. A useful example of this technology would be cloud-based storage solutions, which allow users to create documents and take photos, among other actions, and store them on a remote cloud. These become available for retrieval later on any of a number of devices tied to a particular user account. Services are typically made available through native applications on the operating system, as well as through a web interface. To the end user this seamless interaction with one or a number of other computers results in the appearance of functionality beyond

the scope of which their device is physically capable.

This ability to extend the use of a device beyond its capability is the primary motivation behind the development of medical imaging applications which use cloud computing. Typically, medical imaging applications require a high powered computer to be able to run, and as such having many capable machines is often prohibitively expensive. Being able to perform highly complex operations within a cloud computing system is a useful advancement; the provision of such services is usually subscription-based, under what is called the Software As A Service (SAAS) model. SAAS moves away from software being a commodity, where each user would own a copy, to a service provided by a company under a pay-to-use model, be it on a per-use basis, or on a regular billing (subscription) system.

Volume rendering is the method in which projections of 3D volumes are displayed as 2D images. Early implementations of volume rendering techniques focused on rendering of texture data, initially as a set of blended 2D textures and later, as the hardware permitted, utilising 3D textures, before ray casting was implemented. In medical imaging, a data set is comprised of a series of images which are the result of a patient scan. These images are of either cross-sectional slices of the patient's entire body, or a sub-section thereof. Traditionally, these sets of data are viewed as single images, often as a set of consecutive slices. However, a single study can comprise of as many as thousand images, resulting in more than 1.5 gigabytes of data. Because of this difficulties can arise when processing scans on a slice by slice basis. The use of volumes is therefore desirable. By assigning a thickness to each slice in a scan (which can be determined by the interval at which the patient was scanned) these slices can be composited to a volume representation of the patient's body. This volume can then be used to produce a visualisation of the

patient with volume rendering. Direct Volume Rendering (DVR) generates images without the need to create an intermediate polygonal representation of a volume. Instead, the volume data set is projected onto an image plane. In the image-space oriented ray casting approaches, rays are cast from the view-point through the view-plane into the volume. The volume is equidistantly sampled along the ray and the volume integral is computed by repeated application of the over operator in front-to-back order. Volume rendering offers a number of challenges, especially when scaling this to a large multi-user solution such as a cloud. Memory and processing power are an important consideration. In a cloud environment each user needs access to enough memory to ensure that the system continues to run smoothly. A typical study can contain multiple series, of which an average of three hundred images per series is common. This results in each user requiring, on average, four gigabytes of memory in order to work. Loading of large data sets takes time, an issue which needs to be addressed in a cloud system. While processing power is less of a problem, it is still important to ensure that every user has enough power to perform the tasks they require. In addition, rendering of large data sets can take time without an appropriate acceleration structure.

Pure hardware based volume rendering solutions provide real-time performance and high quality. Consequently they are the most applied approach in practice. However, pure hardware volume rendering solutions are are limited in their their functionality as basic visualization systems are supported by hardware volume rendering solutions. Advanced visualization systems provide pre-processing features such as filtering, segmentation, and morphological operations, among others. If such operations are not supported by the hardware, they have to be performed on the CPU and data must be then be transferred back to the hardware. This

transfer is very time consuming, thus, interactive feedback becomes problematic. In contrast, within a pure CPU based solution this transfer is unnecessary allowing more efficient processing of data. Furthermore, in the framework of a cloud environment dedicated hardware-based solutions become prohibitively expensive: setup cost, maintenance, and even scalability become limited due to hardware constraints. Thus, a pure CPU based solution is by far the most suitable, and probably the only truly viable solution, for cloud based rendering.

To accelerate CPU based rendering and image processing, the underlying memory management has to be modified. In this case we utilise a bricked memory layout. Cross-sectional data, e.g. CT and MRI, are large sets of individual images which combined form a volume in space. Physical memory is typically constructed in a sequential way, therefore the straightforward approach to loading these images into memory is to put them one after the other using a linear layout. This layout has several disadvantages. In a typical set of cross-sectional images, an average 30 percent of the data actually does not contain any useful information. This comes from the fact that the human body consists of a set of tubular structures (e.g. arms, legs, and torso). A cross-sectional cut through a tubular structure using rectangular images does not contain the data well, leaving vast amounts of data to represent empty space around the body. Furthermore, in the case of advanced medical imaging application data needs to be processed in a non-sequential, random way. Volume ray casting has a strong view-dependent data access pattern, and consequently, taking a look at the typical cache hierarchy of today's CPU (L1, L2, L3) it becomes clear that storing images linearly in memory would cause complete cache thrashing. In order to address the aforementioned issues a significant improvement is gained if the cross-sectional data is arranged in a blocked manner.

In this case we subdivide and reorganize the entire volume (one 3D-image) into smaller contiguous lightweight bricks, obtaining a structure analogous to a Rubik's cube.

One of the main problems faced when sharing hardware and software resources between multiple users in an arbitrary manner is the robust and efficient administration of the hardware available. Not only must each user have secured data storage and privacy protection, but it must also be able to exploit its resources without having to directly control how the underlying hardware and software resources are being utilized. Cloud systems are an example of a technology that requires a managing entity that "virtualizes" the usage of hardware and software in a way that each user has a direct and transparent interaction with the system. The challenge is to build a lightweight instrument that allows for a seamless interaction efficiently. A dual strategy that not only permits automatic virtualization of the resources but also specializes in distributing them in a coherent manner by performing a "load balancing" of the tasks on the available resources is required.

Virtualization, as described here, is instrumental to managing secure user sessions and is fundamental to the efficient distributed rendering required to perform advanced imaging applications. In particular, virtualization is achieved by creating "sandboxes", a concept that provides restricted resource sets to individual users including controlled access to data storage, hardware resources and networking privileges. This creates a local, virtual machine for each user and removes the burden of requiring them to manage how their tasks are processed by the system.

In order to achieve this securely and efficiently, the cloud system is split into the following sections: a Global Session Manager (GSM) responsible for managing user specific session sandboxes, and a View Session Manager (VSM) responsible

for managing viewing session sandboxes and load-balancing. The load-balancing itself is done by the Rendering Resource Load Balancer (RRLB), which is part of the VSM. Both the GSM and VSM are deployed as web services and can be mirrored for redundancy.

The scalability of the cloud is an important feature, since it inherently implies a cost effective solution. At any time additional nodes can be added to the cloud to make it more powerful and the cost per user is much reduced compared to that of buying individual workstations. Users can be classified as one of three types; casual users, who have a low level of activity, active users who have a higher level of activity, and power users who are using the computationally expensive features of the system. Whilst a power user may be choosing transformations, transfer functions, and adjusting settings, casual users could be simply viewing an image already rendered to the screen. Thus, while a 32-core machine with 64 users would imply less than a single core per user, in reality this is not the case. It is, in fact, memory that is the limiting factor, since even those who are not interacting with the system will use large amounts of memory.

3.2 Medical imaging

Medical imaging covers a broad range of methodologies such as X-Ray imaging, Magnetic Resonance Imagining (MRI), Positron Emission Tomography (PET), Computed Axial Tomography (CAT), and many more. It also covers software such as Patient Archiving and Communication Systems (PACS), and the Digital Imaging and Communications In Medicine (DICOM) standard [42].

DICOM is the industry standard for the storage and interchange of medical

images and was put together by the American College of Radiology (ACR) and the National Electronic Manufacturers Association (NEMA) of America, originally under the guise of ACR-NEMA. The first definitive version was published in 1985 as ACR-NEMA Standards Publication No. 300-1985, designated version 1.0. This was followed by two revisions, first in 1986, then in 1988, before the standard was reworked under the guise of DICOM. The purpose of DICOM is to provide a worldwide standard for the format of medical images and the transfer of such images between devices, in order to provide a high-level of compatibility between devices and systems developed by different companies.

The Insight Segmentation and Registration Toolkit is an open source toolkit for the analysis of multi-dimensional images (2D, 3D, nD) provided by the Insight Software Consortium. ITK is written in C++ with the intention of being cross-platform, and provides wrappers for other languages such as Java, Python and Tcl [43]. The Visualization Toolkit (VTK) was similarly developed to provide common components for developing applications which require the visualization of 3D graphical elements [44].

These have both been expanded upon in the form of the Medical Imaging Interaction Toolkit, developed by Wolf et al, initially in 2005 and continuously to date [45]. The Medical Imaging Interaction Toolkit (MITK) is designed to reduce the effort involved in developing medical imaging applications which use ITK and VTK, and as such provides a set of components which use ITK and VTK but offer further features, such as architecture for easily combining ITK algorithms with VTK visualization components.

3.3 Volume rendering

Volume rendering is a method of creating two-dimensional projections of three-dimensional volumes. Volumes are created by compositing a series of two-dimensional images taken in parallel planes, such as those of an MRI or CT scan. Images are assigned a thickness (in the case of medical scans this is determined by the scan interval) and images are ordered so as to produce a cuboid of voxels (three-dimensional pixels). This is equivalent to the method used to construct simulation volumes described in chapter 7.

Early implementations of volume rendering focused on blending texture data, initially sets of 2D textures and later, as the hardware permitted, 3D textures. These methods were later superseded by the ray casting algorithm [46]. Due to advances in the ray casting algorithm offering greatly increased speed, ray casting is now the standard method for volume rendering in medical imaging applications. Here, algorithms for texture-based rendering and ray casting will be covered briefly before discussing the modification to the ray casting algorithm for depth rendering.

3.3.1 Texture-based volume rendering

Initially, texture-based volume rendering was performed by blending a set of two-dimensional textures. These textures are acquired by taking 'slices' of a volume parallel to the viewing plane. These textures are then layered in back-to-front order when rendered [46]. Composition of successive color values is performed using the 'over' operator, in the following format:

$$C_{final} = C_a\alpha_a + (1 - \alpha_a)C_b\alpha_b$$

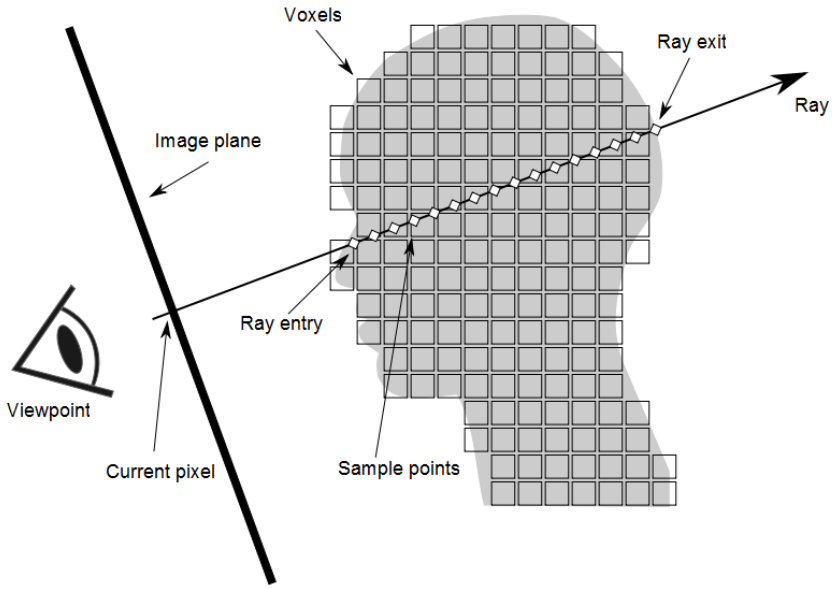


Figure 3.1: Rays are traced from the viewing plane through the volume and sampled equidistantly. Color values are composited until a predefined opacity threshold is reached.

where C_a, α_a, and C_b, α_b are the color and alpha (opacity) values of the fragments to be blended, where b is furthest from the viewing plane.

3.3.2 Ray casting

In ray casting, rays are cast from the view-point through the view-plane into the volume (Figure 3.1). The volume is equidistantly sampled along the ray and the volume integral is computed by repeated accumulation of colors and opacities.

At every sampling position a scalar value is interpolated between the corresponding surrounding eight voxels, that is, in the logical three dimensional extension of a pixel. This value is then classified according to a transfer function which maps density values to a colour function. If the sample is non-transparent,

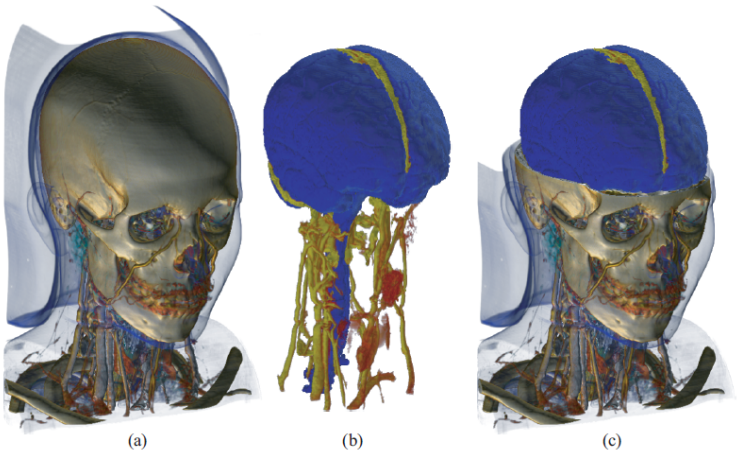


Figure 3.2: Volume renders of the neck and skull using different transfer functions: (a) shows the entire volume; (b) shows a render containing segmentation information where a second transfer function has been applied; (c) displays a combination display, defined by a planar cut-off .

a gradient is computed from the surrounding voxels in order to apply shading. Finally, the sample is composited with the previous samples of the ray. Successive samples along the ray are calculated until a threshold opacity level is reached, or the ray exits the volume. Figure 3.2 shows a typical example of a volume render: (a) shows a render of the full volume; (b) and (c) show renders containing segmentation information where a second transfer function has been applied.

3.3.3 Rendering depth

In medical imaging applications it can sometimes be desirable to know the depth of a certain point in a volume render produced. Examples may include the need to place markers in three-dimensions for alerting colleagues to specific features of interest, or for picking a seed point for performing segmentation operations such as bone removal. To achieve this, the concept of a depth render is created. The depth render allows existing image generation and transfer protocols to be used to

convey another layer of information in respect to a volume being rendered.

The depth render algorithm makes use of the existing ray casting algorithm as a basis, but in this instance does not store color information. The position the ray is cast from (i.e. a point on the viewing plane) is recorded as an initial condition for the depth to be calculated. The ray is then cast as normal, and successive opacity values are accumulated. Once the opacity threshold (defined by the current transfer function) is reached, the end point of the ray is recorded. With these two values and some elementary vector mathematics the length of the ray cast can be calculated, representing the distance of the pixel of interest from the image plane.

Distances computed this way are stored in an image in the form of pixels. This takes advantage of the correlation between storing depth values as 32-bit floating point numbers and the image format consisting of pixels of four channels of 8 bits per channel. The distance value is masked into four bytes, which are stored in the four components of the respective pixel. To retrieve the distance to a particular pixel, the red, green, blue and alpha values can be recombined to form a 32-bit floating point distance value.

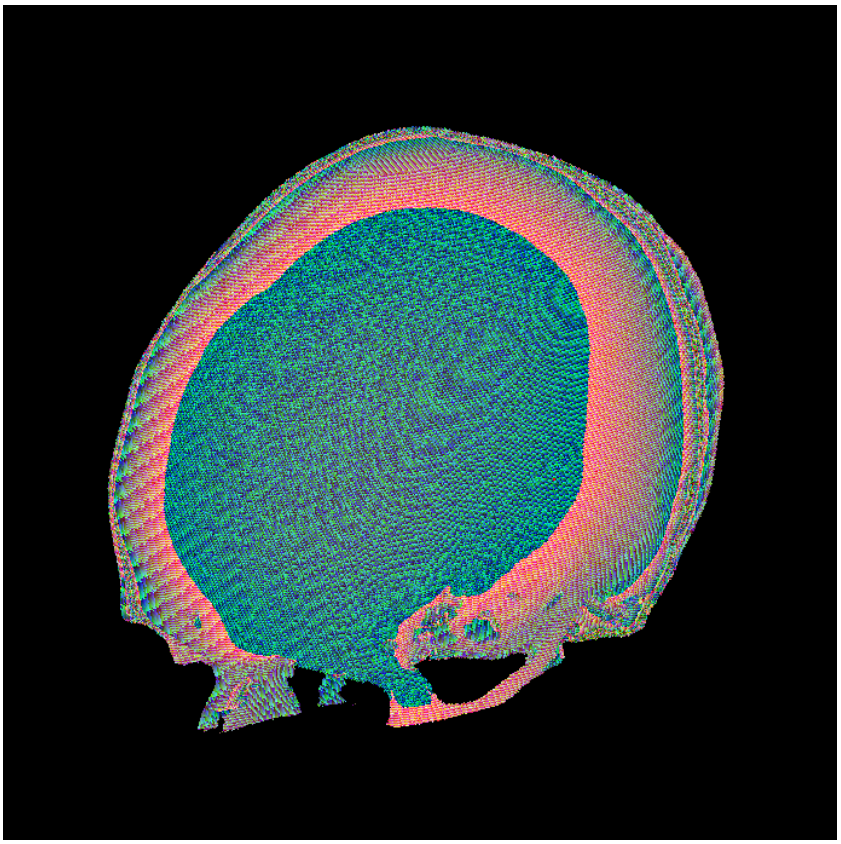


Figure 3.3: The depth render stores 32-bit depth values as colours to make use of the existing imaging technology.

Chapter 4

Modelling angiogenesis in three dimensions

One of the core aims of the work undertaken here was to develop a three-dimensional model of angiogenesis. Compared with the two-dimensional model developed by Anderson and Chaplain [3], the extra dimension introduces computational complexity, which has led to the GPU implementation of the fluid dynamics to provide a mechanism for growth factor diffusion within the bounds of the model, and simulations of blood flow and mass transfer within the vasculature, discussed in detail in chapter 5. As a whole, the method is a combination of a Cellular Potts model (CPM), which models cell movement, and partial differential equations (PDEs), which are used to govern the release and subsequent diffusion of chemical attractants. CPM is a well known lattice-based computational modelling technique which represents unit sections of an arbitrary space as cells on a grid, updating them based on a set of probabilistic rules. Typically, within a CPM an object occupies a single grid location, although they can also occupy multiple grid cells.

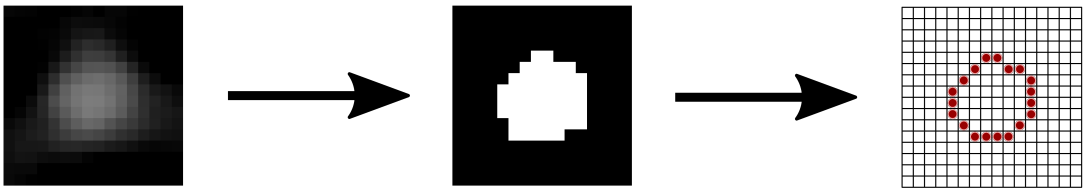


Figure 4.1: Scanned images are thresholded and converted to binary images; these are then used to instantiate endothelial cells making up the initial environment of the simulation.

During the initialization step of the simulation, an environment is set up in a way which reflects a section of human physiology. A stack of images acquired from a microCT scan are imported. The images used are of resin casts of rat cerebral vasculature, taken at a resolution of 15 μm, approximately the size of an individual cell. Images are thresholded, resulting in binary images, and individual pixels are inserted into the domain of the simulation as nascent endothelial cells. Thus, pixels in the scans are likened to cells within the original vasculature, resulting in sections of tubular structures from which the neovascularization arises.

