Bachelor Thesis

The Interface Group

1D Computations of Flow and Oxygen Transport in Micro-Vascular Networks

Berk Nergiz

Advisers
Dr. Kartik Jain
Prof. Dr. Vartan Kurtcuoglu
Interface Group, Institute of Physiology (UZH)
Supervision
Prof. Dr. Bradley J. Nelson
Institute of Robotics and Intelligent Systems
Swiss Federal Institute of Technology Zurich (ETH)

July 2018

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D MAVT

Contents

Abstract

Oxygen is carried by red blood cells in the blood. It is supplied to organs, muscles and generally to the tissue through a complex network of blood vessels that range from milli- to micro- meter in size. This microvascular network consists of micro-sized capillaries, that can be extracted from ex vivo mice organs. This work is performed in the context of the NCCR Kidney.CH project, which aims to develop computational models for the computation of gradients of oxygen in the renal tissue, and associate them with activity of cells for better understanding of kidney physiology. The specific goals of the thesis are to study oxygen transport and pressure gradients in capillary networks using computational methods.

1 Introduction

1.1 Motivation

This thesis aims to understand and specify how blood flow and oxygen transport has been simulated until today. The goal is to see how far existing methods, especially the Green's function method applied to this field by Secomb et al. [15] can produce accurate results for new and especially complex networks. A new model using an open source software called $DuMu^x$ [1] was implemented to offer a different approach for oxygen transport and diffusion simulations.

Oxygen from blood is carried on to the tissue of living organs through a complex network of vessels, in particular by capillaries that provide oxygen to the tissue, which is attached to the red blood cells. The complex biochemistry involved in the release of oxygen from hemoglobin and its transport to the surrounding tissue is a complex process that cannot be fully understood by experimental methods. With the advent of mathematical models and advanced computing architectures, the process of oxygen delivery to the tissue can be simulated to understand various physiological and pathophysiological processes.

This thesis aims to simulate transport of oxygen through capillary networks of various organs using mathematical models. Specifically, the goal is to evaluate the efficacy of the existing numerical methods that simulate flow and transport of O_2 through capillaries. The Green's function method developed by Secomb is evaluated in detail and applied to various microvascular networks. Further, coupling of 1D network with a 3D surrounding tissue is evaluated in the open source package called $DuMu^x$. A simplified approach for the simulation of O_2 transport was implemented in $DuMu^x$ and compared with the results of Green's function method.

1.2 Structure of the Thesis

First, a short insight to the physiological background of blood flow and oxygen transport in the body will be given. The general process of oxygen delivery to the tissue and oxygen consumption by the tissue will be explained. The main subject will be the transport of oxygen in the body, starting in the blood flow and ending in the diffusion into the tissue through a general diffusion process.
The results of oxygen transport simulation methods, especially the Green's function method, will later be discussed and evaluated for a range of different vascular networks from different organs. In addition to this, a new model that was implemented in the open source framework $DuMu^x$ for the computation of oxygen transport through such vessels will be discussed. The focus of this thesis is to understand and evaluate the Green's function method for computations of oxygen gradients in various organs and a range of morphologies of networks.

2 Blood Flow and Oxygen Delivery

In this section the physiological aspects of O_2 transport through vessels into the tissue will be discussed. This part is mainly dedicated to the understanding of physical processes that are involved in the transport of oxygen through capillaries into the tissue, and its consumption therein.
To give the reader a short overview, I will start at the beginning and give a brief explanation of how blood travels through the body. How it is pumped through the heart into the whole body, carrying well oxygenated fresh blood from the heart to the tissues everywhere in the body, and then back from the tissues to the heart through the veins.
In this thesis, the flow and transport through micro-vessels in the capillary range and in the tissue is the main concern, where generally the time-dependency of the blood-velocity will not be taken into account. The periodic beating of the heart causes the velocity to be inconstant, but this effect will be neglected for modeling purposes and computations in our case. For this, we will use a steady-state approximation for the flow rate, and thus assume a constant velocity. This means that there is no variation of velocity depending on time for any given point in the network.

2.1 Blood Flow

Blood is basically a mixture of two components: liquid and cells. The liquid phase is called plasma, and the cells that play the most important role for our concern are the red blood cells. This will later be important when considering the oxygen transport in the blood. For computational models, blood is generally modeled as an incompressible Newtonian fluid [3]. This basically means that the viscous stress is linearly proportional to the strain rate. Modeling blood as a Newtonian fluid is basically a wrong assumption, due to the fact that only the plasma is a Newtonian fluid. Nevertheless, this model of blood is used a lot as it can produce fairly good results and for the fact that taking irrelevant details into account wouldn't be very interesting. However, for oxygen transport simulations, modeling blood as a Newtonian fluid is problematic, due to the fact that the red blood cells are carrying around 99% of the oxygen in the blood. It is also important to mention that the cells, making up to $40 - 45\%$ of the blood volume, are basically not incompressible. They can be compressed and might even shrink in order to facilitate the blood flow in small vessels.
While this model of blood uses big assumptions and turns a two component medium into an incompressible Newtonian fluid, this immensely facilitates computations from an engineering point of view and can produce fairly good results.

The Green's function method code, however, takes everything into account, namely the discharge of oxygen from hemoglobin and the presence of red blood cells, which are the

main oxygen carriers. One of our goals was to understand the influence of ignoring the fact that not blood in general, but only red blood cells are carrying the oxygen. For this, we searched for a way to account for these differences by other modeling, in particular using the $DuMu^x$ based model I will explain in detail in the next sections.

However, modeling blood as a Newtonian fluid leads to the Navier-Stokes equations 2.1, 2.2, which basically describe the flow dynamics of an incompressible Newtonian fluid:

$$\rho\frac{\delta\mathbf{v}}{\delta t} + \rho\mathbf{v}\nabla\mathbf{v} + \nabla p - \eta\nabla(\nabla\mathbf{v} + \nabla\mathbf{v}^T) = \mathbf{f} \tag{2.1}$$

$$\nabla\mathbf{v} = 0 \tag{2.2}$$

In the Navier-Stokes equations, $\mathbf{v}$ is the velocity vector, p is the pressure, and ρ and η are the blood density and viscosity respectively.
The external forces acting on the fluid are represented by the vector $\mathbf{f} = (f_x f_y f_z)^T$.
The first and the second term on the left-hand side are corresponding to inertial forces, while the third term stands for the pressure force and the fourth term for the viscous force. The sum of these forces must be equal to the right-hand side, where we only have the external forces $\mathbf{f}$.
The next step in order to solve these equations for the blood flow in a given vessel is to put appropriate initial and boundary conditions. An example of useful boundary conditions is given here:

$$\mathbf{v}_{in} = \mathbf{v}_0 \tag{2.3}$$

$$\mathbf{v}_{vesselwall} = 0 \tag{2.4}$$

Here we put the velocity $\mathbf{v}_{in}$ of the fluid entering our domain equal to a velocity vector $\mathbf{v}_0$ and prescribe a non-slip condition on the vessel walls. Solving the Navier-Stokes equations 2.1, 2.2 using appropriate initial conditions and making a few other assumptions will then lead to the Poiseuille-Flow which can generally be used to describe Newtonian fluid flows in pipes.

2.2 Oxygen Transport and Delivery

Through breathing, oxygen enters the body and fills the lungs. As gases tend to move from areas with higher concentrations towards areas with lower concentrations, oxygen moves from the oxygen rich air towards the alveoli and enters the blood [11]. This same concentration gradient will later transport the oxygen into the tissue during the diffusion process [13]. The physical processes behind this will briefly be explained in this section.

Oxygen Transport in the Blood

As mentioned in the previous chapter, blood is composed of plasma and red blood cells. The red blood cells are the main carriers of oxygen in the blood [13]. In this section, we will mainly focus on the red blood cells, as these are the main oxygen carriers. As the general concern of this thesis is not the physiological oxygen-red blood cell interaction, I will just briefly explain the basic principle behind oxygen transport in the blood.
The oxygen carrier contained in the red blood cells is called hemoglobin. Hemoglobin is a protein which can chemically bind oxygen molecules. This means that while our blood travels through the vessels, oxygen is supplied to every cell in the body by the hemoglobin which binds oxygen and transports it with the blood flow [7].
In a very simplified way, one could see red blood cells as train driving in a closed loop circuit, picking up oxygen at one station, and then delivering it to every station in the body, which are the tissues and accordingly the cells. The hemoglobin would be the seat where oxygen sits on, as it has the role of binding the incoming oxygen from the lungs and release it to tissue so that it can diffuse out of the vessels and reach the cells.

Oxygen Diffusion Into the Tissue

Clearly, diffusion will play an important role for oxygen transport in the body. This is due to the fact that diffusion is the principle physical mechanism of how oxygen from the vessels is reaching the tissue. Without diffusion, oxygen would basically never penetrate the tissue and would only travel through the body like in a closed loop circuit. This physical phenomenon is described by Fick's Law of Diffusion:

$$J = -D\frac{dc}{dx} \tag{2.5}$$

Where J is the diffusive flux, D is the previously mentioned diffusivity coefficient, and $\frac{dc}{dx}$ is the concentration gradient and is basically the derivative of the concentration with respect to the distance x [12], [13].

The diffusion process can be enhanced by a substance called myoglobin. Myoglobin is a protein that can be found in some tissues in the body, especially in muscle tissue, and enhances the diffusivity of oxygen through the tissue. The magnitude of the enhancement

depends on the myoglobin concentration in the given tissue, and is seen as a *moving-carrier-mediated transport mechanism* [17].
For our problem, it is sufficient to say that the diffusive flux of oxygen into the tissue can be computed as the sum of a normal diffusive flux, depending on the concentration gradient and described by Fick's Law 2.5, and a myoglobin facilitated oxygen flux.
The way this phenomenon is mathematically included in our computational models will later be described in the computational model section 3.1.1.

2.3 Current Trends in Modeling

In order to simulate flows, Computational Fluid Dynamics (CFD) is a tool that can be used to produce results which cannot be obtained through in vitro or in vivo experiments [3]. For this reason, and because the computational capabilities available are growing everyday, modeling and simulating processes in the body is taking an increasingly important role in biological and medical sciences. In our case, the focus will lie on oxygen transport simulations and their computational implementation.

As already mentioned in the section 2.1, blood flow in the body can be approximated by the Navier-Stokes equations 2.1, 2.2. Many solvers have already been implemented to solve the Navier-Stokes equations, and thus an appropriate solver can be chosen to compute the blood flow in the body. The transport of oxygen in the blood can then be linked to this by the transport equations [10]. These physical equations are quite straightforward to implement and solve numerically. The next task remains the biggest problem in actual research: to link the blood flow induced transport of oxygen, described by the Navier-Stokes equations, to the diffusion process out of the vessels and into the tissue. The toughest challenge here seems to be in choosing and especially finding physiologically appropriate boundary conditions to get exact results.
This is due to the fact that often samples of vessel networks are considered, that are embedded in virtual tissue. To compute the oxygen field in the vessels and in the tissue, initial and boundary conditions have to be found for the borders of the sample. The incoming oxygen flux through the vessels will have the form of a Neumann Boundary condition, whereas no-flux boundary conditions may be used on the boundary of the given control volume, which will be the border of the used tissue sample. When using these boundary conditions, the assumption is made that all the incoming oxygen is consumed inside the sample. This often leads to exaggerated estimates of hypoxic region, and thus might be physically problematic.
Secomb et. al [15] offered some alternatives in terms of boundary conditions to solve this problem, as the estimation of hypoxic region is considered to be an essential output many researchers aim to understand.

In general, an other important challenge is to develop methods that are stable for many different shapes of networks, and thus can compute physically accurate oxygen fields for different types of network inputs. Many numerical methods fail or more specifically become unstable when the vessel network becomes complicated and heterogeneous. There are already approaches which produce physically accurate results for the Krogh model, but in order to compute oxygen fields in very complex networks like in the kidney, these models might not provide the desired results.

Similar to other methods used in this field by Goldman and Popel [6] or by Beard and Bassingthwaighte [2], the methods used for this thesis will treat the problem of steady-state oxygen delivery and accordingly will compute oxygen fields for steady-state conditions. This approximation is anyways needed in order to apply the Michaelis-Menten kinetics to get a relation between the oxygen consumption rate and the local oxygen concentration. The equations behind this will be explained in the Computational Methods 3.1.1 section.

2.4 Input Data for the Simulations

In this subsection, a short overview about the background of the used networks will be given.

The networks that were used for the simulations were previously extracted from real ex vivo mice organs. This fact is very interesting and relevant for the quality of the computational results. For this, I wanted to give a short insight to how simulations can be made on networks coming from real organs.