To more accurately reflect the vasculature these cells represent, before the simulation begins the domain is scanned for cells which are entirely enclosed - that is, they have neighbours in both positive and negative directions in all three axes. Any cell found to match these conditions is flagged for deletion before the simulation begins. This also has the added effect of reducing the computational complexity of the system by decreasing the number of agents to simulate. The full methodology for initialising the environment is discussed in detail in chapter 7.

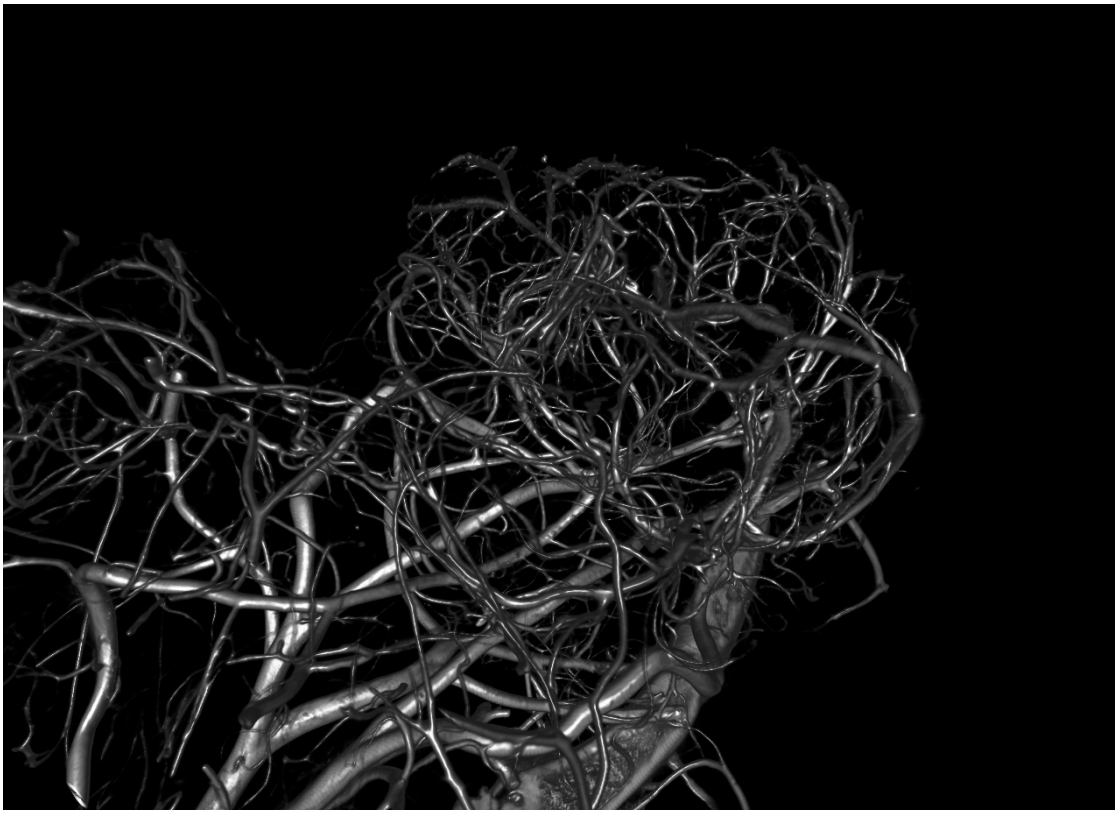


Figure 4.2: MicroCT scans of a resin cast of a rat brain show a high level of detail, and so were an idea candidate as a basis for developing a vascular growth model through the initial elimination and subsequent simulation of growth of microvasculature.

4.1 Endothelial tip cell movement

Typically, models of angiogenesis describe the growth of new vessels from existing vasculature via tracking the path of migrating endothelial cells. The movement of these cells is initiated by disruption of the basement membrane which surrounds the endothelium of existing vessels, thus allowing for free movement of endothelial cells. Movement is then tracked under the assumption of endothelial cells migrating towards the source of the growth factor. The equation for endothelial tip cell motion is defined earlier in section 2.1.1 as equation 2.1.

Movement of endothelial tip cells is derived from a discretized form of equation 2.1. Discretization is achieved by applying the finite difference method [47] which yields three equations, relating to each of the components of equation 2.1. These three equations relate to the three characteristics influencing cell migration; movement up a gradient of growth factor, movement up a gradient of fibronectin, and a random walk component.

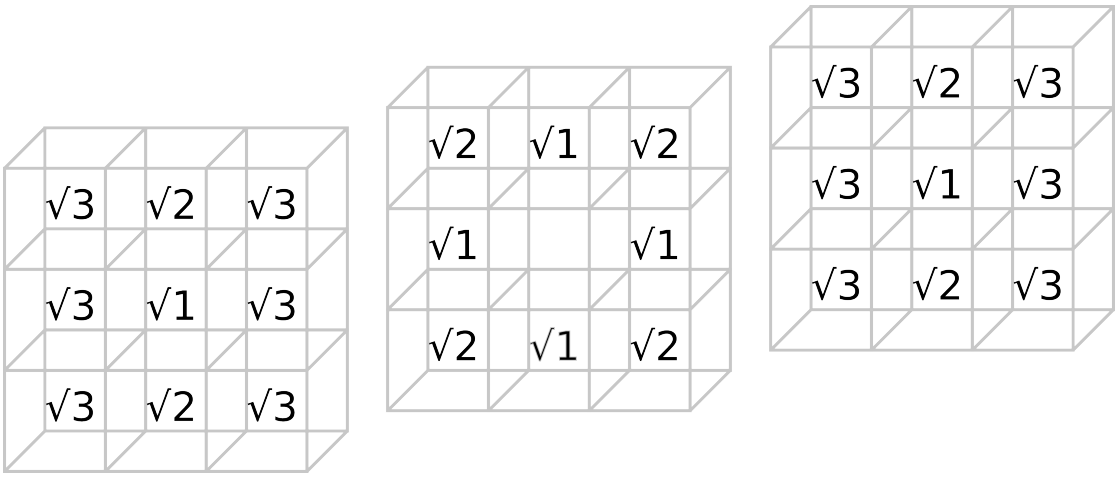


Figure 4.3: When a cell moves, the required time for another move is based on the distance it has travelled, since diagonal moves are further than axially-aligned ones. The multiplier is based on the length of the movement vector.

Endothelial cells are modelled as agents on the CPU. Each agent is modelled individually, and thus is aware only of itself and its immediate environment. As a simplification measure, in terms of program structure, a grid (three-dimensional array) of numerical values is maintained as part of the environment, indicating whether a particular location within the simulated section is occupied by a cell or vessel segment. The numerical value is an identifier of the cell which filled this space - in the case of a vessel segment, the identity of the originating cell is given. These values are used to prevent a cell looping into itself, causing early

termination of the simulation, which sometimes occurs due to the stochastic nature of cell motion.

When a cell moves, the direction in which it can move is constrained to those empty cells which are considered to be directly *in front* of the cell. These cells are identified by taking the dot product of the vector from the cell's current position and the vector of the direction in which the cell last moved. If the result is above a threshold value (with normalised vectors a value of 0.5 is used, implying an angle of less than 90 degrees), the cell is a viable target direction in which to move. Enabling this constraint removes the possibility of ECs moving in a direction which would be physically impossible. Deciding in which direction an endothelial cell moves is performed by distributing probabilities across the set of cells available for movement. This involves finding both the cell in the direction of continued movement and the cell in the direction of the overall chemotactic gradient. The chemotactic gradient is calculated by summing the differences of the growth factor densities across the local environment of the EC:

$$\sum_{\substack{-1<i<1 \\ -1<j<1 \\ -1<k<1}} \mid \chi(i,j,k) - \chi(0,0,0) \mid \times(i,j,k) \tag{4.1}$$

where χ(i, j, k) is the growth factor density at the displacement (i, j, k) from the ECs location. This yields a vector which is normalized to give $\hat{\chi}$, describing the direction of the growth factor gradient at the location of the EC. This vector is used to identify the cell which is in the direction of the chemotactic gradient. The continued movement cell is simply identified by repeating the last displacement.

The two cells identified by the chemical factor gradient and the repetition of the

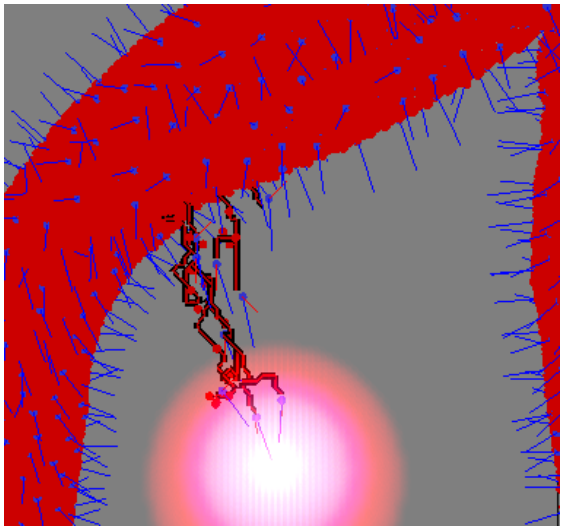


Figure 4.4: Cells move towards the source of VEGF (pink), up a density gradient. Motion is randomised, weighted towards an increase in growth factor density and in the direction previously moved. Activated tip cells are drawn in blue and have an additional visualisation element of a vector describing the direction of the local chemotactic gradient. Red cells are those marked as complete, either through exclusion or anastamosis.

last move are given weighted probabilities as defined by constants χ and ψ. The values for these are adjustable by design, such that different growth patterns can be defined, but through repeated simulations values of 0.5 and 0.25 respectively were found to give the best results.

The remaining cells are assigned an equal portion of 1 - χ - ψ. This represents the random portion of the movement equation, and is more apparent when the values for the chemotactic and haptotactic biases are low (see figure 9.2).

After assigning these probabilities, they are combined to form a distribution which ranges from 0.0 to 1.0. A random number is generated in this range, and the cell in whose range this falls is allocated as the location to which the EC moves.

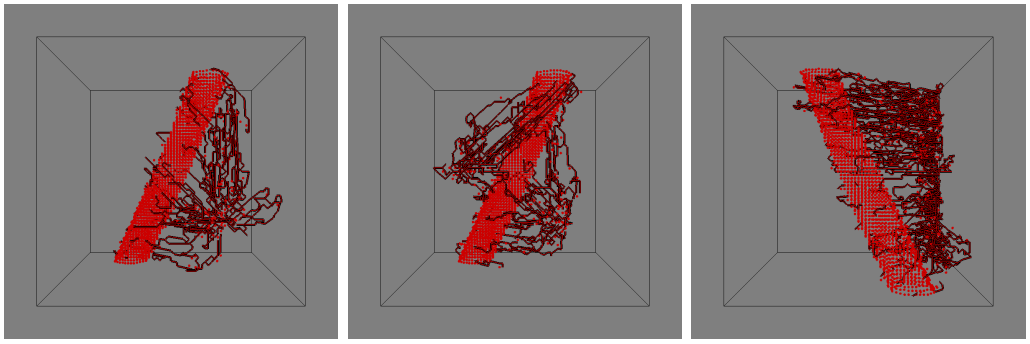


Figure 4.5: Endothelial cells move towards the source of VEGF up the local density gradient. The image on the left shows a simulation run with a single source; the middle image shows the result of defining two sources; the right image contained no sources, but was initialised with a linear gradient from right to left.

4.2 Vessel formation

When the threshold growth factor density, ν, is reached, nascent endothelial cells can become activated, allowing them to migrate away from their parent vessel towards the source of the growth factor. This is the first stage in the angiogenic process, and represents the degradation of the basement membrane caused by the expression of VEGF under hypoxic conditions. When a nascent cell is activated, nearby cells within a fixed distance (a zone of exclusion) are marked as complete. This prevents overcrowding of the area due to too many cells moving towards the source of VEGF. Programmatically, this is achieved by successively stepping away from the activated cell until a threshold level is met, as opposed to a straight vector length calculation which has the possible complication of affecting cells on the opposite side of the parent vessel. This distance can be modified in order to increase or decrease the density of vessels grown, representing a pathological factor in some diseases, such as Alzheimer's (figure 4.6).

Cell movement is only allowed when the cell has reached the threshold time for

cell doubling to occur, to represent the need to fill in the space behind the moving cell. On a cellular level, capillaries are formed from single cells forming tubular sections, chained together to form vessels, and this constraint is designed to reflect that.

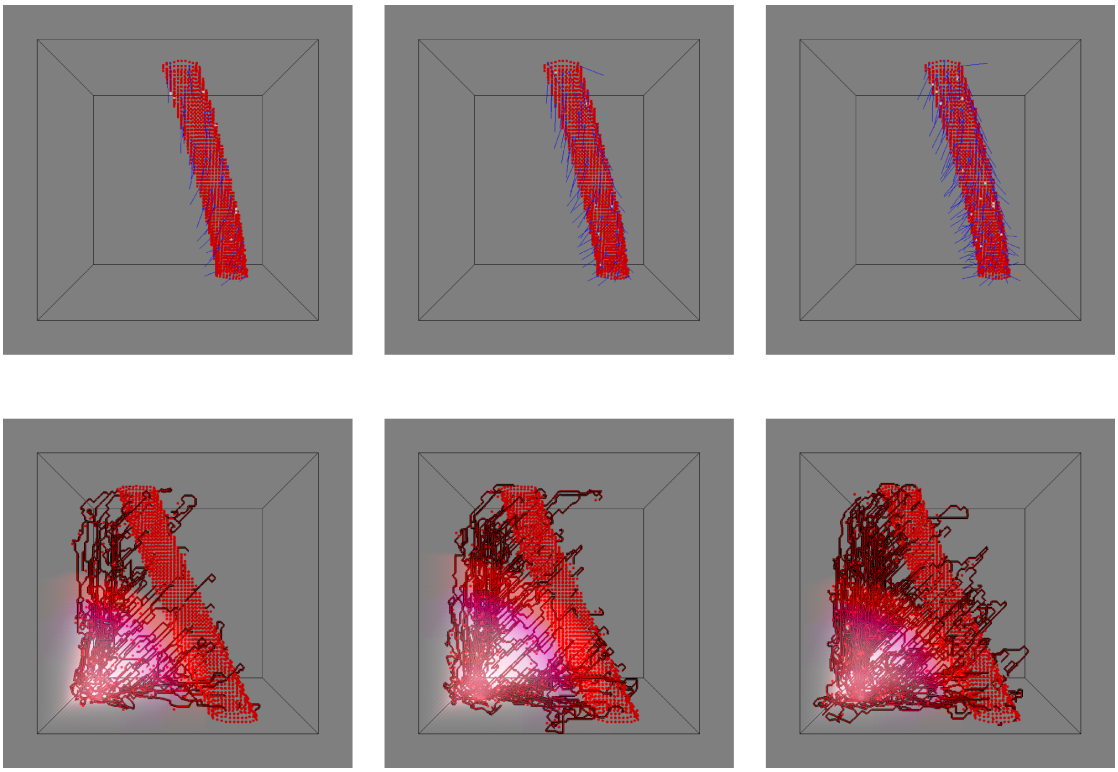


Figure 4.6: Decreasing the size of the exclusion zone around an activated endothelial cell results in increasing numbers of sprouting tip cells (top), and thus increased vessel density (bottom).

As the density of growth factor increases, so does the possibility of branching, that is, splitting into two vessels. Branching is a vital element in the formation of vascular networks, as it allows for the space between larger vessels to be effectively filled. In order for a vessel to branch it must be of a threshold age. When branching occurs, the original vessel sprout is considered to be complete and two new sprouts produced, thus the age is reset. The net result of this is that one new agent

is created and added to the environment at the location of the sprouting; this location is randomized in the locale of the original endothelial cell, and follows the constraint of occurring perpendicular to the direction in which the original cell last moved (cell choice is effectively determined by the dot product once again). While random, the location of the new sprout is weighted by the density of growth factors around the location of the old tip cell, so as to reflect the dependence of growth on the chemical factor. The vector representing the difference between the new cell and the original tip cell, that is the direction in which the the new cell has sprouted, is assigned to the cell as the direction of last movement, and thus encourages the sprout, initially at least, to move away from the parent vessel.

Anastomosis is an event which occurs when a migrating endothelial tip cell comes into contact with another sprout. When contact occurs, the migrating tip cell is essentially removed from the agent simulation, and the sprout becomes fixed. However, if an EC comes into contact with its own vessel sprout, anastomosis is ignored; this is primarily to prevent terminal anastomosis, that is early termination of the simulation due to random motion causing an EC to loop onto itself and stop movement. This does not remove the possibility of a new sprout, as described above, moving into its parent sprout and anastomizing. However, due to the uptake of growth factor by occupied spaces and the diffusion element, this is unlikely to occur; growth factor concentrations will be lower around vessels..

4.3 Physical constraints and boundary conditions

Every model is limited by the size of the simulation, although constraints vary over different models. Models of angiogenesis tend to be limited to a square or cuboid

environment, with no-flux boundary conditions imposed, such as in the model developed by Anderson et al [3]. Some notable exapmles are that of Lemon et al [48], who modeled angiogenesis within a virtual pore (analogous to an hourglass shape) to simulate angiogenesis within a porous tissue engineering scaffold and Perfahl et al [29], who employed differing sets of boundary conditions: initially no-flux as employed by Anderson et al [3] but also periodic boundary conditions.

In this model, boundary conditions were set to be no-flux, that is, nothing can enter or escape the confines of the simulated volume; fluids or agents. This has the effect of simplifying the model to some extent, removing the possibility of external influences. A constrained volume is required to prevent the simulation growing uncontrollably and arbitrarily increasing memory and processor requirements. Similarly, the size of the simulation is limited to an area approximately 1-2mm^3, which maintains reasonable memory requirements and execution speed.

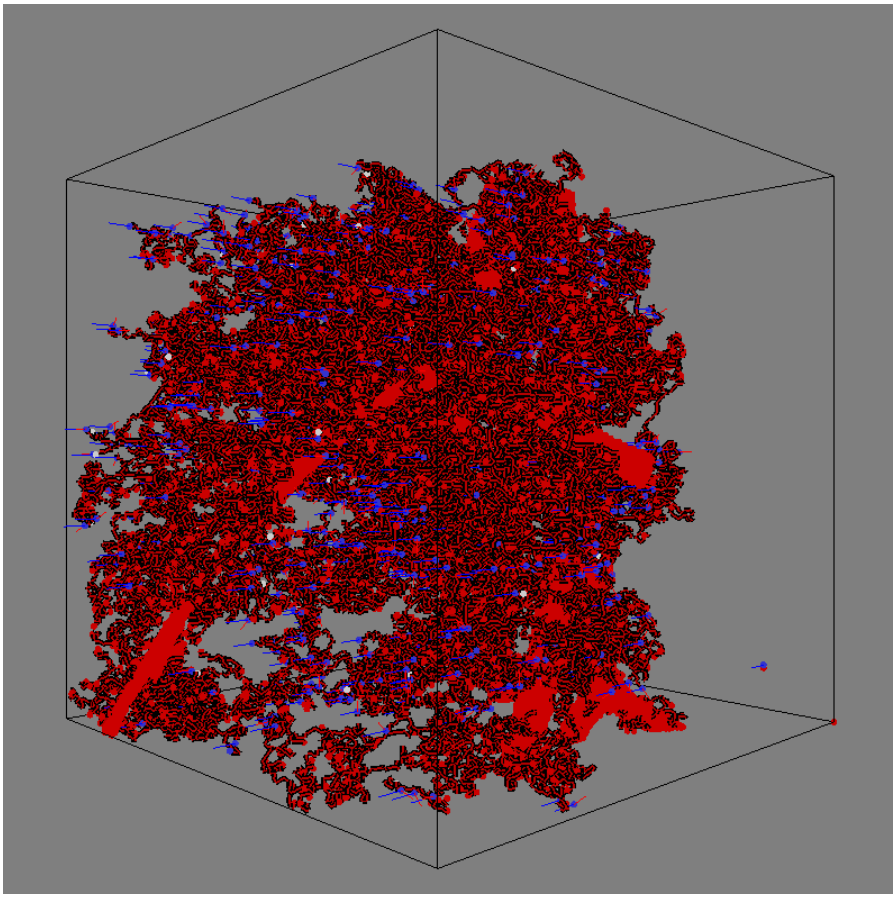


Figure 4.7: A snapshot of a 192^3 simulation running shows how complex the network generated can become. Original scan data (bright red) can be seen through the newly grown vessels (dark red) with moving tips cells higlighted in blue, and newly generated tip cells shown in white.