This process is done in collaboration by scientists from different areas of expertise. In the first step, a multidomain 3-D image of a tissue has to be taken. This can be done using a micro-computer-tomography (micro-CT) machine, and for especially high resolution imaging, it is performed at the European Synchrotron Radiation Facility.
In order to extract the vessel-networks from those images, a second scientist has to recognize the different tissues surrounding the vessels, and differentiate between arteries and veins. For this task, pattern recognition algorithms and specifically machine learning algorithms are used. The incoming data from the micro-CT machine is in the form of many two dimensional captions. This can be seen as many virtual cuts into the tissue. The algorithms then have to detect the vessels on each of this captions and mark them. This data can be discretized and transformed in order to get a three dimensional network of vessels. The three dimensional network of vessels is our main input for the $DuMu^x$ simulations, and can basically also be used for Green's method simulations.

The previously described tasks are being done by the Interface Group and are generally part of the NCCR kidney project.

3 Simulation Methods

As noted before, this thesis focuses on the evaluation of existing methods that simulate flow and oxygen transport in micro-vascular networks and applies them to several vasculature morphologies from various organs. In this section I will try to give an overall understanding to the theoretical background to the Green's Function Method and accordingly the code developed by Secomb et al. [15]. Later on I will present a new model that I implemented myself using $DuMu^x$ [5].

The methods will be explained in this section and the results will be discussed separately in the following section.

3.1 Green's Function Method

The Green's function method was first used for numerical computations of oxygen delivery in the tissue by Secomb et al. [15]. It was developed with the goal of offering a systematic method that can be applied to many tissue domains with arbitrary shape. Green's function method is computationally less expensive than the implicit methods, due to the fact that the number of unknowns is smaller.
It was found that for many networks like the examples specified in the next chapter, the method seems to be stable and can produce accurate results. Even for complex networks like the ones presented in the Results section 4, the computed oxygen fields seem to be physically accurate which makes this method very interesting. However, using a new and even more complex network, the Green's function method couldn't compute an oxygen field and the outputs were not as expected. This will be discussed in the next sections 4.2 and at the end.

The mathematical background behind the Green's Function Method and the numerical method derived from it will be explained here.

3.1.1 The Computational Model

In this subsection, the used physical quantities, the assumptions and governing equations for the Green's method and for the given problem will briefly be explained. The equations cited here are in my opinion, the most important ones to describe the physical background for the treated problem. This part strongly relies on the description by Secomb et al. [15]. When looking at the code of the Green's function method, and especially at the inputs B.1.1, one can see that the tissue is modeled as a homogeneous medium. This results in constant oxygen diffusivity and solubility coefficients D and α respectively. These quantities usually have to be specified in the SoluteParams.dat-file. A more detailed explanation to the inputs can be found in the Inputs section B.1.1.

The governing equations are listed here, with a short explanation for each of them.
As previously described, diffusion will play an important role in oxygen transport in the body, as oxygen is reaching the tissue by diffusing out of the vessels. This physical phenomenon is described by Fick's Law of Diffusion 2.5. As a reminder, I want to mention that Fick's first Law gives the diffusive flux, which we are interested in, as a function of the diffusivity and the concentration gradient in the tissue.
In addition to this, the principle of conservation of mass applies to oxygen, and we have:

$$\frac{\delta\rho}{\delta t} + \nabla(\rho v) = 0 \tag{3.1}$$

Using Fick's Law, these two fundamental equations can be combined to finally get:

$$D\alpha\nabla^2 P = M(P) \tag{3.2}$$

Equation 3.2 describes the dependency between the oxygen partial pressure P and the consumption rate $M(P)$. Here I would like to mention the fact that partial pressure is physically nothing else than the concentration used in equation 2.5.

For the consumption, a Michaelis-Menten relation applied to the problem is used:

$$M(P) = \frac{M_0 P}{(P_0 + P)} \tag{3.3}$$

In equation 3.3, $M(P)$ is the consumption, M_0 the demand and P_0 the partial pressure at half-maximal consumption.
As previously mentioned, the second important physical phenomenon to describe our transport problem is the convective transport in the blood flow. To model this, we need to know the rate of convective oxygen that is transported through a single vessel segment.

$$f(P_b) = Q(H_D C_0 S(P_b) + \alpha_{eff} P_b) \tag{3.4}$$

In equation 3.4, the flow rate of blood is Q, and H_D, S, C_0, P_b and α_{eff} are parameters such as discharge hematocrit, oxyhemoglobin saturation or partial pressure of oxygen in the blood.
The previously mentioned oxyhemoglobin saturation can be computed by Hill's equation:

$$S(P_b) = \frac{P_b^n}{(P_b^n + P_{50}^n)} \tag{3.5}$$

In Hill's equation 3.5, P_b is the previously mentioned partial pressure of oxygen in the blood, P_{50} is the partial pressure of oxygen at 50% saturation and n is a constant.
Another important equation that we need in order to describe the given problem is the

relationship describing the rate of of diffusive oxygen efflux per unit vessel length:

$$\frac{df(P_b)}{ds} = -q_v(s) \tag{3.6}$$

Equation 3.6 is obtained by using the conservation of oxygen along a vessel segment.
The continuity condition for the partial pressure of oxygen on the vessel-tissue interface gives a second equation for the diffusive oxygen flux:

$$q_v(s) = -D\alpha \int_0^{2\Pi} \frac{\delta P}{\delta r} r_v d\Theta \tag{3.7}$$

Equation 3.13 is an integral form of the diffusive flux $q_v(s)$.
The relation between the partial pressure of oxygen in the tissue and the partial pressure of oxygen in the blood can be approximated with the following Hellums relation [7]:

$$P_v(s) = P_b(s) - Kq_v(s) \tag{3.8}$$

Here $P_v(s)$ represents the partial pressure of oxygen averaged around the circumference of the vessel, K is the intravascular resistance to radial oxygen transport, which is assumed to be constant but depends on the vessel diameter. The value for K for each vessel can be found in the input file IntravascRes.dat (for more information, please see the section Inputs B.1.1), which has to be given to the code.
As discussed in the first part, myoglobin is a protein that supports the diffusion of oxygen in the tissue. The effects of myoglobin-facilitated diffusion can sometimes be neglected, when the myoglobin concentration is very low. For the case where the effects of myoglobin are relevant, one can simply replace the partial pressure P with the partial pressure P^*, given in the following equation 3.9.

$$P^* = P + \frac{D_{Mb} C_{Mb} V_m S_{Mb}(P)}{D\alpha} \tag{3.9}$$

In equation 3.9, D_{Mb}, C_{Mb}, V_m and S_{Mb} are the diffusion coefficient, the concentration, the molar volume and the oxygen saturation of Myoglobin respectively. D and α are the same constants mentioned previously, which were the diffusivity and the solubility coefficients of oxygen. As I mentioned this in the Oxygen Diffusion section 2.2, effects of myoglobin can simply be taken into account by adding a term representing the myoglobin-facilitated-transport of oxygen.

3.1.2 Mathematical Details of the Green's Function Method

In this subsection, the Green's function method, its equations and the application to the given field will be explained [14].
In this section, I will focus on giving a good and understandable insight to the Green's function method and its specific application, instead of getting lost in details. More specific low-level informations about the code and its implementation can be found in the Appendix B.1.1.
The main idea of the Green's function method approach is to model blood vessels as discrete oxygen sources. The oxygen consumption is then modeled by a set of discrete oxygen sinks. As previously described, our goal is to simulate the oxygen transport to finally obtain the oxygen concentration for every node in our tissue and vessel network. In general terms, the oxygen field is computed by a superposition of the fields resulting from each of the sources and sinks.
The strengths of the sources are unknowns in this problem. The sinks are unknowns as well, due to the fact that the oxygen consumption depends on the local partial pressure of oxygen, as one can see in equation 3.3.
When defining the Green's Function G as the potential at a point $\mathbf{x}$ and putting this potential equal to the local oxygen concentration, or equivalently to the partial pressure of oxygen P, we get:

$$D\alpha\nabla^2 G = -\delta_3(\mathbf{x} - \mathbf{x^*}) \tag{3.10}$$

The equation 3.10 is basically the same as 3.2, where $G(\mathbf{x}; \mathbf{x^*})$ is the partial pressure of oxygen at $\mathbf{x}$, resulting from a unit point source at $\mathbf{x^*}$.
The superposition of the fields gives the overall potential $P(\mathbf{x})$ in each point $\mathbf{x}$, which can be computed by integrating the distribution multiplied by the local potential over all the sources:

$$\mathbf{P(x)} = \int_{Sources} G(\mathbf{x;\ x^*}) q(\mathbf{x^*}) d\mathbf{x^*} \tag{3.11}$$

Equation 3.11 gives the general potential (here the partial pressure of oxygen) in a point $\mathbf{x}$, with $q(\mathbf{x})$ representing the distribution of source strengths.
The solution computed in an infinite domain is the singular function as defined in the following equation:

$$G = G_1 = \frac{1}{(4\Pi D\alpha|(\mathbf{x} - \mathbf{x^*})|)} \tag{3.12}$$

Coming back to the finite domain that we are looking at, boundary conditions on the domain boundaries have to be taken into account and thus the solution will be different depending on the boundary conditions that we choose. In this part I would like to repeat the fact that choosing boundary conditions has a substantial consequence on the

results, and there is not yet one type of boundary condition that could physiologically be deducted and can produce overall exact results in terms of hypoxic tissue estimation. For the boundary conditions that were used in the research until now, the computed hypoxic tissue areas were either way too small or way too big compared to what seems to be physiologically realistic. Again I would like to mention the fact that no good measurements can be made about the specific size and the distribution of hypoxic regions in tissue surrounding micro-sized capillaries.

The general approximation that is used puts a distribution of point sources along the centerline of each vessel segment. The field resulting from these sources can then be computed, but has a small error due to the mentioned approximation. The size of this error is computed in the code, but is rather irrelevant for the physical relevance of the resulting oxygen fields. To calculate this error, the diffusive flux at the blood-tissue boundary is obtained with the following equation:

$$q_v(s) = \int_0^L F(s - s^*) q_0(s^*) ds* \tag{3.13}$$

In equation 3.13, the distribution of radial flux across the cylindrical surface is obtained by the function $F(s)$, and $q_0(s)$ represents the distribution of source strength per unit length, s being the distance along the vessel.
The main case is a uniform distribution of sources around the circumference, where we get the following formula for $F(s)$:

$$F(s) = \frac{1}{2}\delta_1(s) + \frac{k(K(k) - E(k)}{4\Pi r_0} \tag{3.14}$$

In 3.14, $K(k)$ and $E(k)$ are the following functions, also called elliptic integrals:

$$K(k) = \int_0^{\frac{\Pi}{2}} (1 - k^2 sin^2\Theta)^{\frac{-1}{2}} d\Theta \tag{3.15}$$

$$E(k) = \int_0^{\frac{\Pi}{2}} (1 - k^2 sin^2\Theta)^{\frac{1}{2}} d\Theta \tag{3.16}$$

3.2 $DuMu^x$

The main task of this bachelor thesis project was to implement a model that computes transport of O_2 in plasma through microvascular networks and compare the results with those that take red blood cells into account. The intention was to quantify how much change the consideration of red blood cells indeed brings to oxygen gradients in tissue, and if the morphology of vasculatures play a role. For this purpose, I had to decide which solver I want to use.
At first sight, the decision had to be made between $OpenFOAM$, $FEniCS$ and $DuMu^x$. When starting this project, we knew that the biggest limitation and challenge to face will be the given time, which was quite short to get into such a complex subject and understand the methodology behind it.
For this reason, I had to do some research to see which solvers might produce the best results and will allow me to construct a model as fast as possible. As $OpenFOAM$ is a prominent software for Computational Fluid Dynamics (CFD) simulations, this seemed like an interesting idea. $FEniCS$ seemed like a very good option to implement a 1D-3D coupled problem, but was finally rejected as it rather uses the finite element method (FEM) and the finite volume method (FVM) than finite differences, and the computational disadvantages linked to this were considered as a major limitation. We decided that we did not want to work with FEM and FVM, but in the present, the work done by Holter et al. [8] has shown that good results can be produced using these methods.
Finally, as $DuMu^x$ provided some interesting blood flow simulation models that were already implemented, and thus seemed to be a very attractive alternative, I finally decided to use $DuMu^x$ to implement an oxygen transport model.

However, getting a good understanding of this software to be able to implement a new method was a challenging task, especially for the short time of a Bachelor Thesis, which is limited to a few months.

3.2.1 General Structure of $DuMu^x$

$DuMu^x$ is an open source software developed by a strong user community. The users are spread around many places in the world. The software was basically developed by the University of Stuttgart, and provides a C++ based library of numerical methods and implemented test models, to simulate and solve transport and flow processes in porous media.
$DuMu^x$ is a "multi-scale multi-physics toolbox" that aims to describe the physical properties of a specific problem as correct as possible, by focussing on minimizing the computational cost [5].

As $DuMu^x$ is based on the principle of $DUNE$, which is a modular toolbox that can be used to solve partial differential equations with grid-based methods [1], the core modules of $DUNE$ have to be downloaded and installed together with the $DuMu^x$ module itself.
$DUNE$ basically consists of a few modules that can be downloaded from GitHub in the form of separate packages. The modules contain for example the generally used classes, discretization algorithms and solvers that are used in order to describe and run the physical simulations.
$DuMu^x$ itself is an additional $DUNE$ module, that uses classes, functions and solvers implemented in the other $DUNE$ modules. It provides the user a few very interesting test examples which can be changed, merged or completely rewritten in order to get a new physical problem.
The structure of $DuMu^x$ is specified in this chapter and illustrated with some figures. Figure 1 is the Structure of $DuMu^x$ given from the $DuMu^x$ Version 2.12 Documentation [1].

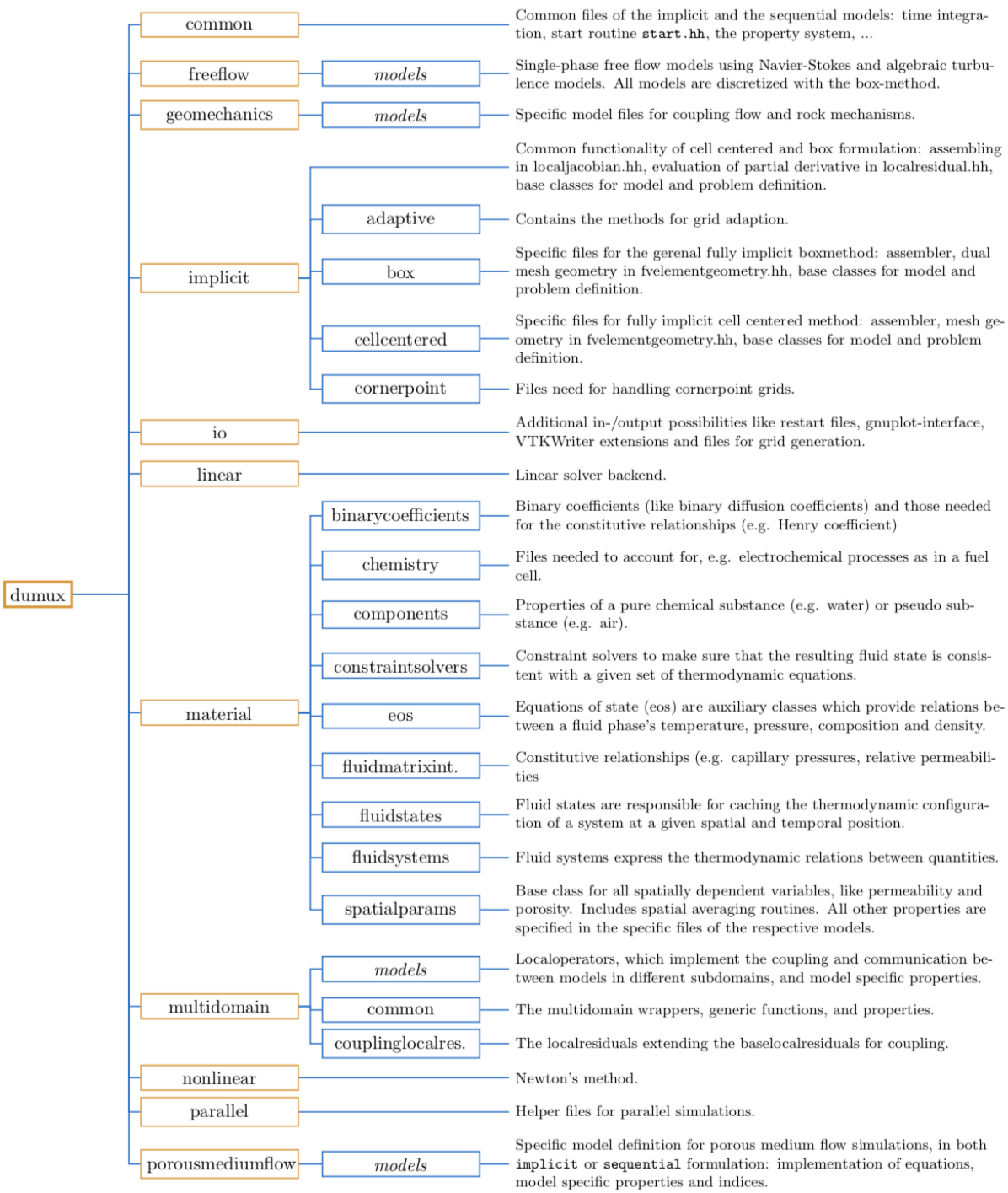

Figure 1: The $DuMu^x$ Structure

Available Numerical Models

As I mentioned, $DuMu^x$ offers a large variety of models that can be used to solve a specific problem. This provides large flexibility in terms of computational cost and stability questions as explained by $DuMu^x$ developers Flemisch et al. [1], [5].
One has to take care to choose a solver that is compatible with the given problem. To develop a better feeling for this, the test cases can be studied in detail. A good overview to the user will be given when checking which solver, grid-creator or discretization is used for each type of problem.
Nevertheless, using my personal experience, I can say that some information is hard extract, and the incompatibility of some modular pieces inside the $DUNE$ modules, like for example grid creators, can be the cause for some serious errors which were hard to understand and debug without a deep understanding of $DuMu^x$, which again takes time to aquire.
I will briefly present some of the numerical models that can be used with $DuMu^x$, so that the reader can later get a better understanding of the model I worked on.

In $DuMu^x$ there are two types of numerical models: *coupled fully-implicit* and *decoupled semi-implicit*. The difference lies simply in the fact that the so called *coupled fully-implicit* models are strongly modularized, and the user can build a model only prescribing physical quantities like fluxes and source terms. For the *decoupled semi-implicit* models, the user has to work with the actual physical equations, as for example the implementation of the transport equation. As I mainly used the 1p-model to build by oxygen transport model, I will shortly explain what this means and give a few other examples.

1p basically means that the model is computing the flow for a one phase liquid. If the considered liquid is a mixture of two components, one should rather use the 1p2c model, which takes into account one phase (1p) and two components (2c). We can also consider a two phase liquid by using the two phase (2p) model. I will not go into further details concerning this, as it is rather irrelevant for oxygen computations, but still would like to mention the fact that thermal effects can also be taken into account for most of these models.
A more interesting point for us is that some models, as for example the 1p model, can be computed in two separate problem-files describing two flow problems in different networks, and can then be coupled to produce a 1p-1p model computing results in an embedded network. In other words, this coupled model of two separate 1p models can be calculated together, to obtain the pressure in a vessel network, and the pressure in the tissue surrounding this network. This will be explained in more detail in the next section.
Further general details concerning $DuMu^x$ and its structure can be found in the $DuMu^x$ reviews [1], [5].

3.2.2 Implemented Model

As mentioned previously, some of the existing test cases can be combined to solve a more sophisticated and specific problem. In our case, this refers to the coupling of the existing tracer model to the 1p-1p model.

The 1p-1p Model

As I indicated, the 1p-1p model itself is the coupled version of a 1p simulation in a network, and a 1p simulation in the tissue surrounding it. It can simulate the flow of a solute in a network by convection and the diffusion of this solute into the tissue through the coupling. This coupled model was already a very useful tool, and helped me to compute pressure drops in interesting vascular networks as for example glomeruli. The computed pressures seemed physically correct, whereas the velocities seemed rather unrealistic. I will come back to presenting and discussing these results in the Results section 4.
My main idea was to use the computed velocities from the 1p-1p model, and feed the tracer with this as an input in order to obtain the oxygen transport and diffusion into the tissue and finally get the oxygen field in the tissue for steady-state conditions.

The Tracer Model

The second model I used to build my oxygen tracking model is the tracer. The tracer model can literally *trace* the path of a solute when fluxes or velocities are given as input data. My main idea was to use the computed velocities from the 1p-1p model, and feed the tracer with this as an input in order to obtain the oxygen transport and diffusion into the tissue and finally get the oxygen field in the tissue for steady-state conditions.
Generally, the tracer model is computing the distribution of a solute inside a control volume for every time step.
As it can *trace* substances in networks of different geometries, and accordingly compute concentrations and concentration fields of solutes in dynamic systems, it seemed like a very interesting option in order to compute the oxygen field in the tissue surrounding a given network.
Even though it is hard to show the outputs of the tracer model, a visualization for the first and last time of a tracing simulation is pictured in the Appendix B.2.

The Coupled Model

To achieve the goal of tracing oxygen in the given vessel-networks, I coupled the previously described models to obtain a 1p-1p-tracer model.

I would like to repeat that the velocity output of my previous 1p-1p model is the input for the tracer model. This is important, as I will later explain the problems I had when trying to produce results with this model. Generally I wanted to get the concentration of my solute of interest, which is oxygen, for every time step.
As my goal was to compute oxygen fields not only in the network, but also in the tissue surrounding it, I had to create a 1p-1p-tracer-tracer model. The second tracer is coupled in order to compute the oxygen concentration not only in the vessels (this is what a 1p-1p-tracer model would do) but also for the tissue. Due to time limitations, the implementation of this model did not complete fully and is left for future efforts. Further explanations concerning this will follow in the Results 4 section.

4 Simulation Results and Analysis

In this section, the produced results using the two introduced methods will be presented and analyzed for a few different networks. Finally the results will be discussed and the methods will be compared. The accuracy of produced results as well as the physiological meaning behind these results will be the main subject in this chapter.

4.1 Results of the Green's Function Method

The Green's Function Method is providing many outputs which are specified in more detail in the Appendix B.1.1. In this section, some of these outputs will be presented and explained. The physiological meaning will be described and finally the results will be discussed.

Krogh Model as an Introduction

The Krogh model describes the vessels as cylinders and the tissue as concentric cylinders surrounding the vessels. In this model, it is assumed that the capillaries are straight and parallel. The blood flow in all the capillaries is assumed to be unidirectional, and the distribution of capillaries is assumed to be homogeneous [9]. As one might expect, this model has a few unrealistic assumptions, but as it is a straightforward network, it is the most basic model to start with to validate a computational method.

The Krogh model has been a standard simulation network used as an initial point for developments as it has a very easy and simple structure. Figure 2 is the result of a Green's Method blood flow simulation computed on a Krogh network. The Solute Concentration in the network and in the tissue is visualized in the form of a slice (see output section B.1.1 for more information). The illustration shows an impression of oxygen levels in the tissue and the vessel segments, and generally gives the local oxygen concentrations in mmHg.

One can clearly see that the results produced are physically realistic. In figure 2, one can see that the blood flow goes from left to right. The O_2 concentration in mmHg accordingly goes down in flow direction. The concentrations as well as the diffusive distance in the tissue also decrease in flow direction, which can be explained by the fact that a lower concentration gradient goes with a lower diffusive flux. This is in accordance with Fick's Law of Diffusion 2.5. As one can see from the input files B.1.1, the diffusivity constant α is constant for homogeneous tissue. Due to the fact that the diameter of the vessels is constant as well when considering the Krogh model, the intravascular resistance to radial oxygen transport K for each diameter/vessel is also constant. Differences in diffusive

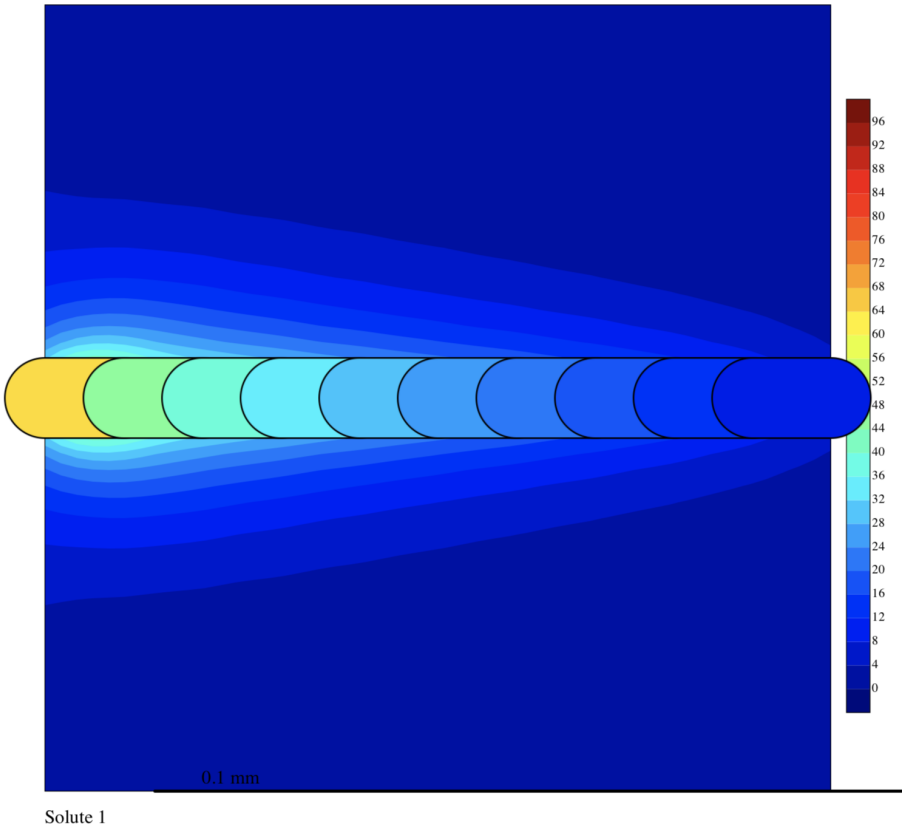

Figure 2: Green's Contour Output for Krogh Network

fluxes and oxygen penetration distances into the tissue therefore mainly depend on the PO_2 gradients themselves.