Chapter 5

Fluid dynamics simulation

5.1 Growth factor diffusion

A key element in the simulation is the chemical factors which trigger angiogenesis [11]. When released by the body chemical factors such as Vascular Endothelial Growth Factor (VEGF), Fibroblast Growth Factor (FGF), Platelet-derived Growth Factor (PDGF) and others, trigger the growth of the particular cells to which they are targeted. Of notable importance in angiogenesis is the factor VEGF, first isolated in the 1970s from tumor mass, which until recently was the only factor proven to be critical to the formation of blood vessels [49], and it is this factor which is considered here. It is its profound effect in the progression of cancerous tumors that has lead to such interest in terms of research, and this is why it is still the main focus of angiogenic studies.

Such factors are released by tumors into the surrounding tissue, through which transport is mediated by diffusion. These chemical factors are subject to both diffusion and decay, and thus must be produced in large enough quantities to

create a sustained gradient in order to cause endothelial cell migration.

The most basic numerical fluid dynamics implementation involves the diffusion or heat equation, where ρ is the quantity of some substance and d is the *diffusion coefficient* controlling the rate of flow:

$$\frac{\partial \rho}{\partial t} = d\frac{\partial^2 \rho}{\partial x^2} \tag{5.1}$$

In order to account for different rates of diffusion through varying materials we must solve the *inhomogenous* diffusion equation, which can be stated as follows:

$$\frac{\partial \rho}{\partial t} = \frac{\partial}{\partial x}\left(d_x \frac{\partial \rho}{\partial x}\right) \tag{5.2}$$

In a discrete model, this can approximated using the finite difference method as described in [47], which lends itself to a GPU-based implementation since it involves iteration of every element of an entire grid. Using the backward Euler formulation:

$$\rho(t + \Delta t) - \Delta t \, \frac{\partial \rho(t + \Delta t)}{\partial t} = \rho(t) \tag{5.3}$$

We can rephrase (5.3), where $\rho(t + \Delta t)$ is unknown, in matrix terms as:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.4}$$

Where: $\mathbf{b}$ represents the vector of known initial values of $\rho(t)$ in the system; $\mathbf{A}$ represents the matrix formulation of the transform; and $\mathbf{x}$ represents the vector of $\rho(t + \Delta t)$ values we are trying to find. We could compute the inverse of $\mathbf{A}$ and multiply both sides to find $\mathbf{x}$, but for a sparse matrix such as a diffusion transform

this is excessive. Instead we can use iterative relaxation methods that converge on the solution to the linear system.

Let $\rho_i = \rho_i(t)$, and $\rho_i^* = \rho_i(t + \Delta t)$, the discretization of 5.3 for the inhomogenous diffusion equation in 1D then yields:

$$\rho_i^* - \frac{\Delta t}{\Delta x^2}(d_{i+1}\,\rho_{i+1}^* - (d_{i+1} + d_{i-1})\,\rho_i^* + d_{i-1}\,\rho_{i-1}^*) = \rho_i \tag{5.5}$$

From this we can derive the base update equation for use in a relaxation method:

$$\begin{aligned}
\rho_i^*\,(1 + \frac{\Delta t}{\Delta x^2}(d_{i+1} + d_{i-1})) &= \rho_i + \frac{\Delta t}{\Delta x^2}(d_{i+1}\,\rho_{i+1}^* + d_{i-1}\,\rho_{i-1}^*) \\
\rho_i^* &= \frac{\rho_i + \frac{\Delta t}{\Delta x^2}(d_{i+1}\,\rho_{i+1}^* + d_{i-1}\,\rho_{i-1}^*)}{1 + \frac{\Delta t}{\Delta x^2}(d_{i+1} + d_{i-1})} \\
\rho_i^* &= \frac{\Delta x^2\,\rho_i + \Delta t\,(d_{i+1}\,\rho_{i+1}^* + d_{i-1}\,\rho_{i-1}^*)}{\Delta x^2 - \Delta t\,(d_{i+1} + d_{i-1})}
\end{aligned} \tag{5.6}$$

Setting an initial guess for ρ^* (for example, $\rho^* = 0$) and applying 5.6 over several iterations causes $\rho*$ to *relax* over time, converging on a stable solution (though not necessarily an accurate solution). For the Jacobi method we would replace ρ^* on the left hand side with $\rho^{(n+1)}$ and the right hand side with $\rho^{(n)}$, where n is the iteration number. That is, we would store the values of $\rho^{(n+1)}$ separately from $\rho^{(n)}$ ready to use on the next iteration. It turns out that this is not necessary, and in fact faster convergence is achieved when the results of the current iteration are used immediately for computing the next grid vale along in the lattice. This inline approach finally gives us the Gauss-Seidel method. In 3D

the iterative update equation becomes:

$$a = \Delta t \, (d_{i+1,j,k} \, \rho^*_{i+1,j,k} + d_{i-1,j,k} \, \rho^*_{i-1,j,k} + d_{i,j+1,k} \, \rho^*_{i,j+1,k} \\ + d_{i,j-1,k} \, \rho^*_{i,j-1,k} + d_{i,j,k+1} \, \rho^*_{i,j,k+1} + d_{i,j,k-1} \, \rho^*_{i,j,k-1})$$

$$b = \Delta t \, (d_{i+1,j,k} + d_{i-1,j,k} + d_{i,j+1,k} + d_{i,j-1,k} + d_{i,j,k+1} + d_{i,j,k-1})$$

Then:

$$\rho^*_{i,j,k} = \frac{\Delta x^2 \, \rho_{i,j,k} + a}{\Delta x^2 + b} \tag{5.7}$$

Where: $\rho_{i,j,k}$ is the known value $\rho_{i,j,k}(t)$; $\rho^*_{i,j,k}$ is the iteratively computed solution to $\rho_{i,j,k}(t + \Delta t)$; and $d_{i,j,k}$ is the location-dependent diffusion coefficient.

5.2 Flow and pressure

The pressure model is constructed using the graph representation of the angiogenesis model described in chapter 4. Nodes are formed where sections of vaculature meet, either through sprouting or anastamosis, which are connected by vessels represented as pipes. In order to measure pressure across the graph, we must calculate pressure values at each of the nodes. This is done by constructing an electrical analogue model such that conductance (i.e. the inverse of the resistance) of pipes is calculated, and pressure measured using a recursive algorithm.

The conductance, G, of a pipe is defined as:

$$G = \alpha \frac{\rho A^2}{8\pi\mu l} \tag{5.8}$$

where A is the cross-sectional area of the pipe, l the length of the pipe, rho the

blood density, μ the dynamic viscocity. Where $\alpha = 1$, this refers to flow across a perfect cylinder; this value was used to provide a simplification to the model.

The flow rate, F, of a pipe can be defined as the product of the conductance of the pipe and the pressure difference, Δp, across the pipe

$$F = G\Delta p \tag{5.9}$$

This is combined with the continuity equation which states that net mass flow through any node which is not an in- or out-flow node is zero, giving rise to the equation for each node i, where G_{ij} is the conductance of the pipe connecting node i with node j:

$$\sum_{j}^{n} G_{ij}(p_i - p_j) = q_i \tag{5.10}$$

where q_i defines the sum of in- and out-flow rates from the node i.

By applying equation 5.10 to each node, a matrix of values can be constructed which describes the relationship between the pressure values at each node as a system of simultaneous equations, such that the following relationship holds:

$$Mp = q \tag{5.11}$$

where M is the matrix of values constructed above, p is the vector of pressure values at each node, and q is the vector of flow rates at each node.

Flow rates are set to be constant values: 0 for the majority of nodes, 1 for an in-flow node and -1 for an out-flow node, although these values can be changed to reflect different flow characteristics of different systems. The aim is to balance

in- and out-flow of the system. In- and out-flow nodes are characterized as those derived from the original geometry which reside on the edges of the domain. By denoting the largest edge-residing section in vascular tree as an in-flow node, the remainder can naively be denoted as out-flow nodes, following which a balanced flow profile can be derived.

This equation is then solved iteratively for p until a steady state is reached. The matrix of coefficients, M, is diagonally dominant and, due to the nature of the application, incredibly sparse for even high complexity graphs. Thus, it is compressed before being transferred to video memory by storing only the non-zero values and the respective row and column indices. Since nodes tend to have an average of four pipes emerging from them, in a graph with several hundred nodes the storage required is several orders of magnitude lower. Not only does this save upload bandwidth and reduce memory requirements, but it also increases computational efficiency since only non-zero elements are required for computations. The equation (5.11) is solved iteratively for p using the Gauss-Seidel method, however other methods could be employed for larger systems such that greater efficiency is achieved such as described in [50]. Iteration ceases once the system reaches a steady state.

Chapter 6

Implementing complex computation on the GPU

Graphics processing unit (GPU) devices devote more of their hardware to raw data processing, rather than reserving a large amount of architecture for memory caching and flow control, as with conventional CPUs. It is because of this that they are best suited to data-parallel applications - linear arrays of data with each data element processed via the same program - focusing on high arithmetic intensity rather than memory operations. As a result, it is not a general purpose solution to computational problems. In this work, NVIDIA's CUDA (Compute Unified Device Architecture) is used. This section will discuss implementation details of both fluid dynamics systems, and how the GPU increases performance. The decision was made that the diffusion and pressure systems would be solved on the GPU in order to exploit the massively parallel nature of the hardware for increased performance.

6.1 The parallel programming model

Programming in parallel requires an entirely different way of constructing a program to that employed when working with a linear processor like the CPU. The GPU is specifically designed to run smaller, computationally intense programmes multiple times in parallel, as opposed to a longer set of instructions which would run in series. Even a low-end GPU (by current standards) can have as many as 384 cores, with those on high-end processors numbering in the thousands [51]. Further to this is the design of the processor; graphics processors are specifically designed to run multiple threads, with a single core on an NVidia processor being capable of running up to 1024 simultaneous threads.

CUDA C is an extended version of the C programming language, which allows for the creation of *kernels* which are functions that, when called, are run on multiple threads simultaneously. For example, a simple programme to add vectors is given as follows:

Listing 6.1: A simple programme written in CUDA C.

```
__global__ void VectorAdd(float *A, float *B, float *C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // kernel invocation with N threads
    VectorAdd<<<N, 1>>>(A, B, C);
}
```

The above code adds two vectors A and B of length N and stores the result in a third vector C. Then the programme is run, it creates as many threads (N) as there are elements in the vector. Each of the N threads adds one component of each vector, and hence the full vector addition is calculated. Note how the index into the vectors (which are in this case stored as arrays) is defined as the ID of the thread, which is accessed through the built-in *threadIdx* variable. This variable is itself actually a three dimensional vector; this assists in creating programmes which run on blocks of data in two or three dimensions as well as one. Execution is performed with the unique «<...»> syntax, which is an extension allowing the definition of multi-dimensional arrays of threads. This example executes *one* block of N threads.

Blocks are used to group threads together with the constraint that all threads within a block must be executed on the same core. Since a single core can execute up to 1024 threads, this constraint is passed on to the size of a block. Thread block dimensions are defined by passing a vector of sizes when executing a kernel, such that the product of the sizes does not exceed the maximum limit for threads per block. A thread block can be defined as a one, two or three dimensional vector, which allows for identifiers to be accessed as described above.

CUDA C is not an object-oriented language, and as such any operations are performed on shared memory assets, stored in graphics memory. With a multi-dimensional block of threads, the thread identifier can be used as an index into an array of the same dimension, giving a natural way to access memory from within the kernel. Such a model of computation is ideal for modelling block calculations as opposed to more intricate systems which require a single element to retain information about itself.

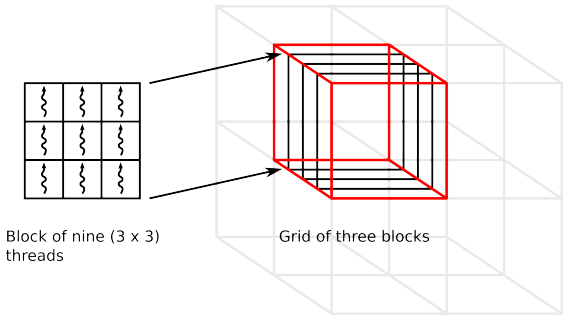


Figure 6.1: Threadblocks are representative of sub-slices of the overall simulation volume, which are layered to form a grid for the purposes of execution on the graphics card. Grids with different coordinates are successively solved in order to process the entire volume.

6.2 Diffusion

6.2.1 Diffusion equation

The transfer of chemicals within the tissue of the body is primarily though the means of diffusion. For example, oxygen diffuses from the blood in the vessels, through the endothelium, and into the surrounding tissue, up to a distance of around 200μm (referred to as the diffusion limit of oxygen). The purpose of adding a diffusion component to this system is to describe the motion of vascular endothelial growth factor from a source, typically a tumor, in the tissue.

Diffusion is the spread of a fluid through a space in a (usually) uniform manner, as described by the diffusion equation:

$$\frac{\partial \phi(\mathbf{r}, t)}{\partial t} = \nabla \cdot [d(\phi, \mathbf{r}) \nabla \phi(\mathbf{r}, t)] \tag{6.1}$$

which describes the rate of change of density ($\phi(\boldsymbol{r}, t)$) at time t at location $\boldsymbol{r}$ with respect to time, $d(\phi, \boldsymbol{r})$ is the diffusion coefficient at $\boldsymbol{r}$, and ∇ is the vector differential operator del.

The diffusion problem consists of needing to know the fluid density at a specific location within the simulation environment. Thus, the diffusion equation is solved for each location within the three-dimensional grid simultaneously using the forward difference scheme described in chapter 5, and a Gauss-Seidel method of computation.

6.2.2 Memory requirements

When setting up the diffusion solver, certain memory requirements have to be met. Initially, a three-dimensional array of fluid concentration values is required which represents the current state of the system. The dimensions of this array must match the dimensions of the simulation environment. Using a three-dimensional array instead of an appropriately sized one-dimensional array allow for threads running the diffusion kernel to index into the array using the thread identifier "threadIdx" as described in the previous section, along with the block identity "blockIdx". Block identities can be indexed in much the same way as thread identities, and when used with an invariant block size can simply be multiplied up to retrieve the full index into the array.

As a single thread block can support up to 1024 unique threads, a reasonable size for the block is eight in each dimension, giving a total of 512 threads. While this may seem like a waste of computing resources, it fits nicely with the convention of using powers of 2 in computer science. Thus, simulation environments can be set up with sizes which are multiples of 8, allowing for easy splitting of the environment into blocks. This also aligns well with memory, preventing the need for padding bytes.

To implement the Gauss-Seidel algorithm the system is double-buffered. The programme is able to overwrite the current value of guess with the newest guess, in line with the operation of the Gauss-Seidel method. In order to safeguard against incorrect values being read from memory, all read and write operations must be atomic in nature, that is the operation is guaranteed not to be interrupted by another thread, and the CUDA API provides this capability through a set of dedicated functions. Thus the above memory requirement is doubled.

Further to this, diffusion constants are required for the entire simulation environment, but since these do not change during execution of the diffusion solver, only a single array is required. As before, a three-dimensional array is used to allow for a natural method of indexing based on thread and block identities.

These arrays are stored in memory on the graphics card, typically referred to as the *device*, however there is also the need for an array of fluid density values within main, or *host*, memory, such that access to values by cells is as fast as possible. This array will be equal in size to its counterpart on the *device*. Transferring data between *device* and *host* memory is a bottleneck, however the increases gained through the accelerated computation speed of the GPU, and local memory access speeds for cells (as opposed to numerous individual reads from GPU memory) more than make up for this.

6.3 Pressure

Being its primary function, the movement of blood is of key interest in studies of vascular structure and function. At its most basic level, the vasculature can be described as a series of tubes. These tubes vary in length, and diameter, and

in function. An appropriate analogy might be to compare the vascular network to a tree; larger vessels are more akin to the trunk, which supports the smaller branches through the transfer of water (analogous here to blood), however it is not until the smaller branches where the water is transferred to the leaves (akin to our tissue). It is at the microvessel level where the walls become thin enough to allow the passing of nutrients and waste products to and from the blood.

Poisueille's 1840 work *"Recherches expérimentales sur le movement des liquides dans les tubes de très-petits diamètres"* on the movement of fluid in pipes led to Poiseuille declaring the relationship between the flow rate (Q) of liquid through a pipe of uniform diameter(D) and fixed length (L), and the difference in pressure across the length of the pipe (p), as follows:

$$Q = k(D^4 p/L) \tag{6.2}$$

where k is a constant. This can be rearranged to give what is commonly referred to as Poiseuille's Law, defined as:

$$\Delta P = \frac{8\mu LQ}{\pi r^4} \tag{6.3}$$

where ΔP is the pressure difference across the pipe, L is the length, Q is the volumetric flow rate, μ is the dynamic viscosity of the fluid, and r the radius of the pipe, π being the mathematical constant *pi*.

With respect to the dynamic viscosity of the blood, the work of Pries et al.[41] comes into play, with their equation for "relative apparent viscosity", which describes the phenomenon of a sharp increase of apparent viscosity observed in vivo for vessels of diameter smaller than (approximately) 20μm (see diagram 6.2), and

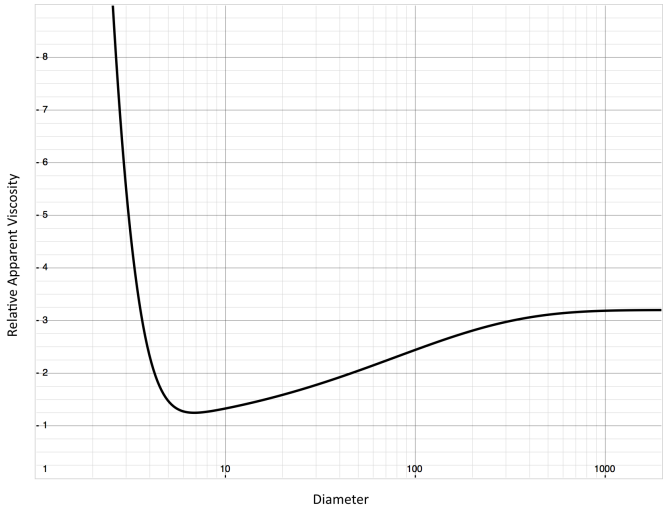


Figure 6.2: Relative apparent viscosity, as described by Pries et al., shows a sharp increase in the apparent viscosity of blood at very small vessel diameters. This is due to the nature of blood as a fluid being not uniform, but a suspension of particles (red blood cells, primarily) in a fluid (plasma).

is of the form

$$\eta_{vivo} = 1 + (\eta_{0.45} - 1) \cdot \frac{(1 - H_D)^C - 1}{(1 - 0.45)^C - 1} \tag{6.4}$$

where H_D is the discharge haematocrit for vessel diameter D; $\eta_{0.45}$, the relative apparent blood viscosity for a fixed discharge haematocrit of 0.45, is given as

$$\eta_{0.45} = 220 \cdot e^{-1.3D} + 3.2 - 2.44 \cdot e^{-0.06D^{0.645}} \tag{6.5}$$

and C describes the velocity dependence on haematocrit as

$$C = (0.8 + e^{-0.075D}) \cdot (-1 + \frac{1}{1 + 10^{-11} \cdot D^{12}}) \tag{6.6}$$

In the range of vessels of 7μm to 100μm, the difference in apparent viscosity of blood with discharge haematocrit 0.45 and cell-free plasma (haematocrit 0) is

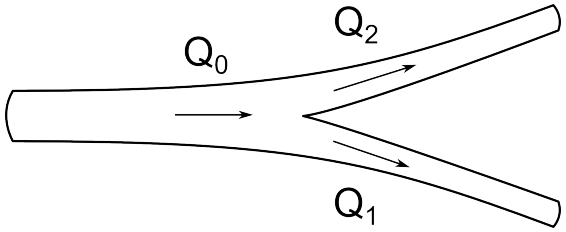


Figure 6.3: Flow is preserved across a node. The sum of the flow into the node *(Q_0)* is equal to the sum of the flow out of the node *(Q_1 + Q_2)*

only 25%, indicating a weak dependence on haematocrit [52]. As such, a computational simplification can be made of assuming a discharge haematocrit of 0.45, rendering the fractional term of equation 6.4 equal to one, leaving the viscosity to be described by the fixed term of $\eta_{0.45}$, equation 6.5.