Eventually, one can see that the results computed on the Krogh network show that the Green's function method performs well and produces physically accurate results on simplified and perfectly homogeneous networks as this one.
It is further remarked that analytical solutions of the Krogh cylinder model have been derived before and the results shown in this figure correspond well to the analytical results. This validates that the Green's function method and the corresponding implementation can produce physically accurate results, and suggests that it is a suitable method to conduct further studies.

A First Tumor Network

The Green's function code provided by Prof. Secomb contains a few microvascular networks as examples, which can be studied and extended. In this thesis, I executed many of those models to get a good understanding of physiological phenomena concerning O_2 transport in various organs and different networks.
Two of these example networks provided by Prof. Secomb are the tumor networks. I will first present the results obtained for the first network, and later get to the more complex and relevant second example.
The Tumor 1 network is more complicated than the previously presented Krogh model but generally gives a way better insight to the results than the Cardiac Network simulation, due to the much lower vessel density. The result is visualized on Figure 3. One can see the result of a Green's method blood flow simulation computed on a simple tumor network. As for the previous network, the oxygen concentration in the network and in the tissue is visualized in the form of a two dimensional slice (see output section B.1.1 for more information). Again the illustration shows an impression of oxygen levels in the tissue and the vessel segments, and generally gives the local oxygen concentrations in mmHg.

The Tumor network provided by Secomb was used for further simulations and to validate the produced data. Figure 3 is the result of a Green's method blood flow simulation computed on a tumor network. The solute concentration in the network is visualized.

As contour-plots generally show a *compressed* picture of a three dimensional network, many vessels that are *above* each other are shown as if they were intersecting. This can be hard to differentiate for the output files of some complex networks. In this case, this tumor network doesn't have many vessels, but is rather supplied by a few thicker vessels. The red color corresponds to oxygen enriched blood, and clearly oxygen in tissue regions proximal to these vessels is higher. We do not have any information about the exact origin of this particular image segmentation, but it can be observed that the length of segments within vessels is quite large. This is a limitation in order to obtain a precise resolution of the computed oxygen field, as PO_2 is always computed for each node and segment. Eventually a PO_2 value is assigned to each segment. This means that a lower resolution for the oxygen field is obtained.
In this case, PO_2 levels reach up to about 40 mmHg which lies within the physiological range.

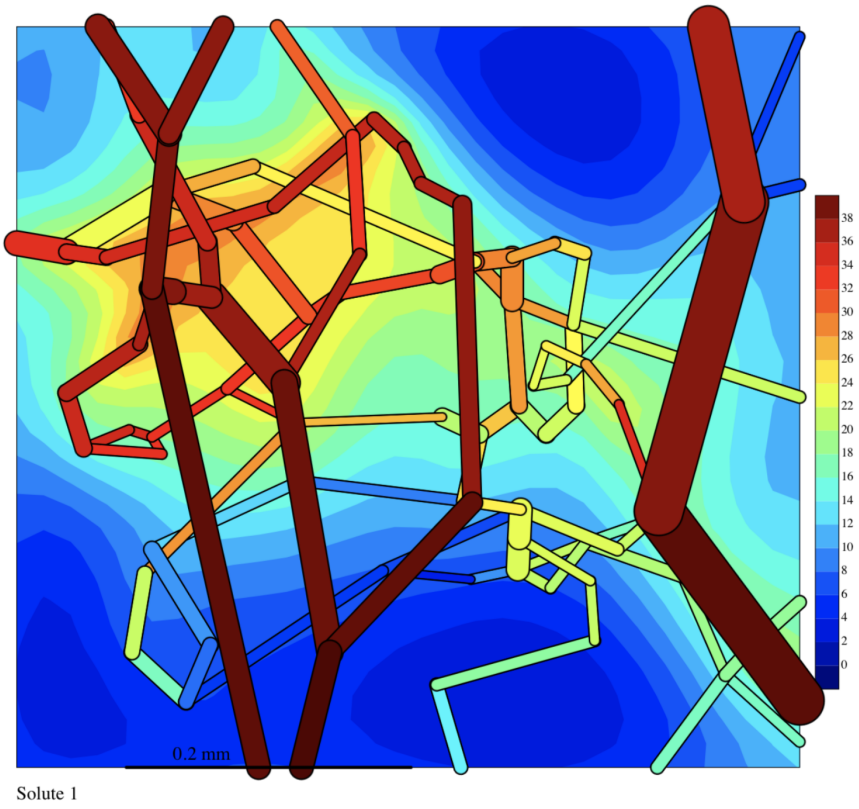

Figure 3: Green's Contour Output for Tumor 1

The Cardiac Network

Another interesting example network is the cardiac network. Even though we do not have specific informations about how this particular image segmentation was obtained, knowing that it is a cardiac network, one might expect a high density of vessels for this network.
In this section, the oxygen distribution in a cardiac network will be presented. This network is more complicated than the previously presented Krogh model 4.1. The result is visualized on Figure 4. The given output shows the result of a Green's method blood flow simulation computed on a cardiac network. As for the Krogh and the Tumor 1 networks, the oxygen concentration in the network and in the tissue is visualized in the form of a slice (see output section B.1.1 for more information). Again the illustration shows an impression of oxygen levels in the tissue and the vessel segments, and generally gives the local oxygen concentrations in mmHg.

Figure 4 shows the distribution of oxygen in a small network of vessels. The three dimensional network of vessels is *compressed* to a two dimensional representation in order to get a simpler output file. This kind of output has the advantage of encapsulating most of the computed values, even though it might look complex for some networks, where the oxygen fields are very heterogeneous. The red color corresponds to oxygen enriched blood.

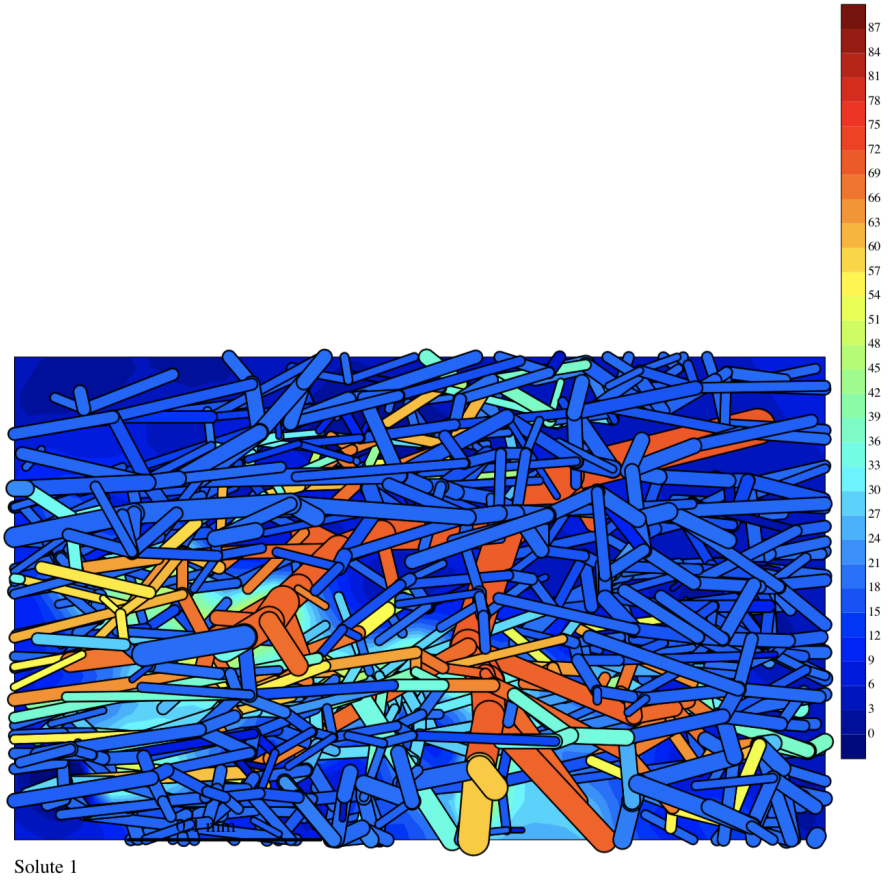

Figure 4: Green's Contour Output for Cardiac Network

The principle vessels go through the middle of the three dimensional tissue section and therefore have the highest oxygen partial pressures which are around 65 mmHg. The oxygen concentration in the finer and smaller vessels surrounding the big oxygen-supplying vessels accordingly decreases with growing distance from the oxygen sources. But still the oxygen in tissue regions proximal to the bigger oxygen-supplying vessels is higher. Once again, it can be observed that the length of segments within vessels is quite large, which causes a lower oxygen field resolution.
The PO_2 levels reach up to about 84 mmHg within a physiological range.

The Mesenteric Artery Network

Another example network provided by Prof. Secomb is the mesenteric artery network. Like for the previously presented networks, we don't have specific information concerning the network. In the absence of information, this network is used as a sample network to study the computational results. Knowing that it is supposed to represent a mesenteric artery network, which is an artery supplying the large intestine, one can clearly see the very small vessels surrounding the main arterial tree. These vessels are embedded in the intestine itself and supply oxygen to the surrounding cells.
Generally the vessels are very fine and the vessel resolution is high, which entails a good oxygen field resolution for the tissue surrounding the vessels.
The result is visualized on Figure 5. The output shows the result of a Green's method blood flow simulation computed on a mesenteric artery network. As for the previous networks, the oxygen concentration in the network and in the tissue is visualized in the form of a slice (see output section B.1.1 for more information). Again the illustration shows an impression of oxygen levels in the tissue and the vessel segments, and generally gives the local oxygen concentrations in mmHg.