If we consider once again the vascular network, and how we see it is described by a series of tubes, we begin to form a concept of how blood flow can be described using equation 6.3. Consider figure 6.3, which displays a simple bifurcation in a vessel. This can be considered as three pipes, having flow rates of Q_0, Q_1 and Q_2.

The graph topology is derived from a simplification of the vascular network created using the algorithm described previously. Figure 6.4 shows a side by side comparison of a simple vascular network produced by the vascular growth algorithm, and a visualisation of the corresponding graph representation used in the pressure calculations. Sections of microvasculature generated are treated as straight pipes in order to reduce the complexity of the calculation. Here, a *section* is any uninterrupted length of vasculature in the resulting network; where a vessel branches, or is met by another sprout, a node is formed.

Calculating blood flow is performed by solving for pressure on the GPU, which becomes a case of formulating the network as a large system of simultaneous equations. The problem, then, is solving a system of equations with many hundreds of

variables, and thus an iterative system such as Gauss-Seidel once again becomes attractive as it allows solution of each equation in a semi-independent manner.

Equations are created for the pressure at each *node*, that is any location where two or more pipes or sections of pipe meet. The pressure at a given node is given by equation 5.10. As such, we construct equations relating the pressure values at each node to flow rates at the in- and out-flow nodes. In this context, an equation is a list of coefficients of conductance values of pipes which meet at the specified node, multiplied by the pressures, such that a large, sparse, diagonally dominant matrix A is formed which satisfies the condition:

$$Ax = b \tag{6.7}$$

where x is the vector of node pressures, and b the vector of flow values. It is important to note at this point that the flow values are fixed, and for the majority are zero as they refer to the sum of in- and out-flows at a particular node. In-flow nodes are designated a positive flow value, whereas out-flow nodes are designated a negative flow value.

In a complex system, a square matrix of many hundreds of lines can be produced. However, due to the nature of the system, a large majority of elements in the matrix are zero, and thus require no storage. In fact, when we consider that an equation is required per-node, it becomes clear that the storage requirements for the matrix can be reduced to the use of row and column indices and element values for non-zero elements only. The choice to treat uninterrupted sections of generated vasculature as single straight pipes further reduces the size of the matrix.

Programming the iterative scheme for the GPU requires choosing an algorithm

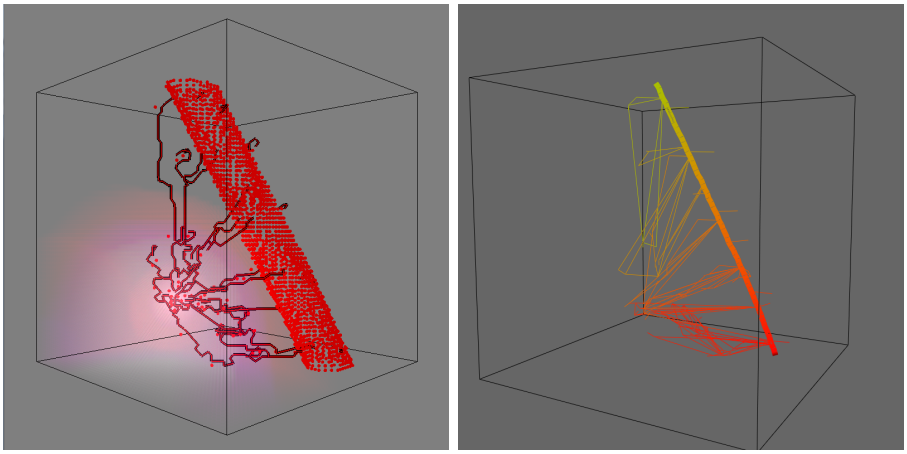


Figure 6.4: A side by side comparison of a simple vessel network grown with the algorithm described previously and the graph representation for the pressure calculations. The graph assumes straight pipes for conductance calculations but uses vessel length instead of the straight line distance between nodes.

which can operate on each equation, that is each line of the matrix, independently, in order to solve the equation Ax = b. Initially, a simple scheme such as Jacobi seems attractive as it works on each line, iterating once each new x (pressure) value has been computed. However, other schemes could be implemented, such as those in [50].

This line-based method for solving the equation, along with the storage requirements outlined above, define the method for storing the main matrix M in memory. This is comprised of three arrays for storage of values as follows:

- an array, Av, of all non-zero values in the matrix, packed in sequence from left to right, top to bottom.
- an array, Ac, of column indices for each element in the array Av - each column index $Ac[i]$ relates the element of the matrix $M[i]$
- an array, Ar, of row indices, where each element $Ar[i]$ contains the index in

Av of the first element in the row i.

With this are two more arrays, one for the x vector and one for the b vector. Additionally, an array of convergence flags can be implemented to allow early termination of the algorithm.

Since these arrays are only populated at the beginning of the solving sequence, elements can be read from them with the confidence that they have not been modified by the solving function (kernel).

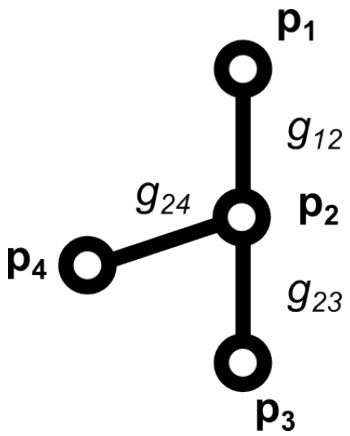


Figure 6.5: Example graph: flow values at nodes p_1 and p_3 are fixed as 1 and -1 respectively, and given fixed pressure values. Conductances g_1, g_2 and g_3 are calculated for each section of vasculature.

$$\begin{bmatrix} g_{12} & -g_{12} & 0 & 0 \\ -g_{12} & g_{12}+g_{23}+g_{24} & -g_{23} & -g_{24} \\ 0 & -g_{23} & g_{23} & 0 \\ 0 & -g_{24} & 0 & g_{24} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{6.8}$$

The equations are solved iteratively for p using the Gauss-Seidel method, however other methods could be employed for larger systems so that greater efficiency

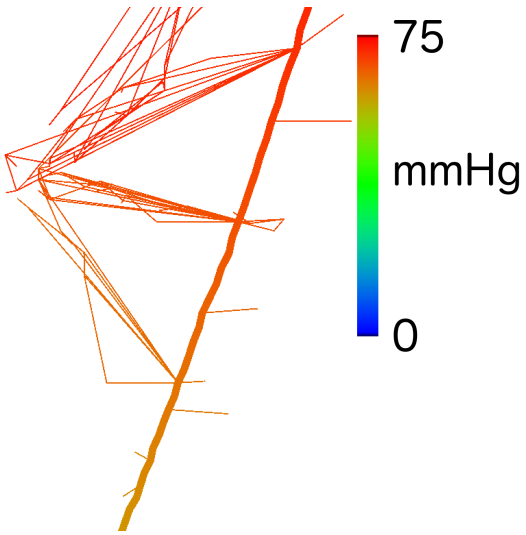


Figure 6.6: Pressure is stabilised across the network after successive iterations of the algorithm. Pressure values range from 75mmHg to 55mmHg, as defined as the in- and out-flow pressure values, which are fixed.

is achieved, such as those described in [50]. Iteration ceases once the system reaches a steady state.

6.4 Mass transfer

The main function of blood within the body can be characterised as the transfer of materials from one location to another. Blood carries nutrients, such as oxygen, throughout the vascular network, as well as antibodies and waste products. However, this same transport mechanism can transfer harmful substances, such as bacteria, or therapeutic substances, such as drugs, as well. Thus, modelling the transfer of materials through the vascular system has wide-reaching implications in terms of planning treatment regimes or assessing the possible efficacy of therapeutic treatments such as chemotherapy [53].

Mass transfer in vascular networks is controlled by two processes; advection

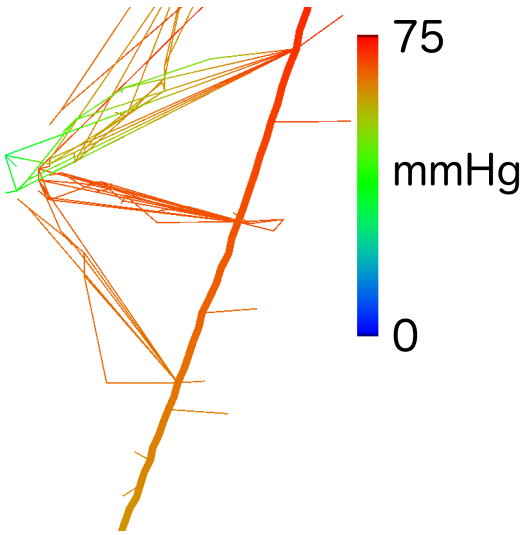


Figure 6.7: Modifying the graph by introducing low pressure nodes shows how the algorithm stabilises under different conditions.

and diffusion. Advection is the transfer of mass through bulk movement of the surrounding material. Within the vascular system, advection occurs as blood moves through the system, causing movement of a substance dissolved within the blood fluid. This is particularly important within the larger vessels as the blood moves at a much faster rate than in microvasculature. Diffusion can occur in the absence of bulk motion, and lacks a macroscopic explanation. Diffusion is explained by motion on a molecular level, and is the result of the random motion of particles at a microscopic scale.

6.4.1 Mass transfer by advection

When considering the effects of advection on mass transfer within a system such as human vasculature, it is important to consider what is happening. Advection within the vasculature is the result of the motion of blood, as propelled by the pumping action of the heart. As described above, the simulation calculates

pressure values for pipe intersections, thus the pressure difference across a pipe, Δp becomes trivial to calculate. Using this pressure difference, we can apply the formula for flow rate of fluid in a pipe

$$Q = \frac{D^4 \pi \Delta p}{128 \mu l} \tag{6.9}$$

where Q is the flow rate, D is the diameter of the pipe, μ is the specific viscosity of the fluid and l is the length of the pipe. Flow rate is calculated in cubic metres per second, and thus by multiplying the flow rate by the simulation time step, Δt, we can measure the volume of fluid moving out of, or into, a pipe in this time.

We model the node volume as opposed to pipe volume. We define node volume as follows

$$N = \frac{1}{2} \sum_{i=0}^{k} P_i \tag{6.10}$$

where P_i is the volume of pipe i which has one end at the node in question. Thus the total volume of the system can be expressed as the sum of the node volumes.

To model mass transfer by advection, we look at the transfer of liquid between nodes. Recalling that the net flow across a node is 0 (except in boundary cases) and considering the flow rates calculated as above, the movement of solutes is modelled as a change in overall concentration at a node. This is discussed in more detail in the following section, but is based on multiplying concentration fractions at each node with the volume of fluid moving through the pipe. The whole system is double buffered to reflect the nature of the process taking place, i.e. all elements being calculated simultaneously.

Diffusion plays a role in the transfer of mass throughout the vasculature. In this model, diffusion is limited. The concept of diffusion is used here when considering the mass concentration within node volumes. Let us consider the result of a single simulation step between two connected nodes; we define the higher pressure node, n_h, as having volume v_h and solute concentration c_h, and the lower pressure node n_l, as having volume v_l and solute concentration c_l. There is movement of a fixed volume Δv of blood from the higher pressure node to the lower pressure node, as well as the removal of the same volume of blood from the lower pressure node. Since we assume node volume to be constant, this results in the low pressure node having combination solute concentration of

$$\frac{(v_l - \Delta v) \cdot c_l + \Delta v \cdot c_h}{v_l} \tag{6.11}$$

It is at this point that we must consider the effects of diffusion. The representation of solute concentration as being calculated on a per-node basis is a discretization of an inherently continuous property. It would be computationally infeasible to expect to be able to represent a spectrum of such a property at a truly accurate (i.e per-molecule) basis. Thus, the concentration values we calculate for each node are in fact average concentrations, as we assume mixing due to diffusion within a node. We consider this assumption reasonable due to the volume of the pipes and the time scale (100ms per step) of the simulation.

There are some obvious computational simplifications in this scheme which are ideal targets for improvement. Whereas pipes in this model are treated as straight and rigid, the method of calculating conductance lends itself to modelling more complex pipe geometry. Increasing the accuracy, and therefore complexity,

of conductance calculations would obviously result in an increase in processing requirements, although these values change infrequently and thus need not be recalculated often.

While the model deals with mass transfer through the vascular network, it does not incorporate any absorption terms. Absorption of solutes through the vessel wall is moderated by a collection of factors, including the rate of diffusion per unit area (itself dependent on the transport mechanism and vessel wall thickness), the surface area of the vessel, and the volume of tissue behind the vessel wall. This can be modelled using a scheme similar to the one proposed in [54], via the application of Fick's Law of diffusion. While the majority of this is reasonably trivial, association of tissue volume to vessels requires some segmentation of the simulated volume; this can be reasonably easy to achieve in simulations of a fixed network, but application to a growing network would be somewhat more complex.

Furthermore, it is important to remember that blood carries not one but a multitude of solutes, through different transport mechanisms. Oxygen, for instance, is transported via the red blood cells (RBCs) themselves - thus oxygen transport is dependent on the fraction of RBCs in the blood (haematocrit) flowing through the vessel in question. On top of this is the fact that the smallest vessels are at times smaller than the width of a single RBC, which results in compression as they move through. Other mechanisms of transport across the endothelium are numerous, in particular across the blood-brain barrier [55], [56],[57], a functional boundary between cerebral vasculature and the tissue of the brain itself.

The particular coupling of a model of angiogenesis and advection-based transport lend themselves to simulation of blood-brain barrier interactions. Since we define each unit length of vasculature generated as equivalent to a single endothelial

cell [58], we can make assumptions about where the tight junctions (those where the cell bonds to itself or other endothelial cells to create tubular structures) are, i.e. between each unit of vasculature, and to a lesser extent along one edge of each unit.

6.4.2 Delivery of mass to the body

The rate at which diffusion across the capillary endothelium occurs is dependent on the location of the vessels in the body. The cerebral vasculature are tightly regulated to protect the brain from transfer of toxic substances (although this is not always effective), whereas kidney and liver vessels are designed to allow for efficient cleaning of the blood as it travels through.

There are several cases we can consider when it comes to mass transfer in therapeutic circumstances. These include the transfer of metabolites, such as oxygen and glucose, the removal of waste products, and the targeted transfer of drugs. The majority of these can be categorised as the diffusion of substances across the endothelium into the surrounding tissue.

With respect to the grid-based network described in previous chapters, calculating the rate at which solutes are transferred across the endothelium requires knowing the surface area of vessel over which the solutes are diffusing. This can be achieved by counting the number of empty spaces adjacent to the vessel (see figure 6.10).

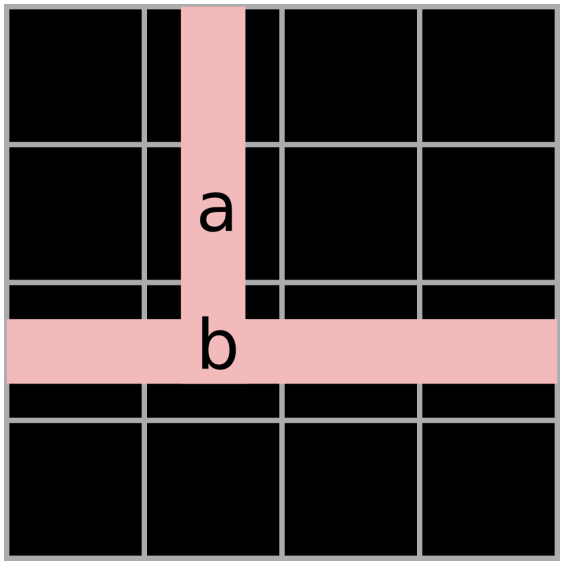


Figure 6.10: Counting spaces adjacent to a vessel segment - in this simplified (two-dimensional) example, the location marked 'a' has two adjacent spaces, whereas the location marked 'b' has only one.

A simple tubular section in the middle of a straight vessel will have four neighbors. The surface area of this section can be calculated simply with

$$SurfaceArea = \pi r^2 l \tag{6.12}$$

where r is the radius and l is the length of the section. Aligning this with the grid-based representation here, it is easy to see that the surface area represented by a single neighbor is a quarter of this. This is appropriate for straight sections, however for more complex sections computing the surface area becomes more difficult. A few strategies are available for this problem.

The most naïve and computationally intensive solution is to calculate the surface area of each section for each simulation step. While accurate, this has obvious disadvantages. A more efficient scheme involves a simplification to the surface area

calculation. Based on the above measure of surface area between neighboring locations, the surface area of any section can be approximated as:

$$SurfaceArea = n\frac{\pi r^2 l}{4} \tag{6.13}$$

where n is the number of neighboring locations which are empty.

A final solution is to create a lookup table based on the possible configurations of pipe intersections. By limiting the calculations to the six nearest neighbors ($\pm$ 1 unit in the x, y, or z dimension), this reduces the complexity of such a scheme to a maximum of 2^6 possible combinations. This can be further reduced by considering that many combinations are rotations of others.

6.5 Programming for the GPU

Given the update equation 5.7 defined in chapter 5, and the memory requirements outlined above, writing the kernel for the diffusion programme is reasonably straight forward. Computing the next iteration involves retrieving both the diffusion constants and the fluid densities at the location to be calculated and its neighbors in each dimension. These are accessed using the thread identifier, threadIdx, as follows:

```
fluidVal = fluidArray[threadIdx.x][threadIdx.y][threadIdx.z];
```

here, the current value of the location is retrieved. To access the neighboring values, the array indices are modified.

Once the diffusion constants and fluid values have been retrieved, these are used to calculate the varying components of equation 5.7, with a time step (Δt)

value which is chosen at the beginning of execution, and a constant grid spacing (Δx). The result is then written back into the fluid array, to be used to continue execution.

Calculating the pressure values revolves around a similar set of operations, including the execution of the kernel (although here we execute a linear array of threads, one per line, as opposed to a three-dimensional grid), and memory fetching operations.

Using the Jacobi iterative formula:

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^k) \tag{6.14}$$

we can compute iterative values of x, which represent the pressure values. When calculating by hand, the above formula is used on each line of equation 6.7, however the GPU kernel allows for simultaneous evaluation of each line on a separate thread. The Jacobi method requires each line to be complete before the next iteration is calculated, but due to the very sparse nature of the matrix A, each iteration only involves a small number of items to be summed, and thus differences in execution time between lines will be very low.