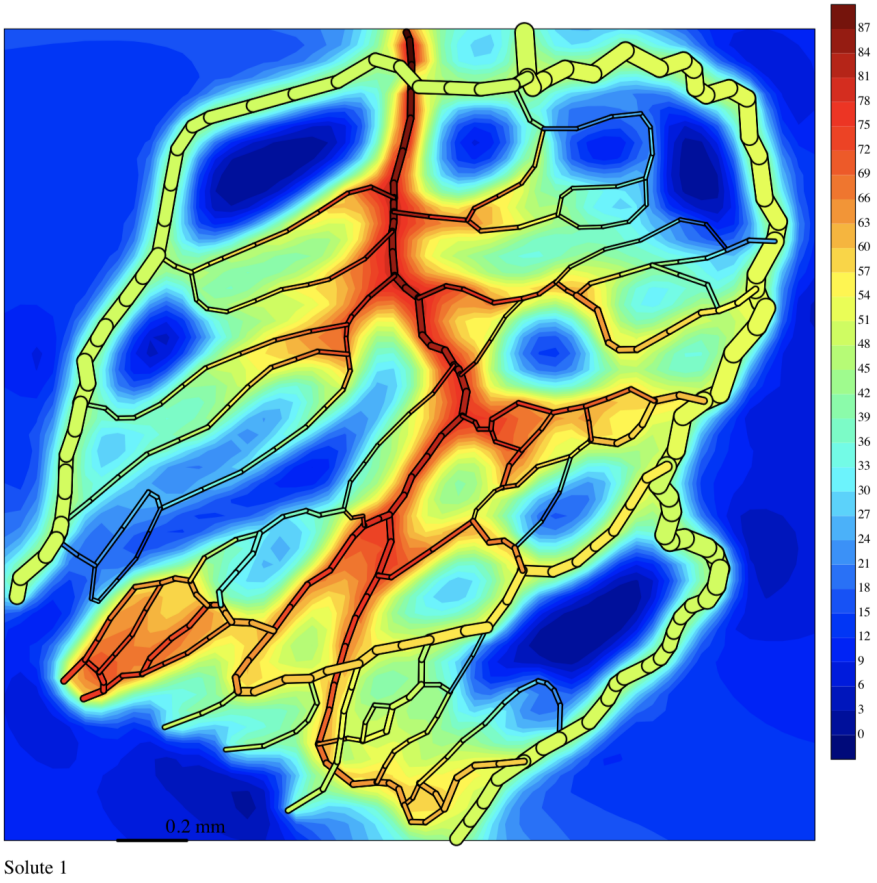

Figure 5: Green's Contour Output for a Mesenteric Artery Network (Solute 1)

This network is more complicated than the previously presented networks but gener-

ally gives a better insight to the results than the cardiac network simulation, due to the much lower vessel density. Especially the tissue concentrations and the changing oxygen concentrations in longitudinal direction of the vessels can be seen very well.
Due to the low density of capillaries for this network, this figure can give a good insight to the values computed by the Green's model code. One can clearly see that the partial pressure of O_2 goes down in radial direction with growing distance from the vessels. At hotspots where the capillary density is higher due to branching and ramification, the diffusion comes from two vessels and the superposition of two close source elements leads to peaks in PO_2 values.

From a physiological point of view, the computational result is correct when assuming that the oxygen consumption through the whole intestine is constant. The oxygen supply to the intestinal vessels surrounding the main arterial tree is constant, and the oxygen concentrations in the intestinal ares are slightly above 50 mmHg, which is in the physiological range.
The general oxygen field clearly shows that some hypoxic regions can be found at spots without vessels or where the vessels themselves have a very low PO_2 concentration. With PO_2 values ranging from 0 mmHg (hypoxic region) to 87 mmHg, the results for the whole sample lie once more in the physiological range.

A Second Tumor Network

Another interesting tumor example network provided by Prof. Secomb is the Tumor 2 network.
Even though there is no information given about how this particular image segmentation was obtained, knowing that it is a tumor network, one might expect a random ramification and distribution of vessels. In fact, this network is build by vessels of different sizes and the oxygen concentrations within the vessels is largely varying.
The Tumor 2 network is more complicated than the previously presented Tumor 1 network and the computational simulation produces a very interesting output. The result is visualized on Figure 6. Here one can see the result of a Green's Method blood flow simulation computed on a simple Tumor network. As for the previous models, the Oxygen Concentration in the network and in the tissue is visualized in the form of a slice (see output section B.1.1 for more information). Again the illustration shows a virtual cut into the tissue and gives local oxygen concentrations in mmHg.

Figure 6 is the result of a Green's method blood flow simulation computed on a tumor network. The oxygen concentration in the network and the tissue is visualized.

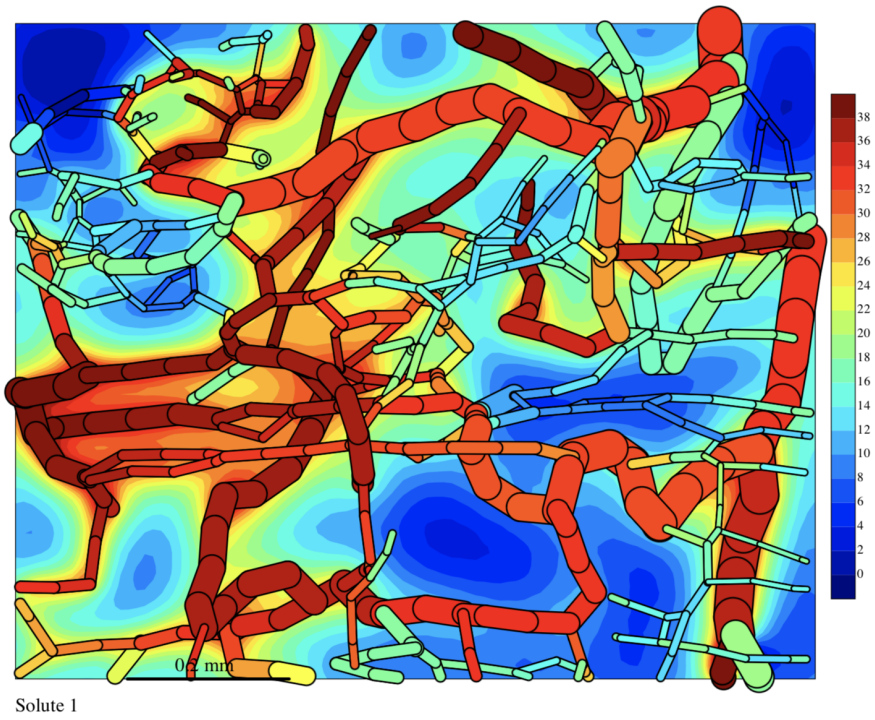

Figure 6: Green's Contour Output for Tumor 2

The overlapping of the vessels with different oxygen concentrations is due to the fact that the three dimensional space of vessels is projected on a two dimensional plane, as for

the previous examples. The red color corresponds to oxygen enriched blood which is in this case carried by the thicker vessels supplying this sample of tissue. The oxygen partial pressure in the tissue regions proximal to these bigger vessels is higher with peaks around 40 mmHg. The segmentation resolution depends on the vessel, but is overall rather high, which entails a high resolution for the oxygen field in the tissue.
Generally the highest PO_2 levels can be found surrounding the big vessels and especially in regions where many sources are overlapping. This is due to the fact that the source terms are superpositioned when calculating the oxygen field during the computational iterations, and makes sense from a physiological point of view, as a lot of oxygen is supplied to these areas.
The PO_2 levels range between around 3 and 40 mmHg within a physiological range. Generally the average PO_2 in this sample is quite high, due to the equally distributed and well oxygenated big vessels.

Further Outlook for New Networks

An overview to simulation results of the Green's method was presented for different networks. The networks covered a range from homogeneous and simple vasculatures to rather complex and heterogeneous samples. Some of the examples were physiologically very interesting as were the computed oxygen fields within the tissue.
Considering how well this method seems to perform for some complex networks I presented in the last section, the question of how well it can perform for new networks obtained through imaging techniques was raised. For this reason, I tried to simulate the oxygen field on a glomerulus network using the Green's method code. The same glomerulus network was also used for $DuMu^x$ simulations (Glomerulus 1, Figure 7).

In general, computing oxygen fields with diffusion into the tissue is physiologically meaningless for a glomerular network, as in reality glomeruli are not surrounded by normal tissue. In this case, I considered the glomerular network as a vascular network example, and used it as a simple vasculature, as if the network was embedded anywhere else in homogeneous tissue.
Figure 11 is one of the ouputs of a Green's method oxygen field simulation computed on a glomerulus network. In this case, the presented output file is not a contour plot, but rather a plot illustrating the labeled nodes and vessel segments in the given network. This type of plot is called *NedNodesSegs* and is supposed to give an overview of the used segments and nodes in the network.
At this point, I would like to mention that the code did not run as expected, and thus didn't produce a sink field. As the Green's method iteratively computes a source and a sink field, which are used to obtain the oxygen distribution field by superposition, no oxygen field could be computed. The numbers written on the output file 11 are the numerations of the segments and the nodes, which was successfully done by the code (see output section B.1.1 for more information).
This shows that this very complex network, which is the Glomerulus 1 network, could be fully recognized, which is already an interesting achievement. However, no computations in terms of oxygen fields could be made, neither in vessels, nor in the virtual tissue (again real glomeruli are not surrounded by tissue). The reasons for this could not be investigated in detail due to time restrictions, and will remain a task for future research. As the network is very complex, for now no correct explanation can be given without doubts. It is likely that there might be a problem with the parameters we chose concerning the solute or the intravascular resistance (see Input Section B.1.1 for more information).

4.2 Discussion of the Green's Method and the Produced Results

As the presented examples show, the Green's method and its implementation can produce some excellent results in terms of oxygen field simulations in networks.

The results are in accord with the previously presented physical laws, which describe the oxygen transport from the vessel to the tissue. Moreover, the results are physiologically accurate. The computed values generally range between 0 mmHg (hypoxic tissue) and 87 mmHg (well oxygenated tissue) and seem to be in the physiological range.
Even though these computations have been made on networks that were derived from organs, no information was given about how the particular image segmentations were obtained. This also makes it hard to compare the results with experimental values, especially as we don't have physiological data available for these specific networks. In particular, I want to point out that the computed values cannot be compared to any specific values that we could expect or that were previously measured in the past. We can only say that the values are generally in the physiological range and that the physical properties of oxygen transport and diffusion are well respected and in accord with the simulation results, as we don't know anything in particular about the provenance of the networks.

Particularly interesting results could be observed for the mesenteric artery network and the second tumor example, as the contour plots for these networks provided a vey good insight to the computed oxygen fields inside the tissue and around the vessels. Being able to compute oxygen fields for complex three dimensional networks on this level underlines the qualities of the used method and its ability to perform computations on complex vasculatures.

However, using the glomerulus network in the last section 4.1, we have seen that the Green's method cannot directly compute an oxygen field for any arbitrary vasculature. The reasons for this are not clear yet and will have to be understood in the future, but it seems like the parameters specified for each network have to be tuned in order to produce the expected oxygen fields. The network data given to the code as an input was in the same form as the previous input files, but still might be an error source that couldn't be resolved. Other possible error sources might be in the values given in the other input files (see B.1.1 for more information about the input files). While the code was running, it seemed like the first guess for the oxygen source field and accordingly for the oxygen sink field couldn't be made, and thus the following iterations couldn't compute an oxygen field.
For a deeper understanding of the performance of this method, further research has to be made using new networks. It would make sense to use networks for which the provenance is known, as for the presented glomerulus.

4.3 Results of $DuMu^x$

Some blood flow simulations were run to evaluate the quality of results that can be produced using $DuMu^x$ 3.2. In this section I will present and explain some results produced by $DuMu^x$, and some rather physical than physiological interpretations of simulation results will be discussed. The results of the previously presented 1p-1p model will be shown, and some details about the implemented 1p-1p-tracer model will be given at the end.

Some Glomerulus Network Simulations

In order to analyze the pressure differences and the pressure drop through a glomerulus network, the 1p-1p model was run on several different glomeruli. Coming from an engineering background, pressure variations and drops in simple pipes was a topic I had worked with. However complex networks like the ones that can be found in glomeruli were nothing I had seen until now, and the explanation behind the pressure distribution inside the glomerulus is quite an unknown field.

The figures 7 (Glomerulus 1), 8 (Glomerulus 2), 9 (Glomerulus 3) and 10 (Glomerulus 4) are results of $DuMu^x$ blood flow simulations using the 1p-1p model in order to compute the pressure field inside the vessels of different glomeruli networks. The pressure field of the flow is visualized, where red is a relatively high pressure, and blue a relatively low pressure. Each glomerulus is pictured from two different angles. The second perspective for each glomerulus (Figures 16, 17, 18 and 19) can be seen in the Appendix B.2.

The figure 7 shows a clear pressure drop in the flow direction for Glomerulus 1. The pressure drop takes place rather constantly over the glomerular network, and pressures fall from around 70 mmHg down to less than 10 mmHg. This pressure drop can be observed with a slightly different range of pressures for the glomeruli 2, 3 and 4. [1]

Physiologically, this pressure drop can be explained by the fact that the afferent arteriol, which is the vessel for the incoming blood, has a bigger diameter than the efferent arteriol, which is the vessel carrying the blood away from the glomerulus. From a simple engineering point of view, a bigger diameter for the outgoing flow results in a *relaxation* of the fluid, and thus results in a pressure drop. By that we can say that the computational results of the 1p-1p model are physiologically correct.
It is possible to observe the same pressure drop when analyzing the pressure field in Glomerulus 2, and the rather constant fall in pressure is physically correct. A constant fall in the pressure is also what we would expect in a straight pipe. For this glomerulus,

[1] The pressure drop is qualitative as initial conditions were heuristically chosen. Future work should apply a pressure profile at the afferent arteriole and then compute the pressure gradient between efferent and afferent arteriole.

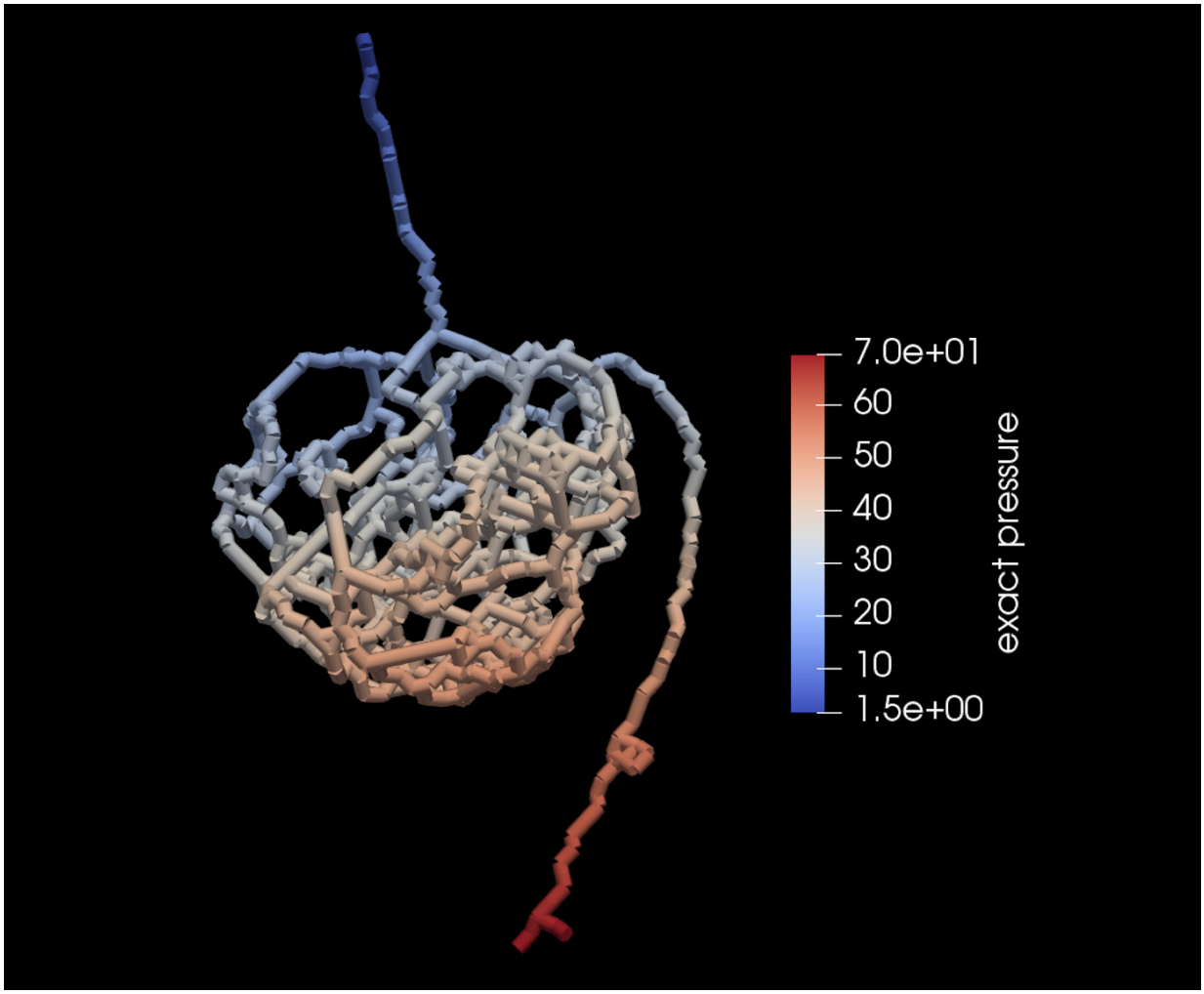

Figure 7: Pressure Field in Glomerulus 1

I would like to point out the fact that due to imaging artifacts and post-processing of the data, arterioles of this particular glomerulus seem to be missing, and thus these parts look rather short, but can still be seen on the figure. An especially interesting point is the fact that some vessels inside the glomerulus and the glomerular network have lower pressures than the pressure computed for the efferent arteriol. This can also be seen for Glomerulus 3 (figure 9) and Glomerulus 4 (figure 10).

Generally, it is possible to observe that the pressure constantly drops over the afferent and the efferent arteriols, before and after the glomerular network respectively. This pressure drop happens due to viscous effects and is proportional to the length, which explains the constant pressure drop. As previously mentioned, the afferent arteriol has a bigger diameter than the efferent arteriol in average, which causes an overall pressure difference from a physiological point of view. It is hard to understand the local pressure gradients inside segments of the glomerular network between the afferent and efferent arteriols, which might be due to diameter changes and in particular due to the complexity of glomerular networks in general.

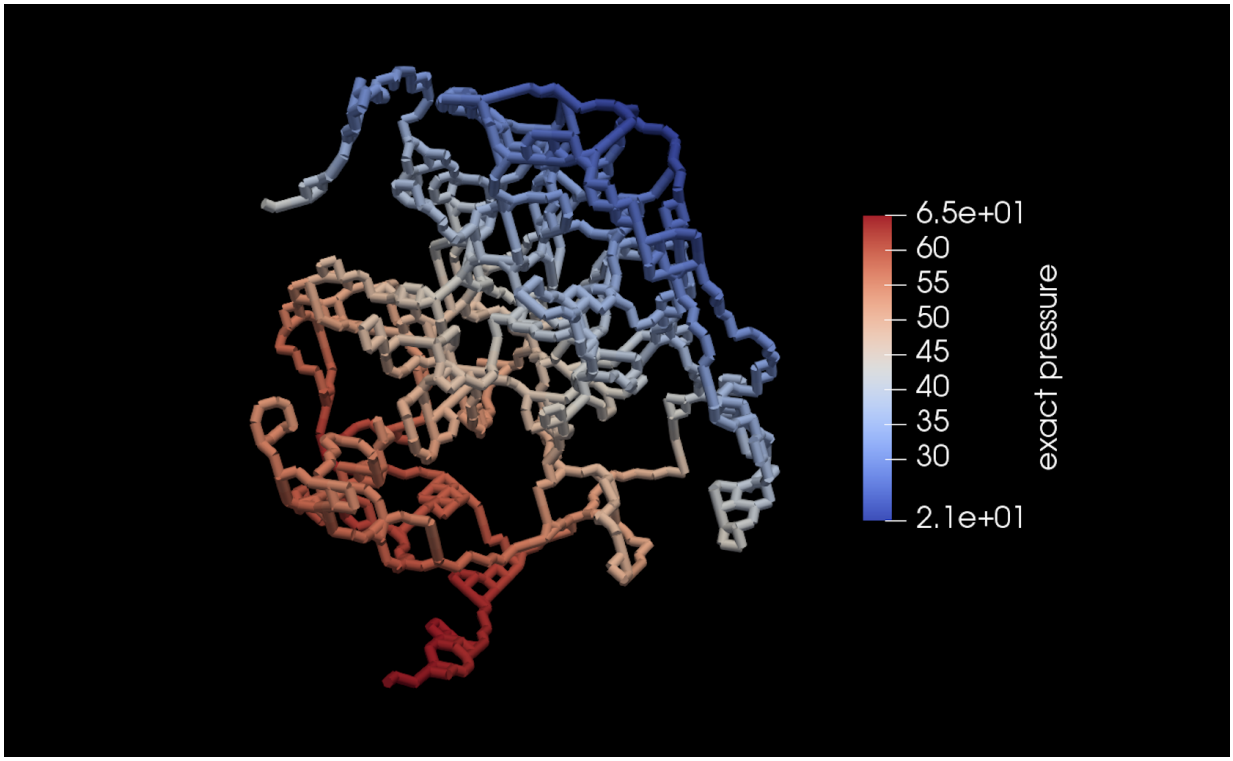

Figure 8: Pressure Field in Glomerulus 2

Results of the Implemented Method

As I already briefly mentioned this when describing this method, the 1p-1p-tracer model cannot produce useful results yet. This new model in which I coupled the 1p-1p model with a new tracer model is basically still in the build-up phase, and with more time, the reasons for the wrong outputs can eventually be resolved so that the model can produce physically accurate results.

After spending a lot of time on debugging the errors I was having, the code finally compiled and the tracer could compute the oxygen distribution in my network (for a specific insight into the code, please see the Appendix 12). However, the results are not satisfying as the oxygen concentration is computed to be zero at every time step in every node of the grid. The reasons for this are still not quite clear and some more investigation has to be made in order to understand why the tracer always has null values as outputs. Moreover, the initial and boundary conditions for the oxygen concentrations were examined and this seems not to be the source of the error.

As $DuMu^x$ is generally a complex software, many errors can occur, and it will surely take some time to understand the underlying problems in order to produce results using this model. However, I am persuaded that this approach can and will produce good results once the coupling of the models works well.

4.4 Comparison of the Methods and Results

In this section the main differences in the produced results by the Green's function method and $DuMu^x$ models will be discussed for networks of similar complexity.

Before starting, I would like to mention that oxygen fields computed on the same networks cannot directly be compared, as the $DuMu^x$ method (1p-1p-tracer-tracer) couldn't produce useful results yet, and thus no oxygen field in the tissue was computed. Additionally, the Green's function method didn't produce results on the Glomerulus 1 network either, which shows how hard it is to reproduce results using different methods in general.
While the Green's function method is presenting a specifically created C++ based software for oxygen transport and diffusion simulations, the results show that the obtained outputs can be very interesting for many heterogeneous and complicated networks. Unfortunately, oxygen fields could not be computed on every arbitrary network that was used, which raises the question of when and how the Green's method can produce results in general, in terms of oxygen field computations in the tissue.

Compared to this, the $DuMu^x$ based model is still in its very initial phase and cannot produce correct results yet. However, due to the fact that it is a $DuMu^x$ based model, it offers both a large flexibility to the user in terms of further development, as well as in terms of computational flexibility when choosing a time- and space-discretization through choosing discretization modules and different grid creators. The solvers can also be chosen by the user, which doesn't guarantee an overall stable numerical solver, but can provide great flexibility when adapting the model to specific networks.

Both the Green's function method and the $DuMu^x$ based 1p-1p model produced physically accurate results, which proved that they can be used to compute physiologically relevant results. Finally, the two methods currently compute entirely different entities, which is oxygen concentrations for the Green's method and overall pressure field for the $DuMu^x$ based 1p-1p method, and as the 1p-1p-tracer-tracer model is still in the build-up process, the results of the two different softwares used for this thesis cannot directly be compared for the moment.

5 Discussion

This thesis aimed to understand how computational methods can be used to simulate blood flow and oxygen transport in microvascular networks. For this, two approaches have been presented 3.1, 3.2 and the results or outlooks have been discussed 4.

While oxygen transport is a crucial topic in the fields of medicine and physiology, many processes in microvascular networks are not understood yet. Especially the measurements of microvascular and tissue oxygen concentrations is an uncharted research field. For this reason, a lot of research is being done in oxygen distribution computations [6, 12, 9, 14] . In view of the challenges that are faced when simulating oxygen fields in tissue surrounding vessel networks, I would like to split this last section into three parts: a technical discussion, a physical discussion, and finally a physiologic discussion.

5.1 Technical Aspects

As the main tools used for the previously presented simulation results are computational solvers, I would like to focus on the technical aspects behind these tools in general.

The share computational simulations are taking in physiologic research is growing everyday. This is also due to the fact that some data is very hard to extract from measurements, and thus researchers try to understand certain processes through running computational simulations on physical models. However, the challenges behind using these technical tools are not to be underestimated. As I experienced myself when running simulations with the Green's method code 3.1 and $DuMu^x$ 3.2, many errors and bugs can stop the code from running and can be very hard to fix. Also, once a model is implemented, one has to take great care to examine if the produced results physically make sense. As for example for the $DuMu^x$ model I implemented, the zero outputs are clearly wrong and the model has to be improved in order to produce realistic results 4.3.
The reason computational models are used so frequently is also due to the fact that some calculations, like the pressure gradients in the previously presented glomeruli 4.3, can not be obtained without computational methods. This is especially due to the particular vasculature of these networks. Thus, these tools can give us very interesting insights, even though their complexity is certainly challenging, and will demand a good understanding of the methods underlying, depending on the used software.

5.2 Physical Aspects

As the exact physical equations and mathematics behind the different simulation methods was explained in the first sections, it was obvious that the success of computational simulations is strongly linked to the correctness of the physical assumptions.