Integrating with the main programme requires developing a set of access and control functions to initiate execution on the GPU and deal with memory allocation and access. The CUDA API provides a library of functions which are available to manage memory and programme execution on the graphics card. Rather than using the *new* operator as is normal in C++, we must make use of the functions *cudaMalloc* and *cudaMalloc3D*, which allow for the allocation of one-dimensional and three-dimensional arrays respectively, as single blocks of memory, enabling us

to access it in the manner described above.

Similarly, modifying or retrieving values in graphics memory from the CPU involves using dedicated functions *cudaMemset* and *cudaMemcpy* respectively. These functions are designed to work the same way as their C counterparts, making memory manipulation easy for anyone with some level of C programming skill. Use of these operations is minimised, however, since transfer between graphics memory and main memory is slow.

6.6 Unifying a modular system

Until now, the various components of the system have been discussed as separate systems, almost acting independently of one another. At this point it is important to consider the *functional unit* of the simulation. As a discretised model of a continuous system, the simulation environment is described as a three-dimensional grid of adjacent locations, which reside within a cubic section of tissue. The term *functional unit* refers to an abstract concept which does not represent any physical part of the system, but instead describes any single location on the grid, and encapsulates the properties of the cellular motion, diffusion, and pressure subsystems at that location.

The concept of this system abandons that of typical layered multiscale models, which involve modelling on a variety of biological timescales [59], and instead adopts a centralised model where each subsystem acquires information about the others through this functional unit. This is achieved through the use of a messaging system which has the effect of decoupling sub-systems from each other and allowing a modular approach to simulation building.

The functional unit, being just a single element in the grid, is created for each element in the grid. As such, a centralised controller is created which handles their creation at run time. It is through the controller that a subsystem registers which properties are available for the functional unit to retain, through a function supplying a property name and type. This is done at the beginning of the simulation, before the grid is created, so that each unit in the grid is initialised with the same properties. Properties can be numerical values, such as chemical densities or occupancy, or can be pointers to a larger structure elsewhere.

Each subsystem is responsible for updating the functional unit as required. When a subsystem requests a value from the functional unit, the unit must return whatever value it currently holds for the property requested.

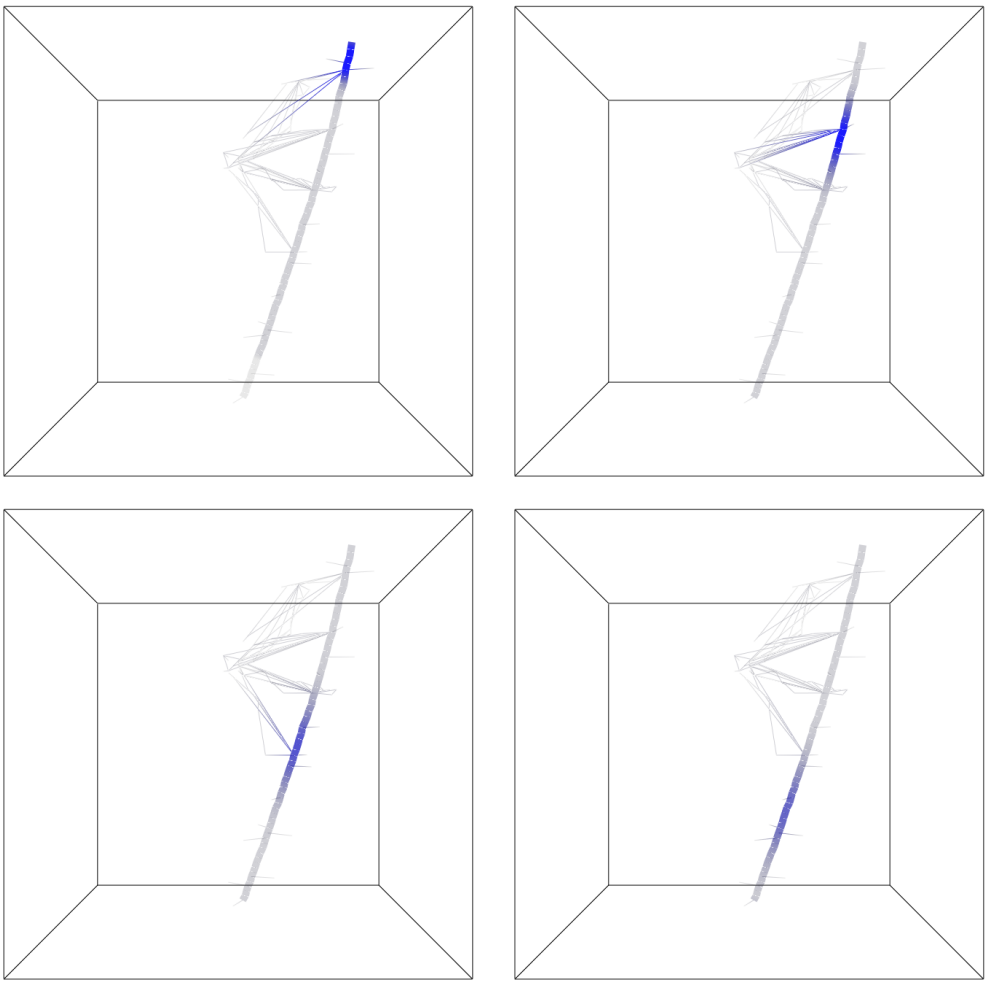


Figure 6.8: Mass is transferred throughout the network over time as advection takes place. The density of the solute appears to reduce in the main vessel as it spreads throughout the system.

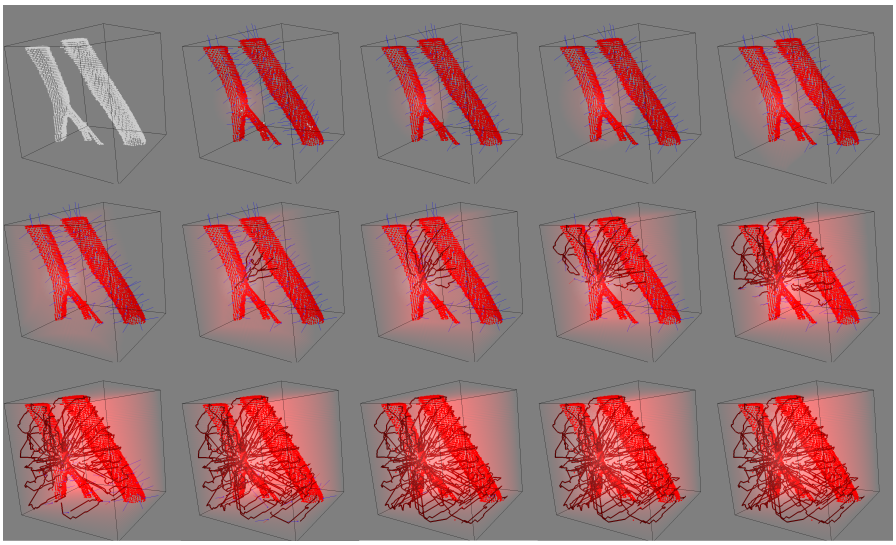


Figure 6.9: Example of growth of a capillary network over time. Initially, cells are nascent. Introduction of a growth factor gradient (pink) activates cells (blue cells) around which is an exclusion region (red cells). As the growth factor density reaches a threshold level, activated cells migrate towards the source of the gradient. New vessels are drawn differently to highlight topographical features.

Chapter 7

Individualised modelling

7.1 Introduction

In 1895 Wilhelm Röntgen discovered X-Ray radiation, and noted how the human body absorbed these rays in differing amounts throughout the different types of tissue [60]. This, combined with photographic paper, gave way to the first form of what we recognise now as medical imaging - that is, advanced techniques which can image the inside of the human body in a non-invasive way. In the last few decades medical imaging has provided an increasingly accurate way of viewing the internal state of a body (or other object, as these techniques are not limited to living things) without the need for any invasive procedures, such as exploratory surgery. The uses of imaging techniques are widespread, and partial or complete body scans are common in diagnostic medicine; for example, ultrasound scans for assessing the health of an unborn child.

The advent of new imaging techniques, and later more powerful scanners, has offered a way of viewing more than just two-dimensional images of the inside of the

body, such as an X-Ray displays. Some modern scanners, such as MRI machines, are capable of acquiring entire volumes of space as a series of 2D images, or slices, which when composited form a three-dimensional image of the body being scanned. Others rely on post-processing of a series of images, such as X-Ray Computed Tomography, which takes X-Ray images acquired at specific intervals across a 180 degree rotation to reconstruct a three-dimensional image of the object being scanned.

Thus, an accurate internal representation of the body can be obtained. It is from this that we are lead to the idea of personalized medicine, that is the customization of treatments to individuals based on the physical features of their internals. Such an approach has been used already in surgical planning, for example in measuring the width of an artery where a stent is required to provide a personal, custom piece of equipment and personal treatment.

Beyond this, however, is the idea of predictive medicine - taking a persons unique data and using that to provide a model of how a disease is likely to progress. The idea that vascular structure is relevant in disease is becoming ever more prevalent in the research community. The need for modelling based on a persons own unique data thus becomes more apparent.

7.2 Personalised medicine

More common these days is the idea of personalised medicine. This is the concept of specifically tailoring treatments and diagnoses to an individual, based on not just medical history, but actual data regarding the patients physical body. Scanners are now a permanent fixture in every hospital, and their use in diagnosing a variety

of maladies is routine, such as the use of X-Ray imaging in the diagnosis of broken bones and obstructions, to the use of Magnetic Resonance Imaging (MRI) in the diagnosis of cancer.

In fact, MRI scans are now commonly in use during the planning stages of cancer surgery and the design of replacement bones for those who require them. The evolution of this is predictive medicine: personalised simulations of the progression of disease, in order to best plan treatment of a condition. As scanner fidelity increases, and our knowledge of disease pathogenesis grows, it becomes apparent that the combination of these elements could lead to improved treatment of medical conditions. This can also be applied to simulating the effects of a given treatment, to predict how a patient might respond over a given period of time.

7.3 Imaging

The type of images required is governed by the particular application required. For example, design of an implant, and surgical planning for fitting, can use data retrieved from a full body MRI scan at resolutions offered by typical strength scanners found in many hospitals, since they offer sub-centimetre, and even sub-millimetre accuracy. However, modelling of phenomena such as cell growth requires a much greater resolution, in the region of micrometres, to match that of the size of the cells being modelled and accurately represent the surrounding environment.

The images used in this work were MicroCT scans. The reason for this is that the scanning resolution is very high, up to values as small as 5μm [61]. This value corresponds well with the size of endothelial cells, which are the main target of the model.

It is at this point that we must consider the structure of blood vessels, and how this affects and is affected by the process of angiogenesis. Blood vessels, particularly the microvasculature, are composed on the outer layer of a matrix of endothelial cells. During the early events of angiogenesis, endothelial cells are recruited from the vessel wall by the angiogenic factors, and start migrating up the factor density gradient.

Thus, in terms of modelling this phenomenon, we must be able to model endothelial cell movements on a scale at least equivalent to the size of a single cell - that is, the domain in which we model is split into units of approximately one endothelial cell in size. This leads to the assumption that each location in the grid is either 'on' or 'off', that is occupied or empty. It allows for easier testing in terms of whether a cell can move into a location in the simulation space, or not.

We can also consider the structure of capillaries at this stage. Because the larger vessels are considerably wider, the circumference of such a vessel will be composed of multiple endothelial cells. However, capillaries are not structured this way. In fact, a capillary is similar in size to a single endothelial cell - its circumference is created by a single cell forming a tubular section, which is then connected to other sections to produce longer pipe structures.

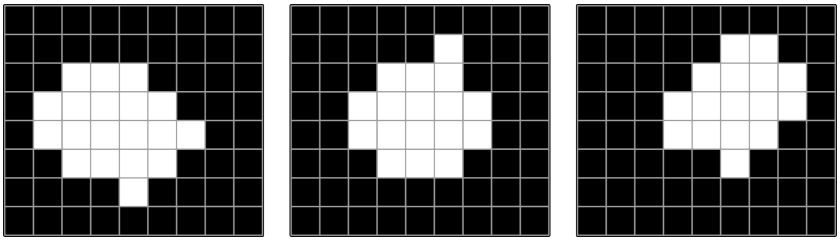
The scan data is being used to create the initial conditions for the simulation. The idea here is that the scan shows the existing vasculature, and the simulation is run to discover how the introduction of angiogenic factors affects it. So when images are loaded in at the beginning of the simulation, some thresholding is performed to determine which locations in the simulation space are already occupied. Since the scan data relates to vascular structure, this is then taken to mean that an occupied space relates to an endothelial cell in the vessel lumen.

Following this, the diffusion coefficients for the occupied locations are set to a higher value, which represents the decrease in permeability compared to that of the extra-cellular matrix. Vessels are then hollowed out. This is achieved by locating occupied cells which have occupied neighbors in all six of the principle directions (up, down, left, right, forward, backward) as described by figure 7.1. These are removed from the simulation, since they represent areas within the vessel, which would be filled with blood. These locations, which now appear empty, are marked as filled, however, to prevent the accidental movement of a cell into the vessel space. *In vivo* this would be prevented by the mechanics of blood flow; pressure of blood would push cells outward. Similarly, the migrating cells bind onto the extra-cellular matrix, and this would be absent, and thus there would be nothing onto which the cells could anchor themselves.

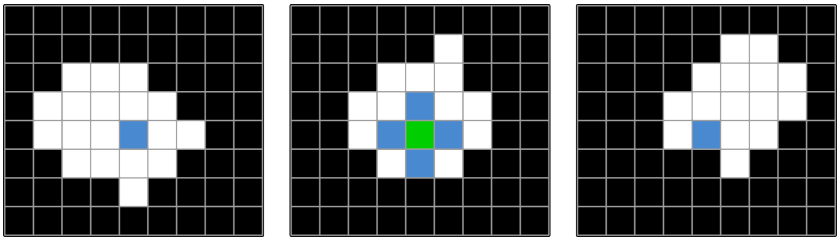
To construct the initial graph representation of the vessels for blood flow simulations slices are processed as they are imported into the simulation. For each slices, regions are calculated using region growth, and individual regions are tagged as sections of pipe in the graph. These are approximated as having a length equal to the thickness of the slice (defined by the scan resolution), and are idealised slightly by calculating a radius based on the area of the region; assuming a circular section, this is easily calculated as $r = \sqrt[2]{\frac{\pi}{A}}$.

Pipe sections generated in this manner are connected through detection of overlap in successive slices, and graph nodes are created in-between these sections. In the event that a new region overlaps more than one region in the previous slice, this is easily handled by the graph system.

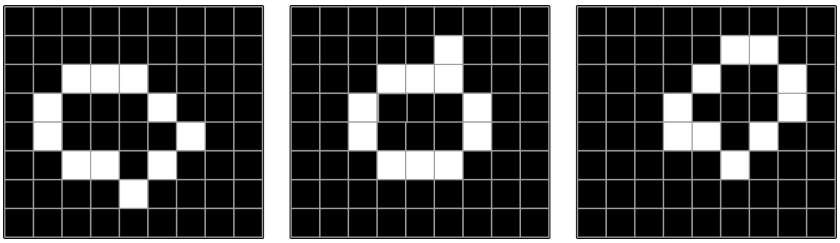
At the same time, the endothelial cells instantiated by processing the slices are assigned to the pipe sections created from the same region. This ensures that



(a) An example of scanned images before hollowing is performed, sections are described as contiguous solid blocks. A white pixel denotes space taken by the resin cast, whereas a black pixel denotes an empty space.



(b) The green pixel is identified as having six neighbors, and thus is marked for deletion.



(c) The result after hollowing is complete. Note that pixels on the edge of the domain are assumed to have a neighbor outside the domain which prevents formation of "caps" on the ends.

Figure 7.1: The hollowing process removes internal pixels to provide a more natural seeding process for endothelial cells.

when they begin to move, the pipes they generate are correctly connected to the original graph.

7.4 Limitations of image based methods

It is incorrect to assume that the structure of vasculature is so regular; *in vivo* the cells which form the vessel walls are not axially aligned, nor entirely uniform in size. These assumptions, however, make the model vastly simpler to create and interpret. A future improvement would be to work on a much smaller grid, perhaps 10 times higher resolution, such that an endothelial cell takes up a larger space in terms of grid locations, but can move in a smoother way, or perhaps to take it a step further through the use of a continuous space for such movement. Currently, the model works on the Anderson and Chaplain method of checking a probability of movement based on how much VEGF has been absorbed, along with the local gradient of growth factor, and the contents of surrounding grid locations.

The images used in this research were MicroCT scanned images of resin cast rat cerebral vasculature. The resin cast is created by injecting liquid resin into the blood vessels of terminally anaesthetised rats, which is then allowed to set. Once the resin has set, the remainder of the tissue is dissolved with acid to expose the cast.

This allows the inspection of the vascular structures without the surrounding tissue, which means that scans become much more binary in nature - either a location is filled with a section of vasculature, or nothing at all. This, of course, fits in nicely with the prescribed initial conditions of the simulation, such that either a location is populated by an endothelial cell, or it is empty.

The resin casts were scanned in a SkyScan1174 compact microCT X-Ray scanner in which rotational X-Ray images are taken at an interval of 0.5 degrees. These images are then reconstructed using an implementation of the Feldkamp algorithm [61] which results in a new set of images corresponding to horizontal slices. These slices, when laid on top of one another in order, represent the entire volume in three-dimensional space.

Of course, we cannot use this method to acquire images of living patients. Typically, internal images of humans are taken using MRI scanners. In contrast to a MicroCT scanner, which typically has a functional resolution in the range of microns, the resolution of an MRI scanner is usually in the range of millimetres. This scan resolution is effectively limited by the strength of the magnetic field generated by the scanner.

Chapter 8

Modular Analysis on the Cloud

8.1 3DNet

The 3DNet Medical cloud system was developed by Biotronics3D and was designed around the idea of moving the functionality of a medical imaging software package to a cloud-based system, removing the need for users to acquire dedicated hardware. Medical imaging requires a considerable amount of power, and as such makes an ideal candidate for cloud computing. Typically, medical imaging solutions are high-end applications which require expensive hardware, and at times similarly expensive software. Thus, removing the hardware component by moving the functionality from local machines and into a cloud solution is an appealing prospect as it will greatly reduce the cost.

As a way of ensuring the relevance of the 3DNet Medical cloud system, it has to be capable of expanding with new technologies and techniques. To this end, a modular system for integrating new functions was devised. Modular frameworks have been in use for some time, often termed *plugin* systems, and can be seen in

a number of computer applications.

The cloud system described in chapter 3 forms the basis on top of which the module integration framework was built. The framework is composed of three separate components which communicate to each other using messages passed via web service and DICOM association. The following sections provide an overview of the interactions between components, followed by detailed descriptions each component.

8.2 A flexible pipeline for medical image analysis

We developed a flexible pipeline for integrating analysis modules into the 3DNet Medical system in the early stages of the project. This pipeline allows for the generation of rule sets to identify incoming scans which fit the criteria for a particular module, and can be automatically submitted to the analysis subsystem for processing. Studies can also be manually passed to the analysis subsystem at any given time, through a UI option integrated into the main system. The framework design is specifically tailored towards easy integration of a wide range of code modules, essentially anything which can be compiled to an executable or can exist as a standalone DICOM node. The importance of such a scheme is that it makes creation of novel analysis modules more attractive to developers who would not normally have the resources to develop an entire medical imaging solution.

In the 3DNet Medical cloud, the analysis subsystem is run on a separate server, which ensures a dedicated set of resources for analysis tasks. The overall system is divided into four separate components, which act on the data in turn. Each module is configured by the addition of an XML file describing the module, in

a specific directory the analysis server scans periodically for changes. Any new files are parsed by the server, and the details added to an internal list of available modules.

Data is passed to the analysis server and back via an exploitation of the DICOM standard which allows the inclusion of metadata in unique tags. DICOM compliant applications are allowed to only add private tags, and as such the server must scan incoming files for the relevant tags, identified by a unique string as per the requirements of the DICOM standard. A tag block then contains information regarding the size and type of the file being transferred. The file itself is embedded in the end of the DICOM carrier file, in the pixel data element - it is important to note that the pixel data has no requirement to be only the size specified by the DICOM header, and thus extra information can be safely added to the end of the file.

In the current form, the Analytic Module Manager (AMM) queues incoming studies for analysis, using a first-in-first-out scheme, and releases them to the individual analysis modules one by one (figure 8.3). Under this scheme, each module can be executing only once, initially, although multiple different analysis modules can be executing simultaneously. However, this can be scaled up to allow for multiple instances of a single analysis module to be executing in tandem. Return values from complete analysis modules are caught by the AMM. Depending on the return value received, the AMM can choose to run a module again, if there were problems, discard data after a given number of failed attempts, or return successful results to the main system for storage.

8.2.1 Event-driven workflow

Liu et al [62] put forward a framework for an event-driven workflow management system, which included elements of both automatic and user-interactive image processing [62]. Their system includes management of the image database including standard image viewing and editing capabilities, and thus was somewhat beyond the scope of this project. An event-driven system is a system of components which interact with each other using messages. This method of interaction through message passing is key in the design of such a system, in that it means that any component can run independently, without the need for the other components to be available. It implies a robust structure which allows elements to continue to function in the event of a partial system failure, a desirable aspect in a system distributed across multiple machines or sites.

The module integration framework was split into three distinct elements: the Module Wrapper, which initiates execution of the module (Section 8.2.4), the Analytic Module Manager, which controls data transfer between modules and the main system (Section 8.2.3), and the Logic Engine, which defines when to execute a module (Section 8.2.2). The flow of data between these components is shown in figure 8.1.

The data flow can be summarized as follows:

1. Incoming images are checked against predefined rule sets for eligibility.

2. The Module Manager is informed of eligible images, and queues them for processing.

3. The original images are stored as normal.

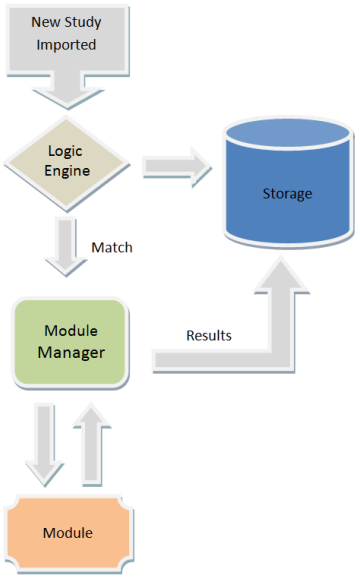


Figure 8.1: Data are checked against the Logic Engine upon import, and matches forwarded to external modules for additional processing. The original data are inserted into the system whether or not they are flagged for processing.

4. Eligible images are processed by the module binary.

5. Results are placed into storage.

8.2.2 Logic engine

Analysis modules can differ in operation for a variety of reasons, including the modality of the scanned image required (for instance, CT, MRI, PET, etc.), number of images required (single image, volume stack, temporal series), and body part scanned, to name but a few. It is necessary, therefore, that the import system be able to differentiate between when a module should be executed. A Logic Engine is required to perform checks on incoming data in order to decide whether or not the images meet requirements for additional processing.

A tiered approach is required to deal with availability of modules, creation and

administering of rule sets (referred to as module protocols), and activation of rule sets. Rule sets are organized in a logical tree on a per-user basis in the following configuration:

- User
    - Module
        - Protocol
            - Rule

Additionally, command line parameters can be defined on a per-protocol basis; these are covered in section 8.2.4.

Rules are of the format:

(Level) Tag Qualifier Value

where *Level* refers to whether the information is on a study or series level, *Tag* is the DICOM header tag to check (for instance, description, modality), *Qualifier* is the method of comparison (for instance equals, does not equal, contains), and *Value* is the particular value to check for. An example of a rule would be:

(Series) Modality Equals "CT"

Protocol evaluation is performed in two stages, first fetching, then checking. The fetch stage consists of retrieving active module protocols for the current user for modules which are available. Checking then evaluates each rule against data included in the DICOM headers of the files in question.

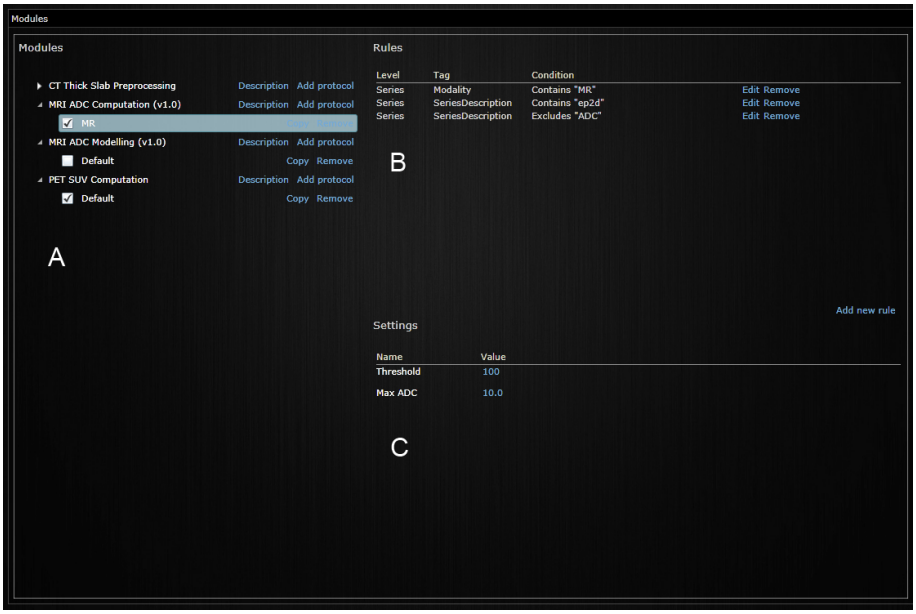


Figure 8.2: Users create protocols for modules (A) they are subscribed to. For each protocol a set of rules (B) is created, and any extra parameters (C) defined. When a set of rules is fully met by a series, the module is run with the given series and the associated parameters.

We identified two principle types of module, User Interface (UI) modules and Black Box (BB) modules. A User Interface module requires input from a user in the form of a sort of feedback loop, and can involve processing information multiple times, for instance where a user modifies parameters after running a module to try a new configuration. A Black Box module runs one time, and simply accepts a set of images and some parameters, and produces some results with no interaction. Placement of the logic engine is dependent on the type of module the system is required to support.

In the 3DNet implementation, the logic engine is integrated into two components of the system. To control execution of black box modules the logic engine works on the import pipeline. Thus, as new images are being imported they are

assessed by the logic engine and forwarded for extra processing if the rules are met. To control the execution of user interface modules the logic engine is integrated into the module launcher, which is in this case a context menu in the study browser.

Figure 8.2 shows the user interface developed to support creation and editing of module protocols in 3DNet. The interface automatically populates the list of modules (Figure 8.2, A) with all modules that the user or user's organization is subscribed to. The user is able to create, modify, and delete protocols for each module, for which they define a set of rules (Figure 8.2, B) which control when the module is run. If the module requires it, additional parameters can also be set per protocol (Figure 8.2, C).

8.2.3 Module management

Probably the largest component in the system, the Analytic Module Manager (AMM) was conceived to perform a controlling role in module execution and administration. The AMM acts as a gateway between the main system, which includes the logic engine, and the modules. As such, it must maintain a record of the connection details of every module.

Communication between the module manager and modules is performed over DICOM association, which allows the module manager to communicate with modules within the cloud and without. Therefore, the module manager requires knowledge of the AET (Application Entity Title), IP address, and port of all modules registered with the system. It is not necessary, however, that the module manager have knowledge of additional parameters that modules can accept. These

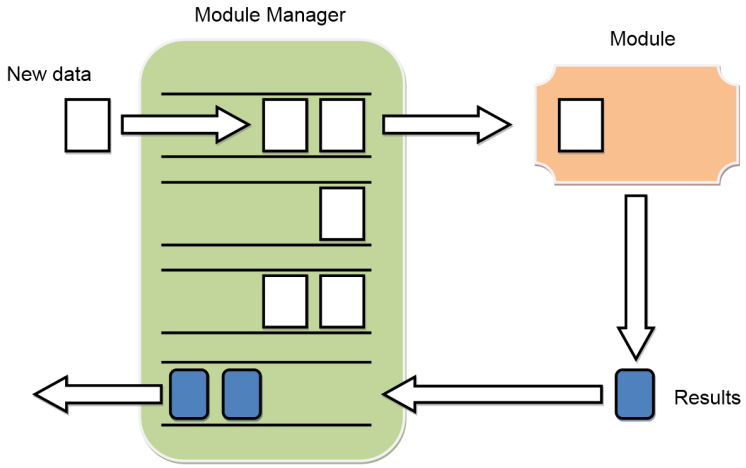


Figure 8.3: The module manager maintains a queue for each module and forwards data, where it is processed. Results are returned to the module manager, into a return queue, which inserts them into the main system.

are accepted as an extensible list structure, and placed in an XML file which is embedded in the first image to send, using the technique described in section 8.2.5.

Concurrent processing is a requirement of the system, and one of the leading factors in the design of the module manager. It is necessary that it be able to transmit and receive multiple data sets simultaneously; thus a multi-threaded approach is desirable. The design proposed uses a collection of queues, one per module (figure 8.3). Each module queue is governed by a separate thread, which controls administration and communication. This approach also takes advantage of multi-core and multi-processor architecture, where the operating system spreads process threads across processors.

The module manager maintains records of processing transactions, whether successful or not, in an attached database. These are to provide an audit trail,

and to allow for structured payment plans, as well as for bug tracking and fixing.

8.2.4 Module wrapper

Development methodologies can vary greatly between institutions, organizations, and individuals. As such, there is no standard development platform or method for creation of medical image analysis modules. Hence, provision of a unified communication interface between modules with the main management system becomes problematic.

In order to address this issue a wrapping element was devised upon which to mount analysis modules. Such a wrapper has the following requirements:

- Support multiple development languages. The wrapper should be able to mount modules which are produced in a variety of forms, such as executable or Java archive.
- DICOM connection support. The wrapper should provide DICOM connectivity for modules which do not support it natively. All communication with the module manager is performed over DICOM association.
- Customizable to be able to support different numbers and order of command line parameters.
- Scalable instancing to allow the module to be run multiple times simultaneously.
- Failure recovery. The module should be able to recover from errors and continue processing indefinitely.

A set of standard parameters are defined for analysis modules in order to make integration simpler. Analysis modules must be able to accept a minimum of three parameters which describe the location of the input data, a location in which to place DICOM format output data, and a location in which to place other output data. Additional parameters are also supported, and this is discussed in section 8.2.3. It is important to consider that a module may produce, or even require, data in a proprietary format, which is not supported by the DICOM standard. Section 8.2.5 discusses how binary file exchange was implemented over DICOM association.

Instancing is required to allow the module to concurrently process multiple data sets. The module wrapper maintains a queue for each instance, and distributes processing jobs according to queue sizes; that is, a new job is added to the shortest queue, so as to balance the load across all the queues.

Configuration of the module wrapper is achieved through a simple configuration in the form of an XML file. This file contains the DICOM connectivity details (application entity title, IP address, and port number), the path to the module executable (or jar, or dll), maximum time the module should take to run, the number of instances to create, and the order in which the three default parameters should be supplied (Listing 8.1).

Listing 8.1: Example contents of a module configuration xml file.

```
<config>
  <aet>MYMODULE</aet>
  <ip>127.0.0.1</ip>
  <port>804</port>
  <modulepath>C:\Modules\MyModule.exe</modulepath>
```

```
  <maxrunningtime hours="0" minutes="10" seconds="0"/>
  <initialinstances>1</initialinstances>
  <parameters input="0" outputdicom="1" outputcustom="2"/>
</config>
```

Failure recovery and redundancy In addressing failure recovery we have to consider two possibilities; first, that of a module error causing an infinite loop; second, that of a software failure which causes the module wrapper to exit unexpectedly. In the case of an infinite loop, the wrapper will have no way of knowing the module is stuck, hence the maximum running time argument in the configuration. This allows the wrapper to know when a module has been processing for too long, and the process can be terminated and logged as a failure. To deal with software failures (crashes) an extra layer of protection is in place in the form of a module installer service, run automatically on the host machine.

Robustness of service is required in modules being run using the module wrapper. This is achieved through a combination of installation of the wrapper as a Windows service, and an additional service called the Module Installer Service. The Module Installer Service (MIS) is a unique service which controls the running and installation of modules wrapped by the module wrapper. Because the module wrapper requires a command line argument to start correctly, installation and set up with a view to automatically starting the service is difficult when using the standard Windows tools. Hence, the MIS was created as a way of automatically starting instances of the module wrapper.

Installation of module services is simplified greatly by the inclusion of the MIS, which is designed to run continuously. The MIS uses a similar xml-based scheme as

the AMM, scanning a specific location for files which it will use to populate its list of modules. Each configuration file contains two pieces of information regarding the installation and running of the module, the service display name (how it will appear in the list of Windows services) and the path to the configuration file described above.

Upon initial execution, the MIS will read each configuration file and check for the relevant service. If a service with the given service name does not exist, the MIS will use the Windows tools to install the service as described in the configuration file, and start the service. If the service exists but is not running, the MIS will restart the service, providing some redundancy against possible failure. Finally, if a service exists and is running, the MIS will simply ignore the file. This process is repeated periodically which both ensures that the services remain running and that any additional modules added have a new service installed. While providing an essential part of the 3dnet Medical cloud, the MIS is not required to run on external hardware for obvious reasons.

8.2.5 Exploiting the DICOM standard for binary file transfer

Situations can occur where binary data transfer is required. Module results, and prerequisites, are not exclusively of the DICOM file format. It is not unusual for proprietary file formats to be used, and thus a file exchange method for these is required. The DICOM standard only supports transfer of DICOM format files, and pdf files encapsulated in a DICOM file. For reasons previously mentioned, it was desirable to make use of DICOM associations for data transfer.

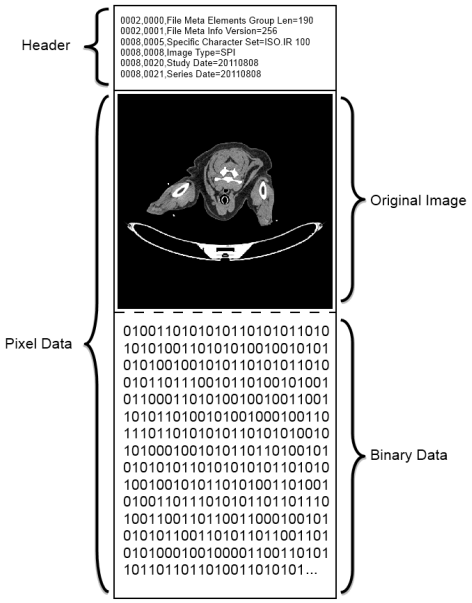


Figure 8.4: Binary data is placed after the original image in the pixel data tag. Information about the binary data (size, file name, type) is placed in the file header in the form of private tags.

To allow for binary file exchange over DICOM connection it was required that the format be exploited somewhat. The important aspect of the file format is that the pixel data is encoded last in a file. It is possible to add extra data to the end of a file as is required, and this can all be encoded in the pixel data tag (figure 8.4). This does not affect reading of the pixel data, as the pixel data itself remains unchanged.

These DICOM files then have a set of supporting private tags embedded to identify them as carrying extra binary data (and to identify the file by name, extension, and size); thus extraction is trivial. To accommodate the varying transfer syntaxes of different files, requiring that the pixel data be either OW (Other Word) or OB (Other Byte) value representation (VR), byte-padding is sometimes required. While the OB VR allows the use of bytes, the OW VR requires words

(in this case, pairs of bytes), and thus the number of bytes present is expected to be even. In this case, odd-numbered byte arrays are padded with one extra byte. This added byte has no effect on the pixel data stored using the OB VR, thus it is performed as a matter of course.

8.2.6 Inter-component communication

Each component in the system is a uniquely identified DICOM node. This allows for a single communication system, but also allows communication with external servers and systems which conform to the DICOM standard, and are open to accepting DICOM connections.

This allows the invocation of remote analysis modules. If a developer decides to host their analysis module on their own hardware, the AMM can be configured to pass data to it via DICOM networking protocols. Given the nature of the hosting cloud it is possible, and even likely, that extra computational resources will become available during the lifetime of the system, either in the form of upgraded hardware, or more likely in the form of additional servers within the cloud. The benefit of having such a flexible system is that it can take full advantage of the extra resources through such remote invocation schemes, without modification.

8.2.7 Individualised modelling in the cloud

The clinical relevance of the angiogenic process in human pathology makes an individualised simulation of angiogenesis an ideal candidate for an analysis module within the 3DNet Medical imaging system. As the goal of this work is to provide a basis for an element of personalised medicine, a cloud system like 3DNet Medical

provides a way of increasing accessibility to the technology, and availability to a large range of possible data sources.

In real terms, the hardware of the 3DNet cloud is inadequate to support a GPU-based analysis module for two reasons. First, the cost of even off the shelf graphics cards can be a large portion of setup costs, restricting choices. However, more importantly, a cloud needs to be as reliable as possible, and the life-span of graphics hardware is typically shorter than that of CPUs. However, the module system was designed with external as well as internal connections in mind, and so this could easily be hosted on an external server, wrapped in a DICOM connectivity layer - indeed, the MWS could happily run on an external server to allow communication with the AMM, and integration with 3dnet as a whole.

Chapter 9

Conclusions and Future Work

Probabilistic models of cell motion are an excellent method of simulating the growth of microvasculature under the given circumstances. The model of cell motion presented here describes cell motion as movement between grid cells (see figure 9.1), whereas previous efforts have limited motion to grid lines, such as the work of Anderson and Chaplain [3]. The decision to use grid cells instead of lines allows for cell motion across diagonal elements, which produces smoother transitions across chemical gradients, and more natural curves. This is further enhanced by the decision to limit the curvature of emerging vascular segments by preventing endothelial tip cells from doubling back upon themselves.

This approach has many degrees of flexibility. One interpretation of this could be that the model is uncertain, in need of refinement. However, limiting options for simulation will only limit the applications of a model. This model seeks to provide a flexible base for simulating a variety of angiogenic pathologies, enabling further customisation if required.