When building a computational model like the models used in this thesis, one has to start by studying the physical equations describing the given problem 3.1.1. In our case, the physical equations describing the blood flow, and accordingly the oxygen transport from the vessel to the tissue have been used in general. Solving these equations will then yield the results that the user wanted to compute 4. This basic procedure is an engineering procedure, and is the same for a flow and heat exchange computation in a pipe or a blood flow and oxygen diffusion computation in a tissue. Due to this fact, engineers are strongly engaged in this research field, as they have a good knowledge about the technical tools and have a good understanding for physical quantities. Also, the used networks and fluiddynamic challenges can be very interesting in the physiology field, as for example the flow in a complex vascular network like the one found in glomeruli 4.3, where particularly interesting pressure differences and gradients could be observed. To understand and analyze these results physically, a deeper understanding of the Navier-Stokes equations is required, as the pressure drop was not comparable to the one in a straight pipe.

Due to these reasons, engineers, mathematicians and physicists are and will be used more and more often in these research domains in the future, as some physical problems can not be solved without computational methods to this day.

5.3 Physiological Aspects

Finally, I would like to give a short outlook to the research done in this thesis from a physiological point of view.

As the methods and the produced outputs were presented in the previous sections 3, 4, one could clearly see that the computed results were physiologically very interesting. With computed oxygen concentrations ranging from 0 mmHg to around 87 mmHg in the tissue, the values seemed realistic and the distributions inside the tissue seemed to be qualitatively correct. The pressure gradients computed by the $DuMu^x$ model 3.2 were also analyzed and the general pressure field could physiologically be explained. Again, we do not really have experimental values to compare the computed values with, so most of the results can only be analyzed qualitatively.
Especially the oxygen fields computed by the Green's method 3.1 in some complex networks produced very interesting and physiologically useful results 4.1. Even though we

don't have experimental data to compare the computational values with, it is possible to see that the results were all in the physiologic range and the general oxygen distribution showed that the outputs were in accord with the physics describing our problem 3.1.1. Extrapolating from the examples presented for oxygen distribution computations, I would like to underline the fact that computational models are very useful in this specific field and will very likely take an even more important share in the future.

References

[1] Dumux homepage. URL http://www.dumux.org/documents/dumux-handbook-2.12.pdf.

[2] D. A. Beard and J. B. Bassingthwaighte. Modeling advection and diffusion of oxygen in complex vascular networks. *Annals of biomedical engineering*, 29(4):298–310, 2001.

[3] A. Caiazzo and I. E. Vignon-Clementel. Mathematical modeling of blood flow in the cardiovascular system. In *Quantification of Biophysical Parameters in Medical Imaging*, pages 45–70. IEEE, 2018.

[4] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, A. Tatomir, M. Wolff, et al. Dumux: Dune for multi-{phase, component, scale, physics,...} flow and transport in porous media. *Advances in Water Resources*, 34(9):1102–1112, 2011.

[5] B. Flemisch, J. Fritz, R. Helmig, J. Niessner, and B. Wohlmuth. Dumux: a multi-scale multi-physics toolbox for flow and transport processes in porous media. In *ECCOMAS Thematic Conference on Multiscale Computational Methods for Solids and Fluids*, pages 82–87, 2007.

[6] D. Goldman and A. S. Popel. A computational study of the effect of vasomotion on oxygen transport from capillary networks. *Journal of Theoretical Biology*, 209(2):189–199, 2001.

[7] J. Hellums. The resistance to oxygen transport in the capillaries relative to that in the surrounding tissue. *Microvascular research*, 13(1):131–136, 1977.

[8] K. E. Holter, M. Kuchta, and K.-A. Mardal. Sub-voxel perfusion modeling in terms of coupled 3d-1d problem. *arXiv preprint arXiv:1803.04896*, 2018.

[9] F. Kreuzer. Oxygen supply to tissues: the krogh model and its assumptions. *Experientia*, 38(12):1415–1426, 1982.

[10] D. Kuzmin. A guide to numerical methods for transport equations. *University Erlangen-Nuremberg*, 2010.

[11] R. Law and H. Bukwirwa. The physiology of oxygen delivery. *Update in Anaesthesia*, 10:6, 1999.

[12] C.-J. Lee, B. S. Gardiner, J. P. Ngo, S. Kar, R. G. Evans, and D. W. Smith. Accounting for oxygen in the renal cortex: a computational study of factors that predispose the cortex to hypoxia. *American Journal of Physiology-Renal Physiology*, 313(2):F218–F236, 2017.

[13] R. N. Pittman. Regulation of tissue oxygenation. In *Colloquium series on integrated systems physiology: from molecule to function*, volume 3, pages 1–100. Morgan & Claypool Life Sciences, 2011.

[14] T. Secomb, R. Hsu, M. Dewhirst, B. Klitzman, and J. Gross. Analysis of oxygen transport to tumor tissue by microvascular networks. *International Journal of Radiation Oncology* Biology* Physics*, 25(3):481–489, 1993.

[15] T. W. Secomb, R. Hsu, E. Y. H. Park, and M. W. Dewhirst. Green's function methods for analysis of oxygen delivery to tissue by microvascular networks. *Annals of Biomedical Engineering*, 32(11):1519–1529, Nov 2004.

[16] R. Weiss. Parameter-free iterative linear solvers. *Mathematical Research*, 97, 1996.

[17] J. B. Wittenberg. Myoglobin-facilitated oxygen diffusion: role of myoglobin in oxygen entry into muscle. *Physiological Reviews*, 50(4):559–636, 1970.

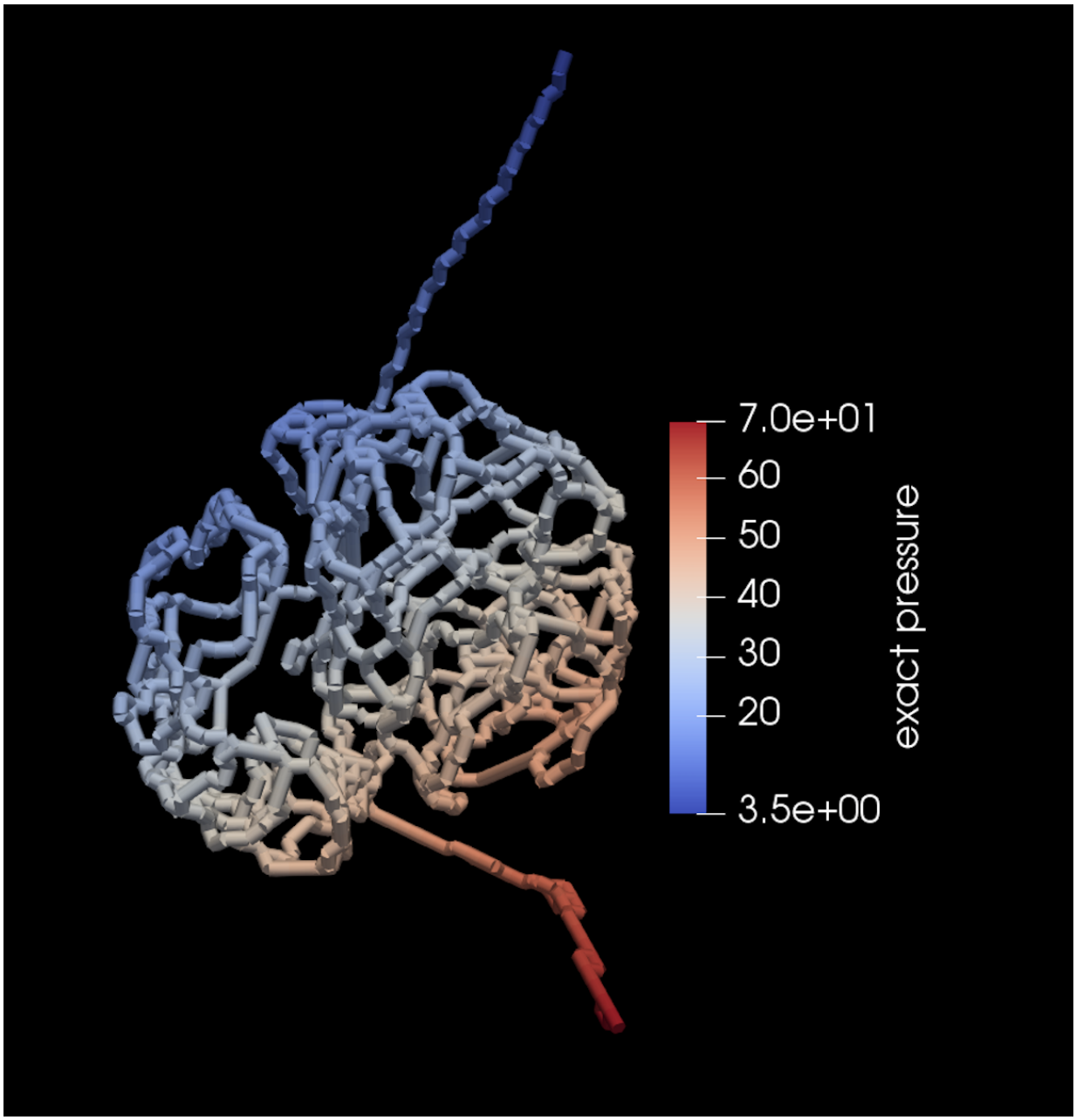

Figure 9: Pressure Field in Glomerulus 3

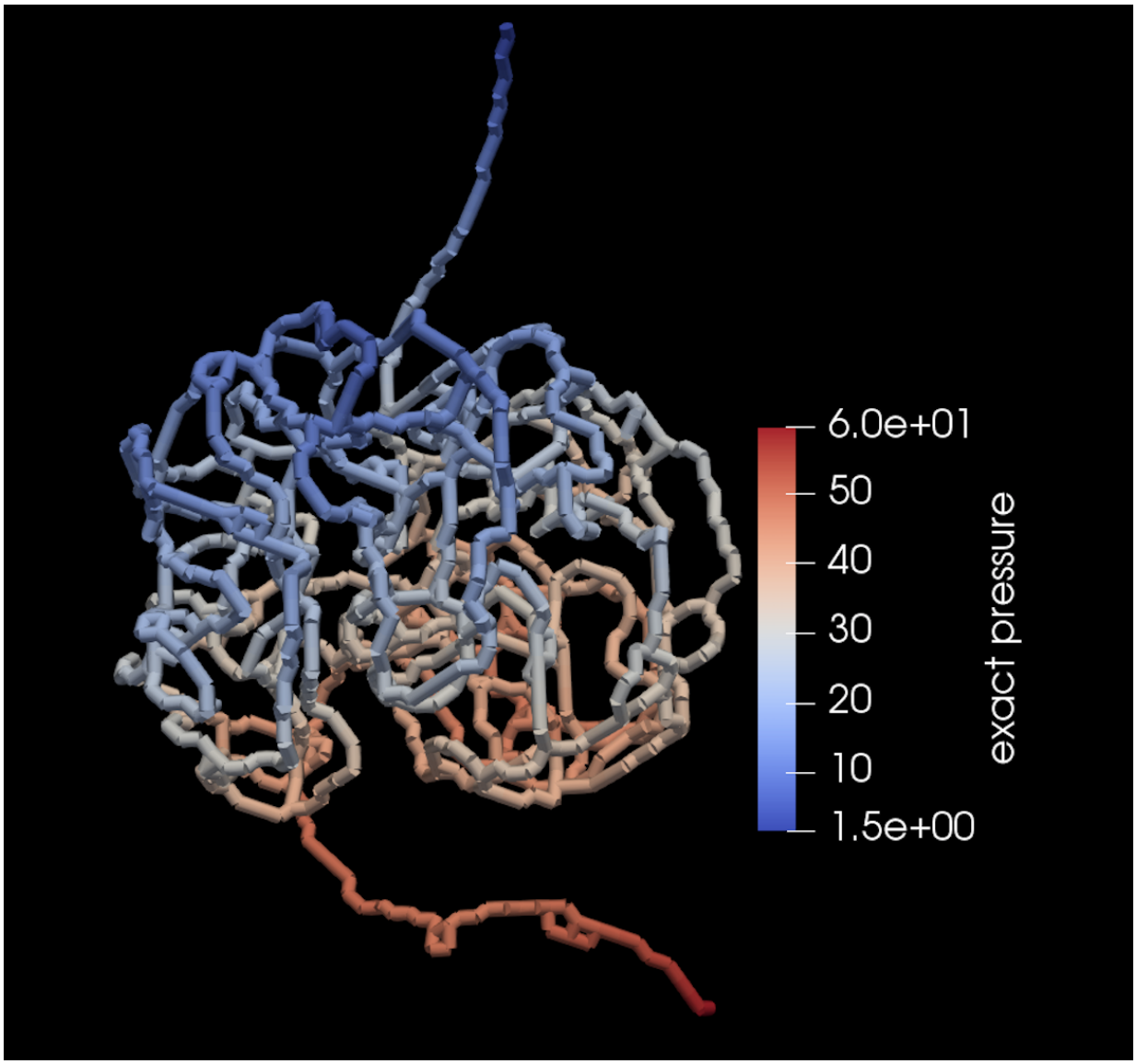

Figure 10: Pressure Field in Glomerulus 4

A Acknowledgements

First of all, I would like to thank my supervisor Kartik Jain for helping me build a good understanding of both the topic and the used tools, and supporting me during the project with great advice. In addition to helping me gain a good knowledge about subject-specific contents, he helped me gain an insight into research on a high-level. While being very demanding, he taught me to pay attention to the crucial details, and especially showed me how to convert ambition into good work by his own example.
The time I spend at the *Interface Group* was a valuable experience, which will surely help me in my future work. I think that I gained a considerable amount of new knowledge, as working inside the group gave me a particularly interesting insight into how things are being done in academic research on a high-level.