Building upon this, the decision to use scan data is an entirely new way of

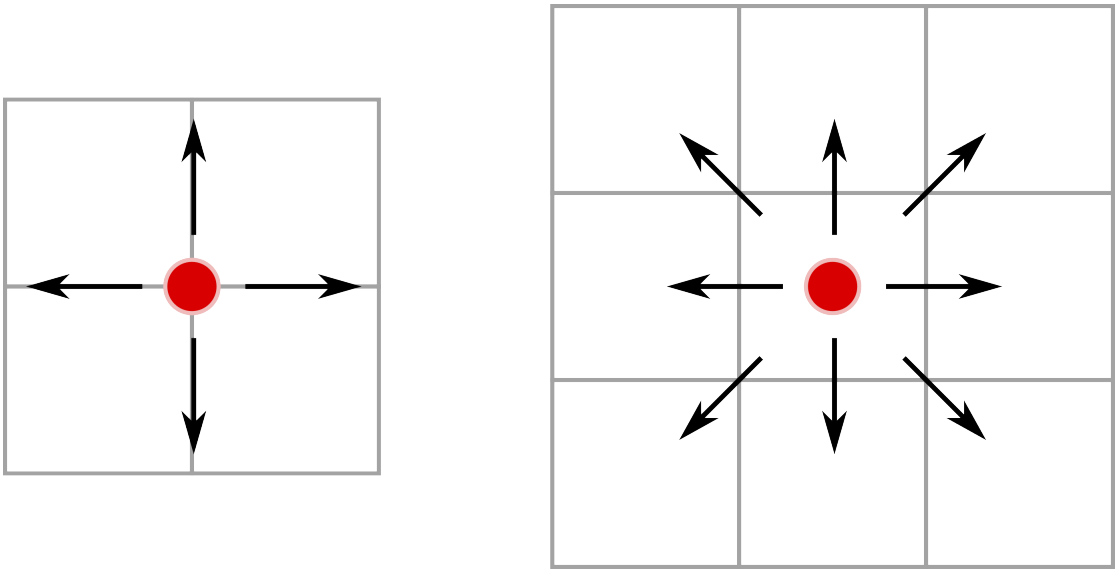


Figure 9.1: Previous models have limited motility of cells by constraining motion to grid lines (left). This model uses grid cells instead, adding an extra degree of freedom in the form of diagonal motion (right). In three dimensions, this increases the number of potential movement sites from eight to twenty-seven.

setting up an environment for angiogenic simulations. This technique allows the simulation of angiogenesis in a considerably more realistic environment than previous efforts. As opposed to other simulations which have focused on reproducing the results of *in vitro* models, this technique was designed with the aim of reproducing *in vivo* growth patterns *in silico*, which is key in closing the gap between simulations and reality.

The structure built from the scans is aligned along a regular grid, which appears naïve given the irregular, and in fact continuous, environment the body exists in, however this is an assumption made for two very important reasons. Regarding the motion of the endothelial cells themselves, this provides a computationally efficient way of organising the simulated motion and fits in neatly with other elements of the simulation. In respect to the images used and the physical structure of the

vascular network, the truth is that an exact cell-by-cell map of even a small piece of vasculature is virtually impossible to acquire. In this respect, the simulation makes an assumption about the regularity of cells in the endothelium, which is well documented, and what results is a best guess to describe the starting conditions of the vasculature, which in reality is not that far removed from the truth.

Due to the nature of the system, it being a three-dimensional environment, it takes large amounts of memory to set up the simulation environment. As such, it was a requirement at the time that the functional units were allocated in main memory. This could theoretically be moved into graphics memory given sufficient hardware capabilities. Current top-of-the-range graphics cards are available with as much as twelve gigabytes of memory, providing adequate capability for storage of the entire simulation environment.

Diffusion and fluid flow are both integral to the function of human vasculature. To obtain any level of realism, the fundamentals of the function of the vascular system need to be understood and correctly implemented. Incorporating elements such as these into a simulation of vessel growth and function is essential in achieving a degree of realism which provides and meaningful representation of the systems at work in the human body.

The graph-based method of flow calculations offers an efficient way of solving a complex system. A considerable sacrifice in terms of accuracy is the treatment of blood as a uniform fluid, which it is not. It is arguable that a more realistic solution should be used, however it is important to remember that the goal of this work is to find an acceptable balance between speed and realism. While, as discussed previously, a Lattice-Boltzmann approach offers much higher spacial resolution, the assumed increase in realism is lacking due to its treatment of blood as a uni-

Size of Grid	CPU	GPU
64	0.49s	0.16s
96	1.99s	0.63s
128	10.88s	1.527s
160	17.16s	3.01
192	37.87s	5.27s

Table 9.1: Comparing the execution times of a diffusion solver in CPU- and GPU-based solutions. The GPU produces markedly faster execution times, even when the number of cores available is exceeded.

form fluid. As such, the additional increased demand in power and time required offers little in the way of benefit. Conversely, systems which seek to simulate erythrocite movement within vessels will indeed offer more realistic results, however the increased complexity of such a system makes it undesirable here, especially considering the goal of increasing simulation sizes.

An issue which arises with this system is the need to regularly transfer data between graphics memory and main memory. Typically, in applications that make heavy use of graphics hardware, this is minimised wherever possible as it is a performance bottleneck.

When running like-for-like simulations on the GPU and on the CPU, it is easy to see that the GPU solution is preferable. Table 9.1 shows the difference in execution times for 100 iterations over a different sizes of grid of the diffusion algorithm. The increasing simulation times for the GPU may seem counter-intuitive, however this is due to a limitation in the hardware used. Three-dimensional grids of thread blocks are available in later versions of CUDA, however the hardware in use did not support this. To maintain the three-dimensional analogue, three-dimensional threadblocks were used in a two-dimensional grid. Thus, the simulation domain is split into thick slices, which are processed sequentially one section after another.

Due to the nature of the Gauss-Seidel method for solving the equation, this split becomes unimportant as it simply uses the latest values available.

When the concept of using scan data for instantiating the simulation environment was initially conceived, it was with the goal of mimicking the fine details of the scan data itself. The finest vessels would be removed, a result which tended to occur anyway through the use of thresholding, and simulations results would be directly compared to see if the same original vessel pattern was regrown. However, due to the stochastic nature of the simulation process, and early simulation results, it became apparent that this was not a feasible goal to have. A more appropriate method of comparison was thus sought.

To this end, a measure of tortuosity, or deviation from a straight path, was used. This can easily be measured by comparing the end to end length of a vessel segment, i.e. the straight-line distance between the start and end nodes, with the actual length of the vessel segment generated.

When analysing scan images, processing is in three stages: first, the images are converted to binary images; second the resulting binary images are reduced to a skeleton using a thinning system; finally the skeleton is analysed to find node points and vessel lengths. This analysis was performed using the free tool ImageJ. Analysis of generated vasculature was somewhat simpler, as it was stored in terms of geometric information, and not as images.

There will always be outliers in a simulation which is guided, at least in part, by random motion. Figure 9.3 compares the distributions of real scan data with simulated results: the best results were obtained using a chemotactic bias of 0.6.

9.1 Future work

This work seeks to provide a method for simulating a biological phenomenon with immediate clinical relevance in a realistic manner in real time. An important goal, therefore, was to find a balance between the requirements of realism, requirements in terms of power, and completing the simulation within a reasonable time frame.

The method presented here is entirely scalable, limited only by hardware factors, notably memory and processor speed. As such, there is no technical limit to the size of the simulation, however there are practical limits to how useful an enormous simulation could be. An angiogenesis simulation across the entire body, for example, would be of little use in studying the effect of a tumor site in one isolated location, as well as being infeasible in terms of memory, processor and graphics card requirements.

Current simulation sizes are between 1 - 2 millimeters cubed, which are a useful starting point for simulation efforts, but not of much use in terms of clinical relevance. The combination of parallel processing and CPU techniques is a useful way of spreading the load across available hardware, however, and bearing in mind current trends, increased simulation sizes are far from impossible. Running this simulation with a grid size of two millimetres per side takes approximately two point four gigabytes of memory. Of course, the memory requirements rise at a cubic rate, and can be obtained using an assumption of three hundred megabytes per cubic millimetre.

Even conventional home-user systems have gigabytes of memory available, and high-end workstation graphics cards offer as much as twelve gigabytes of video memory, so it is not outside the realms of possibility that such simulations will

soon become feasible for sizes measured in centimetres, not millimetres.

Simulating angiogenesis is not a new subject, although it is one of continuing research. However, this is the first time graphics hardware has been used to accelerate the simulation environment. The use of parallel processing in this context is key in balancing the resource requirements. By moving the mathematically heavy fluid dynamics equations to the graphics hardware, the CPU is freed to concentrate resources on the cellular motion and supporting functions, such as graph building.

Coding for the GPU and CPU uses two distinct models of programming. The parallel programming model as described in chapter 6 is not suited to the kind of individualised decision making process of a Cellular Potts Model. An object oriented approach to coding is much more flexible. The ability to create member variables for each object means that accessing personal data is much easier than with a parallel model where memory is shared and must be specifically indexed to be accessed. To emulate this kind of functionality on the GPU would require creating a buffer for each variable and populating via an index representing the identity of each "object". This has two distinct disadvantages.

Allocating memory dynamically during programme execution on the graphics card is undesirable, as it requires the reallocation of entire buffers, followed by copying memory, which is a costly procedure in terms of time. For the pressure system, which changes as the vascular network grows, buffers are doubled when more space is required in an effort to reduce the frequency of rebuffering data (up to a point - when the buffer exceeds a certain size, further rebuffers are of a fixed size to reduce the likelihood of running out of memory).

In addition to this, accessing data which might belong to another "object" could feasibly cause conflicts in access. In an object oriented model this is much safer

with the use of accessor methods which provide a safe way of retrieving information from other objects.

As an additional consideration, it is worth bearing in mind the increased development cost versus the potential gains of having an all-GPU solution. Doing so negates the benefits of having a centralised unit which distributes information if all subsystems reside on the graphics card, but the flexibility of such a modular system is lost.

Aside from size, a major barrier at present to the use of the techniques outlined here in a predictive capacity lies in the limitations of scanning technology available. The images used were microCT scans of resin casts, which provide a very good spacial resolution. Unfortunately, this technique is inappropriate for human use. Scanner technology is field of continuing research, however, and spacial resolutions similar to those of microCT scanners are sure to be available in less invasive methods (such as MRI) before too long.

9.2 Tissue engineering

The need for studying microvasculature is not limited to clinical diagnosis of diseases. A major difficulty in engineering tissues in vitro is engineering vascularised tissues. Any tissue that is more than a few hundred microns thick needs a proper vascular system because every cell in a tissue needs to be close enough to capillaries to absorb the oxygen and nutrients. This is why skin and cartilage were among the first to be ready for human testing, since they do not require extensive internal vasculature. An engineered tissue implant will have to connect quickly with the host vascular system. Creating appropriate environments/scaffolds to

promote vascular growth, especially those that mimic vasculatures in real tissues, and imaging of this growth in vivo are therefore also important aspects of tissue engineering. However, vasculatures engineered in vitro are at present governed by, at best, information gathered from the literature or, at worst, by uninformed trial and error. They lack the functional properties of real vasculatures, especially microvessels or capillary beds. Though there has been active research using mathematical models of angiogenesis to generate vascular structures , these cannot match the microvascular patterns in real tissues.

In their 2011 paper, Novosel et al. [7] stated that the key challenge in tissue engineering is vascularisation . Current methods for tissue engineering involve growing tissue in an oxygenated solution, in order to simulate perfusion by blood vessels. This, however, means the tissue is not grown with a network of blood vessels of its own. One application of simulations of angiogenesis would be to design blood vessel scaffolds which not only successfully perfuse a region of tissue, but also mimic the natural formation of vascular networks. An individual-based model for angiogenesis means that the network can be tailored to any domain specified. Recently, Miller et al [20] described a modified three-dimensional printing technique for creating pre-vascularised scaffolds for tissue engineering, which combined with a realistic generation method for vascular networks could aid the advancement of tissue engineering by allowing the growth of pre-vascularized structures.

9.3 Cloud computing

Future research will be to go back to where this PhD project started. The project was sponsored by an EPSRC Industrial Case Award, working with Biotronics3D

on medical imaging on the Cloud.

Continuing the work here naturally leads to the concept of developing the simulation in such a capacity as to be incorporated In as a module in the 3DNet Medical cloud. In developing this solution as a module for the cloud certain ending conditions need to be specified. These could, realistically, be temporal, i.e. setting a limit for simulation time, or cell-based, i.e. when there are no more cells eligible for migration the simulation is over. Providing both as optional parameters would give researchers a flexible way of ensuring that simulation times do not get out of hand.

There has been interest in the effectiveness of anti-angiogenic therapies in the treatment of cancer. Some work has been performed in this area which shows that anti-angiogenics, whilst decreasing the active blood vessel network around a tumour, impede the delivery of other treatments such as chemotherapy. This is a promising area of research, and an ideal candidate for future developments in this system. This would incorporate additional elements of the pathology, starting with the introduction of growth factor producing tumour cells, which would then grow based upon the delivery of oxygen through induced vessel growth, or demonstrating how this is impeded by the introduction of anti-angiogenic substances.

9.4 A more general scheme for modular development

Building on the current work, one could assume the goal of designing an entirely modular system and toolkit for the development of medical simulations. The work

here already describes a basis for how such a system can be achieved, combining both CPU and GPU techniques towards the creation of a single unified system.

While image processing and visualisation efforts have become unified over the years, there is still a considerable distance between medical simulation developers and researchers. The Biotronics3D solution is a step in the right direction, encouraging developers to share their work on a unified platform, but in order to improve research efforts in *simulation* such a system is still lacking. While, as previously discussed, visualisation and imaging toolkits are widely available, there is a distinct lack of any such framework for simulations, making this avenue ideal for future work.

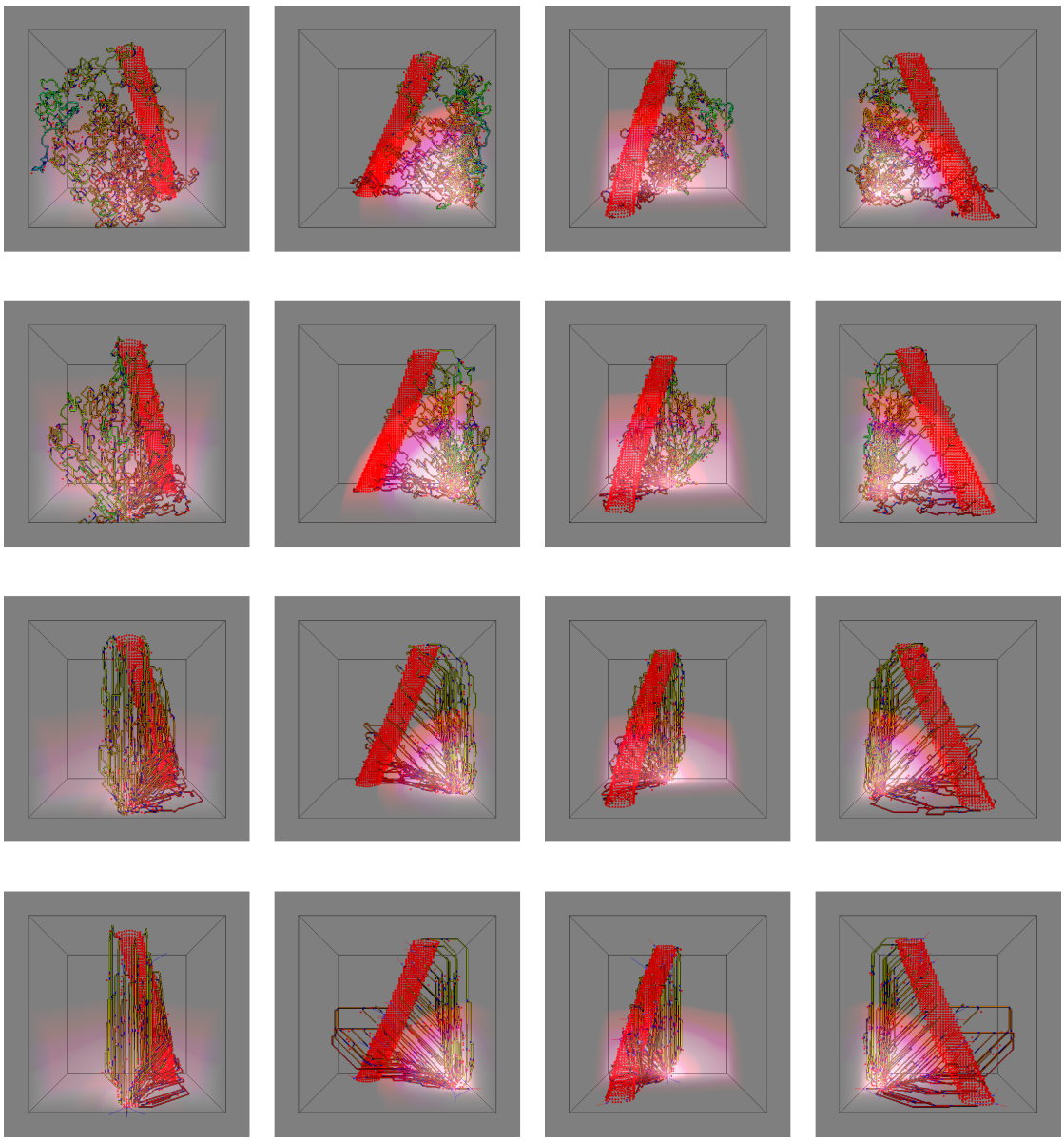


Figure 9.2: Increasing the chemotactic bias produces straighter vessels, whereas decreasing produces more random, tortuous results. Shown here are results with bias values of 0.2, 0.4, 0.6, and 0.8, in descending order. Identifying factors which affect the endothelial cell's response to the chemical gradient could be beneficial in early diagnosis of disease.

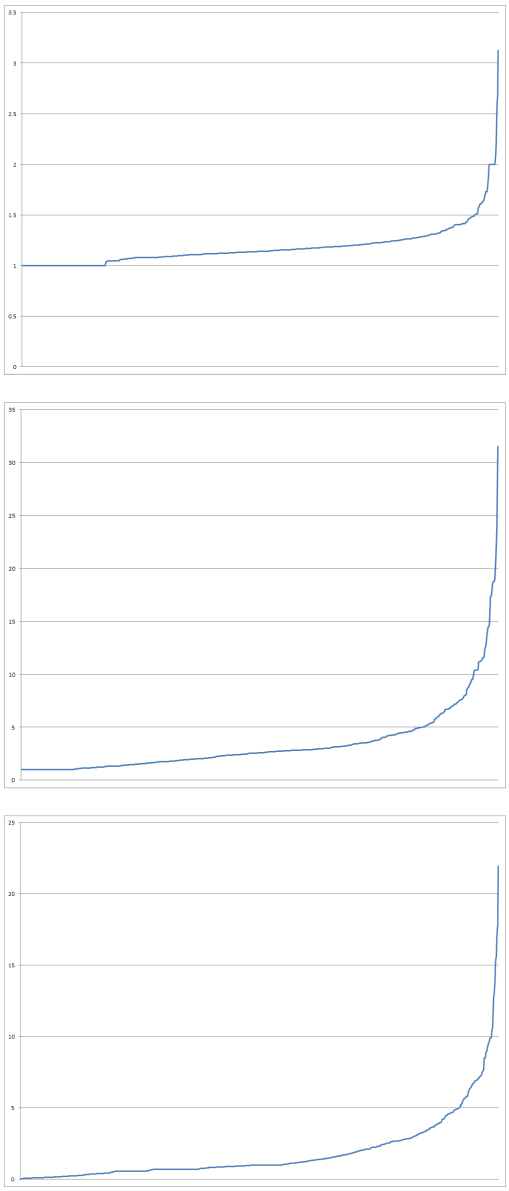


Figure 9.3: The distribution of actual length divided by end to end length. First is the distribution from scan data; second is using a low chemotactic bias; bottom uses a high chemotactic bias.

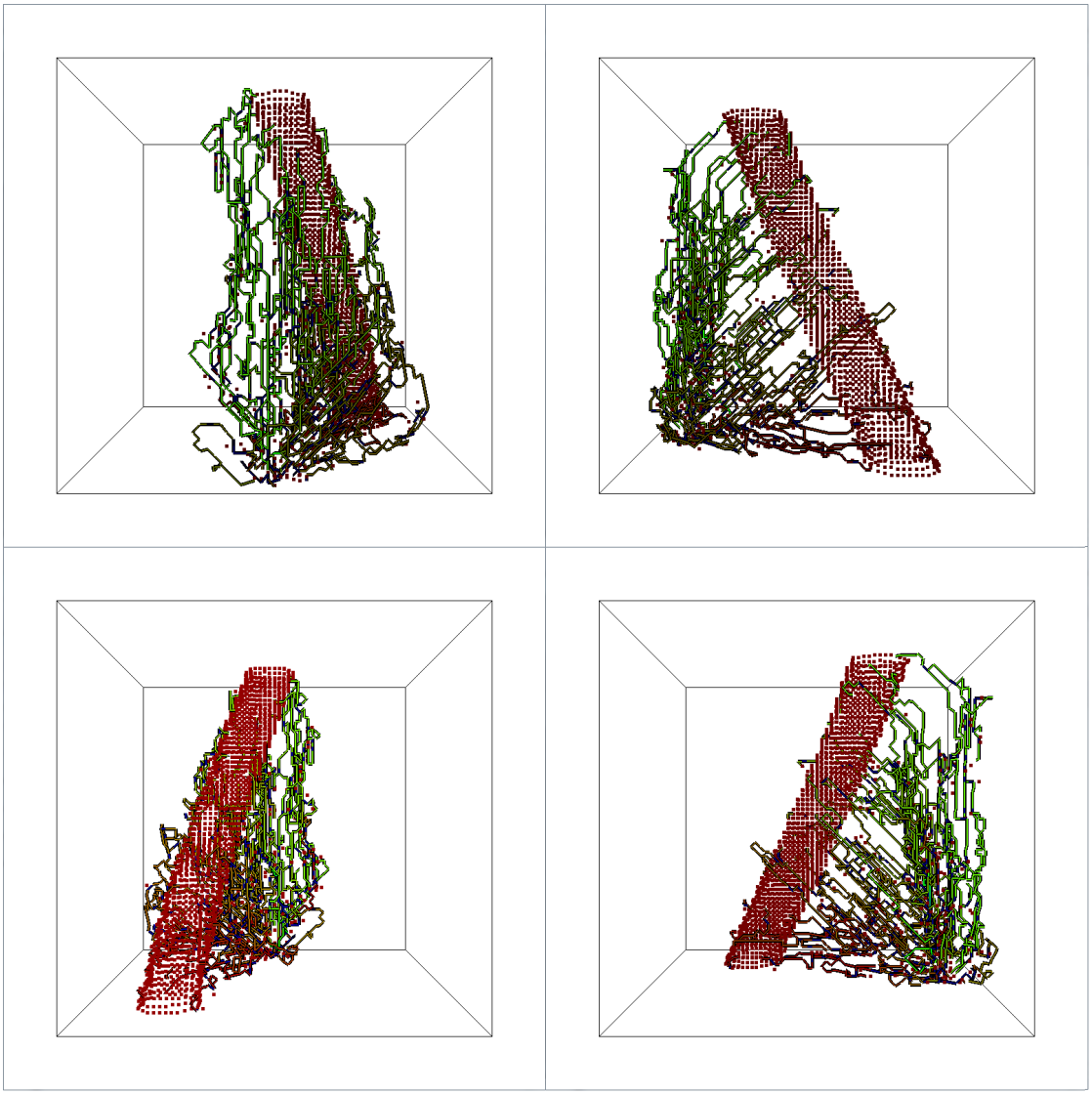


Figure 9.4: A complete simulation shown from four distinct view points. New vessels are coloured based on the blood pressure, solved using the algorithm described in 6.

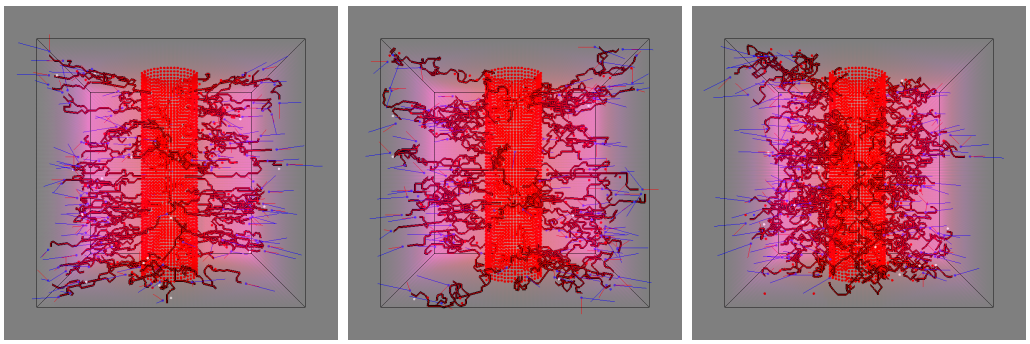


Figure 9.5: By creating a stack of images to reflect a manufactured tube and initialising a gradient of growth factor, this method can be used to fill an arbitrary space with vasculature. Changing the parameters (decreasing chemotactic bias from left to right) produces different densities of vessels.

Publications

A list of the papers published during the course of this project is give here.

Medical Imaging in a Cloud Computing Environment, L. Parsonson, L. Bai, L. Bourne, A. Bajwa, S. Grimm, Closer, 2011

3DNET - An ecosystem for the development, evaluation, and sharing of visualization workflows, S. Grimm, A. Paluszny, L. Parsonson, R.Andrian, W. Hernandez, L. Bourn, A. Bajwa, H. Hatzakis, L. Bai, WorldComp, 2012

Individual Based Modelling of Angiogenesis on the Graphics Processing Unit, L. Parsonson, L. Bai, Advances in Fluid Mechanics, Wessex Institute, 2014

3D Angiogenesis Modelling on the GPU, L. Parsonson, L. Bai, Accepted for publication in The 2014 7th International Conference on BioMedical Engineering and Informatics (BMEI 2014),

List of Figures

2.1 A Rapidly-exploring Random Tree 25

2.2 Deformation of red blood cells in capillaries 28

3.1 Ray casting through a volume 40

3.2 Volume renders with different transfer functions 41

3.3 The depth render 43

4.1 Converting scanned images to endothelial cells 45

4.2 A volume render of a scanned resin cast 46

4.3 Distances in three dimensions 47

4.4 Cell motion towards the source of growth factor 49

4.5 Differing growth factor distributions and the effect on cell motion . 50

4.6 The effect of different sized exclusion zones 51

4.7 A simulation of size 192^3 54

6.1 How threadblocks relate to the simulation volume 64

6.2 Relative apparent viscocity 68

6.3 A vessel bifurcation and preservation of flow 69

6.4 A comparison of a vessel network and it's graph representation . . . 71

6.5 An example graph for the pressure model 72
6.6 Pressure stabilised across a generated network of vessels 73
6.7 Introducing low pressure areas to the graph 74
6.10 Counting spaces around a vessel segment 79
6.8 Mass transfer across a network 84
6.9 Timeline of the growth of capillaries in the simulation 85

7.1 Hollwing imported vasculature 91

8.1 The flow of data being imported into the cloud 98
8.2 The user interface for creation and administration of module protocols 100
8.3 Queuing studies for processing via the AMM 102
8.4 Using DICOM files as a vehicle for binary data transfer 107

9.1 Degrees of freedom in cellular motion 111
9.2 The result of modifying the chemotactic bias 121
9.3 Distributions of vessel actual length divided by end to end length . 122
9.4 A completed simulation showing vessels and pressure values 123
9.5 Using the model to fill an arbitrary space 124

References

[1] Judah Folkman. Angiogenesis in cancer, vascular, rheumatoid and other disease. *Nat Med*, 1(1):27–30, 01 1995.

[2] Costantino Iadecola. Rescuing troubled vessels in alzheimer disease. *Nat Med*, 11(9):923–924, 09 2005.

[3] Alexander RA Anderson and MAJ Chaplain. Continuous and discrete mathematical models of tumor-induced angiogenesis. *Bulletin of mathematical biology*, 60(5):857–899, 1998.

[4] L Arakelyan, V Vainstein, and Z Agur. A computer algorithm describing the process of vessel formation and maturation, and its use for predicting the effects of anti-angiogenic and anti-maturation therapy on vascular tumor growth. *Angiogenesis*, 5(3):203–214, 2002.

[5] Markus R Owen, I Johanna Stamper, Munitta Muthana, Giles W Richardson, Jon Dobson, Claire E Lewis, and Helen M Byrne. Mathematical modeling predicts synergistic antitumor effects of combining a macrophage-based, hypoxia-targeted gene therapy with chemotherapy. *Cancer research*, 71(8):2826–2837, 2011.

[6] François Graner and James A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Physical Review Letters*, 69(13):2013–2016, 09 1992.

[7] Esther C. Novosel, Claudia Kleinhans, and Petra J. Kluger. Vascularization is the key challenge in tissue engineering. *Advanced Drug Delivery Reviews*, 63(4–5):300 – 311, 2011.

[8] Talia N Crawford, III Alfaro, John B Kerrison, and Eric P Jablon. Diabetic retinopathy and angiogenesis. *Current diabetes reviews*, 5(1):8–13, 2009.

[9] Rohit Khurana, Michael Simons, John F Martin, and Ian C Zachary. Role of angiogenesis in cardiovascular disease a critical appraisal. *Circulation*, 112(12):1813–1824, 2005.

[10] Anthony H Vagnucci and William W Li. Alzheimer's disease and angiogenesis. *The Lancet*, 361(9357):605–608, 2003.

[11] Judah Folkman and Michael Klagsbrun. Angiogenic factors. *Science*, 235(4787):442–447, 1987.

[12] Judith H Harmey and David Bouchier-Hayes. Vascular endothelial growth factor (vegf), a survival factor for tumour cells: Implications for anti-angiogenic therapy. *Bioessays*, 24(3):280–283, 2002.

[13] N. Howlader, A. M. Noone, M. Krapcho, J. Garshell, N. Neyman, S. F. Altekruse, C. L. Kosary, M. Yu, J. Ruhl, Z. Tatalovich, Cho, A. Mariotto, D. R. Lewis, : Feuer E. J. Chen, H. S., K. A. Cronin, and (eds). Seer cancer statistics review, 1975-2010. Technical report, Bethesda, MD, 2013.

[14] Ruman Rahman, Stuart Smith, Cheryl Rahman, and Richard Grundy. Antiangiogenic therapy and mechanisms of tumor resistance in malignant glioma. *Journal of oncology*, 2010, 2010.

[15] Rakesh K Jain. Normalization of tumor vasculature: an emerging concept in antiangiogenic therapy. *Science*, 307(5706):58–62, 2005.

[16] Gabriele Bergers and Douglas Hanahan. Modes of resistance to anti-angiogenic therapy. *Nature Reviews Cancer*, 8(8):592–603, 2008.

[17] Urszula Ledzewicz, John Marriott, Helmut Maurer, and Heinz Schättler. Realizable protocols for optimal administration of drugs in mathematical models for anti-angiogenic treatment. *Mathematical Medicine and Biology*, 27(2):157–179, 2010.

[18] Marianne O. Stefanini, Amina A. Qutub, Feilim Mac Gabhann, and Aleksander S. Popel. Computational models of vegf-associated angiogenic processes in cancer. *Mathematical Medicine and Biology*, 29(1):85–94, 03 2012.

[19] Sébastien Benzekry, Guillemette Chapuisat, Joseph Ciccolini, Alice Erlinger, and Florence Hubert. A new mathematical model for optimizing the combination between antiangiogenic and cytotoxic drugs in oncology. *Comptes Rendus Mathematique*, 350(1):23–28, 2012.

[20] Jordan S Miller, Kelly R Stevens, Michael T Yang, Brendon M Baker, Duc-Huy T Nguyen, Daniel M Cohen, Esteban Toro, Alice A Chen, Peter A Galie, and Xiang Yu. Rapid casting of patterned vascular networks for perfusable engineered three-dimensional tissues. *Nature materials*, 11(9):768–774, 2012.

[21] William Lafayette Mondy, Don Cameron, Jean-Pierre Timmermans, Nora De Clerck, Alexander Sasov, Christophe Casteleyn, and Les A Piegl. Computer-aided design of microvasculature systems for use in vascular scaffold production. *Biofabrication*, 1(3):035002, 2009.

[22] Volodymyr V Kindratenko, Jeremy J Enos, Guochun Shi, Michael T Showerman, Galen W Arnold, John E Stone, James C Phillips, and Wen-mei Hwu. Gpu clusters for high-performance computing. pages 1–8. IEEE, 2009.

[23] Sli best practices, February 2011.

[24] Rui D M Travasso, Eugenia Corvera Poiré, Mario Castro, Juan Carlos Rodrguez-Manzaneque, and A Hernández-Machado. Tumor angiogenesis and vascular patterning: A mathematical model. *PLoS ONE*, 6(5):e19989, 2011.

[25] MG Watson, SR McDougall, MAJ Chaplain, AH Devlin, and CA Mitchell. Dynamics of angiogenesis during murine retinal development: a coupled in vivo and in silico study. *Journal of The Royal Society Interface*, page rsif20120067, 2012.

[26] Luigi Preziosi and Sergey Astanin. *Modelling the formation of capillaries*, pages 109–145. Springer, 2006.

[27] Andreas Linninger and Nicholas Vaicaitis. Computational modeling of cerebral vasculature. 2011.

[28] S Moore and T David. A model of autoregulated blood flow in the cerebral vasculature. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 222(4):513–530, 2008.

[29] Holger Perfahl, Helen M Byrne, Tingan Chen, Veronica Estrella, Tomás Alarcón, Alexei Lapin, Robert A Gatenby, Robert J Gillies, Mark C Lloyd, and Philip K Maini. Multiscale modelling of vascular tumour growth in 3d: the roles of domain size and boundary conditions. *PloS one*, 6(4):e14790, 2011.

[30] Ralf Gödde and Haymo Kurz. Structural and biophysical simulation of angiogenesis and vascular remodeling. *Developmental Dynamics*, 220(4):387–401, 2001.

[31] Markus R Owen, Tomás Alarcón, Philip K Maini, and Helen M Byrne. Angiogenesis and vascular remodelling in normal and cancerous tissues. *Journal of mathematical biology*, 58(4-5):689–721, 2009.

[32] John Conway. The game of life. *Scientific American*, 223(4):4, 1970.

[33] Martin Gardner. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223(4):120–123, 1970.

[34] Paweł Topa. *Towards a two-scale cellular automata model of tumour-induced angiogenesis*, pages 337–346. Springer, 2006.

[35] Haymo Kurz, Peter H Burri, and Valentin G Djonov. Angiogenesis and vascular remodeling by intussusception: from form to function. *Physiology*, 18(2):65–70, 2003.

[36] James J Kuffner and Steven M LaValle. Space-filling trees. *RI, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-09-47*, 2009.

[37] Steven M LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

[38] Herbert H Lipowsky. Microvascular rheology and hemodynamics. *Microcirculation*, 12(1):5–15, 2005.

[39] Timothy W Secomb. Mechanics of blood flow in capillaries. 2009.

[40] Robin Fåhræus and Torsten and Lindqvist. The viscosity of the blood in narrow capillary tubes. *American Journal of Physiology – Legacy Content*, 96(3):562–568, 03 1931.

[41] AR Pries, A Fritzsche, K Ley, and P Gaehtgens. Redistribution of red blood cell flow in microcirculatory networks by hemodilution. *Circulation research*, 70(6):1113–1121, 1992.

[42] ACR-NEMA. Digital imaging and communications in medicine, 2014.

[43] Kitware. Insight segmentation and registration toolkit, 2011.

[44] Kitware. Visualization toolkit, 2014.

[45] Ivo Wolf, Marcus Vetter, Ingmar Wegner, Thomas Böttger, Marco Nolden, Max Schöbinger, Mark Hastenteufel, Tobias Kunert, and Hans-Peter Meinzer. The medical imaging interaction toolkit. *Medical Image Analysis*, 9(6):594–604, 2015/02/08.

[46] Jens Kruger and Rüdiger Westermann. Acceleration techniques for gpu-based volume rendering. page 38. IEEE Computer Society, 2003.

[47] G. Lippold. Mitchell, a. r./griffiths, d. f., the finite difference method in partial differential equations. chichester-new york-brisbane-toronto, john wiley & sons 1980 xii, 272 s., £8.95. isbn 0-471-27641-3. *ZAMM - Journal of Applied*

Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 60(12):741–741, 1980.

[48] Greg Lemon, Daniel Howard, Felicity RAJ Rose, and John R King. Individual-based modelling of angiogenesis inside three-dimensional porous biomaterials. *Biosystems*, 103(3):372–383, 2011.

[49] Harold F Dvorak, Lawrence F Brown, Michael Detmar, and Ann M Dvorak. Vascular permeability factor/vascular endothelial growth factor, microvascular hyperpermeability, and angiogenesis. *The American journal of pathology*, 146(5):1029, 1995.

[50] Rashid Mehmood and Jon Crowcroft. Parallel iterative solution method for large sparse linear equation systems. *Computer Laboratory: University of Cambridge*, 2005.

[51] NVidia. Nvidia quadro processor specification, 2014.

[52] AR Pries, TW Secomb, and P Gaehtgens. Biophysical aspects of blood flow in the microvasculature. *Cardiovascular research*, 32(4):654–667, 1996.

[53] Angelique Stephanou, Steven R McDougall, Alexander RA Anderson, and Mark AJ Chaplain. Mathematical modelling of flow in 2d and 3d vascular networks: applications to anti-angiogenic and chemotherapeutic drug strategies. *Mathematical and Computer Modelling*, 41(10):1137–1156, 2005.

[54] Qianqian Fang, Sava Sakadzic, Lana Ruvinskaya, Anna Devor, Anders M Dale, and David A Boas. Oxygen advection and diffusion in a three-

dimensional vascular anatomical network. *Optics express*, 16(22):17530–17541, 2008.

[55] Mark S McAllister, Ljiljana Krizanac-Bengez, Francesco Macchia, Richard J Naftalin, Kevin C Pedley, Marc R Mayberg, Matteo Marroni, Susan Leaman, Kathe A Stanness, and Damir Janigro. Mechanisms of glucose transport at the blood–brain barrier: an in vitro study. *Brain research*, 904(1):20–30, 2001.

[56] Sean HJ Kim, Sunwoo Park, Amina A Qutub, and C Anthony Hunt. In silico modeling of blood-brain barrier: Agent-based simulation of cerebral glucose transport. pages 2–8, 2005.

[57] N Joan Abbott, Lars Rönnbäck, and Elisabeth Hansson. Astrocyte–endothelial interactions at the blood–brain barrier. *Nature Reviews Neuroscience*, 7(1):41–53, 2006.

[58] Louis Parsonson and Li Bai. Three-dimensional angiogenesis modelling on the gpu. pages 1–6. IEEE, 2014.

[59] Amina A Qutub, Feilim Mac Gabhann, Emmanouil D Karagiannis, Prakash Vempati, and Aleksander S Popel. Multiscale models of angiogenesis. *Engineering in Medicine and Biology Magazine, IEEE*, 28(2):14–31, 2009.

[60] Wilhelm Conrad Röntgen. On a new kind of rays. *Science*, pages 227–231, 1896.

[61] Skyscan NV. Nrecon user manual, April 2011.

[62] L Liu, D Meier, M Polgar-Turcsanyi, P Karkocha, R Bakshi, and C Gutterman. Event-driven workflow management for medical image processing and

analysis in a large image database. *Medical Image Computing and Computer Assisted Intervention Society*, 2004.