I would like to thank everyone for integrating me in the group and making the work for this bachelor thesis both interesting and very enjoyable.

Finally, I would like to thank Vartan Kurtcuoglu for offering me the possibility of doing my thesis in the group, and taking the time to supervise me.

B Appendix

B.1 Additional Material for the Green's Function Method

B.1.1 General Structure of the Used Code

In this section, an insight to the code provided by Secomb et al. [15] will be given. Especially a description about what data this code uses and what data it computes will follow.. The main goal is to understand what the results are and where they really come from. For this, a general explanation of the structure and organization of the code and the methods behind the computations will given.

Inputs

In general, the Green's function method code takes .dat-files as inputs, which basically contain the network files and the parameters for the diffusion and oxygen transport inside the vessels.

The input data in form of .dat-files consists of four files:

- The SoluteParams.dat-file: This file contains all the information about the solutes we are looking at. In our case, this is simply oxygen. The specific informations are things like the tissue solubility, the Michaelis constant of consumption, etc.

- The Network.dat-file: This file contains all the information about the network we are looking at. This information is basically in form of an array. This array defines for each segment the type of vessel, the start point, the end point, the diameter (in microns), the relativ flow and the hematocrit level of the blood inside the given vessel.

- The IntravascRes.dat-file: This file contains all the information about the diameter on which the sources are computed around the vessel centerline. The second important information given here is the intravascular resistance to radial oxygen transport K for each diameter and vessel.

- The ContourParams.dat-file: This file contains all the information necessary to create the contour-plots. This means that we create a plot, where the oxygen concentration inside a three dimensional domain can be visualized on a two dimensional projection. In this file, the projection plane can be specified, and the focus geometry inside the three dimensional space can be selected.

The part of the code where the read-in of the network data and the specific transport

parameters through the input files is being done is the input.cpp code.

The analyzenet.cpp-part of the code analyzes the input-files and gives each vessel and node a number. Through this, the oxygen concentration can be computed for each node of the three dimensional space. The output after this step can be seen in Figure 11, but it is important to note that usually in the next step, the oxygen gradients are computed, and the segmentation numbers are replaced by the specific oxygen levels for each node and vessel.

Outputs

The Green's function method code gives .dat-files and images (contour.ps-files) as outputs, which basically contain the results that have been discussed previously in the Results 4 chapter.

The contour.ps-files are produced by the contour.cpp-part of the code, which basically generates and writes the data for the contour-plots. This data is a plot of the vessel from bottom to top according to the z-coordinate. In other words, this is a two dimensional projection of the computed oxygen fields in a three dimensional space. One could see this plot as a 2D picture of a 3D network, giving some information about the general oxygen fields everywhere in the segment. There is an option of computing the transport of different solutes in the network, in which case a new page is generated for each solute, and the computed solute concentration for each area is shown in colors. In our case, the only used solute is oxygen and thus only one contour-plot is shown for each network.

The lines on the contour plots are generated by the contr-lines.cpp-part of the code, whereas the contr-shade.cpp-part generates the colors and the color bar on the side of the generated contour-plots.

The code responsible for the creation of the NetNodesSegs.ps and the NedNodesOxygen.ps-files is picturenetwork.cpp. Here, the network and the computed oxygen fields of the network are projected on a z = 0 plane. For this, the parameters from the Countour-Params.dat file are used. Finally, a postscript file called NetNodesSegs is produced, in which nodes are labeled with nodvar and segments with segvar. A second file called NetN-odesOxygen is produced, in which the computed oxygen level for each node is pictured.

The general outputs of the code can be classified into two categories:

- The first category is in the form of PostScript-files that can be visualized. These outputs are basically pictures showing the computed oxygen concentrations. The three files are the ones that were previously mentioned: the Contour.ps-file, the NedNodesSegs.ps-file and finally the NedNodesOxygen.ps-file.

- The second category is in the form text-files, containing the computed data in arrays. These values are the computed oxygen partial pressures for each node in the tissue, and represent the oxygen field output for a three-dimensional segment.

Organization of the Code

The code provided by Secomb et al. [15] consists of a few .cpp-files, so that tasks like input read-ins, calculations and output-file generations are separated. Each file has a specific role, which will be explained in detail in this section. As one can think, the most important part of this code is the implementation of the Green's function method and its application to the given data.

- The solver:
  The bicgstab.cpp-part of the code is the implementation of a solver based on the Parameter-free iterative linear solver by R. Weiss [16].

- O_2 partial pressure computation:
  The blood.cpp-part of the code is quite interesting, as it basically does an important part of the calculation to obtain the oxygen concentration field in the vessels and the tissue. In this part of the code, the partial pressure of oxygen is computed depending on the oxygen concentration in the red blood cells. The oxygen concentration in the red blood cells is taken as an input and the partial pressure of oxygen is given as an output for each node. This step is then repeated iteratively as the results for the computed oxygen field is converging.

- Computation of the convective flux:
  The convect.cpp-part of the code does the convective transport part of the calculation. It examines the flow direction, to compute the oxygen concentration in each vessel segment in the right order.

- Evaluation of solute fields:
  The eval.cpp-part of the code does the evaluation of the solute field depending on the source strengths provided. As mentioned previously, the source strength field is computed after a first random guess, and the solute concentration field calculations are being done iteratively starting from this guess.

- Numerical methods:
  There are some numerical methods and mathematical calculations behind the simulation, as for example the *Gauss-Jordan elimination* or the *Lower-Upper decomposition*, which have not been studied in detail for the purposes of this thesis, but are implemented in the gaussj.cpp-part and the ludcmp.cpp-part of the code.

- Green's method:
  The main implementation of the Green's function method is in the greens.cpp-part of the code. Here the Green's function approach for multiple reacting species is applied. The equations described in the Methods 3.1 section are implemented to compute the oxygen partial pressure PO_2.

- Histogram computation:
  In this part, histograms of the solute levels are evaluated, and a histogram.dat-file is generated as an output. This file has the form of a text-document, and contains a array with the data.

- Initial guessing of the source field:
  As previously mentioned, a guess for the initial tissue source strengths is being done. Later, the source strength is computed iteratively and the computed sink field depends on this calculation. This is due to the Michaelis-Menten relation we are using to calculate the oxygen consumption. For this reason, the initgreens.cpp-part of the code is only used at the beginning of each simulation.

- Node generation:
  The direction of the flow is an essential information which we have to take care of, in order to get physically accurate results. For this, the nodes have be generated in the right order. The rank of the node is exactly the information pictured on the output NetNodesSegs.ps. For this, the putrank.cpp-code generates nodes in order of flow direction, depending on the physical direction of the flow.

- Array generation:
  Arrays with fixed dimensions, as for example vessel segments, have to be generated. This is done by the setuparray.cpp-part of the code, where the dimensions of segments is set.

- Testing the convection direction:
  The correctness of the direction in the convective flow is tested by comparing the matrix values with values obtained by numerical differentiation. This is done in the testconvect.cpp-part of the code.

- Calculation of the uptake rate:
  In the tissrate.cpp-part of the code, the uptake rate of the computed solutes (in our case it is only oxygen) in the tissue is computed as a function of solute levels. This means in our case, that the tissue uptake (the diffusion and the myoglobin enhanced transport) of oxygen is calculated depending on the concentration surrounding the given tissue.

B.1.2 Some More Green's Results

The figure 11 shows the previously mentioned NetNodesSegs.ps output computed for a glomerulus network. One can see that a number (or more precisely a rank) is assigned to each vessel segment and to each node. This step is the first step that has to be done before any computations concerning oxygen fields in the vessels and the tissue can be done.

B.2 Additional Material for DuMuX

The 1p-1p-tracer-tracer Code

To give the reader a short overview about the code used for the implementation of the 1p-1p-tracer-tracer model, a code snippet of the coupled model is attached on Figures 12, 13, 14 and 15.

Some More Glomeruli

To picture the previously mentioned pressure computations on glomerular networks, the pressure field simulation result are shown from a second perspective. The figures 16, 17, 18 and 19 show the simulation results for the Glomeruli 1, 2, 3 and 4.

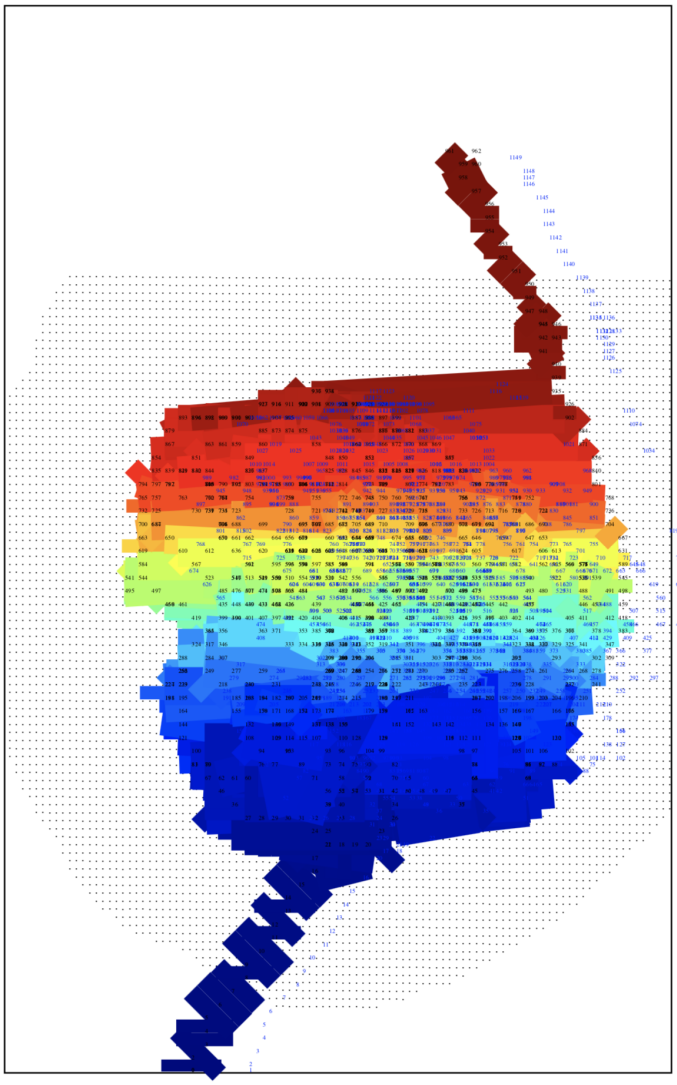

Figure 11: Green's Net Nodes Output for a Glomerulus Network

Some Tracer Simulations

To give an example of what the tracer model is computing, a simulation of a simple diffusion of groundwater in a porous rock is given. The velocity field of the solute (Figure 22) is assumed to be constant over the time. The solute field for the first time step can be seen on Figure 20. The solute field in the rock for the last time step is visualized on Figure 21.

Clearly the water has moved in the porous rock, and the solute field for the last time step is different than the field in the first time step. The solute is *traced* in the given domain, and a solute field for each time step is computed.

I attached these images to qualitatively give the reader an insight into how the tracer works, and what the idea behind tracing oxygen in the tissue, in order obtain oxygen fields, was.

test_1p_1p_tracer_tracer.cc 〉 nonLinearSolver(assembler, linearSolver, couplingManager)

```
249
250     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
251     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
252     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
253
254         ////////////////////////////////////////////////////////
255         // compute volume fluxes for the tracer model
256         ////////////////////////////////////////////////////////
257         using Scalar =  typename GET_PROP_TYPE(LowDimTypeTag, Scalar);
258         std::vector<Scalar> volumeFlux(lowDimFvGridGeometry->numScvf(), 0.0);
259
260         using FluxVariables =  typename GET_PROP_TYPE(LowDimTypeTag, FluxVariables);
261         auto upwindTerm = [](const auto& volVars) { return volVars.mobility(0); };
262         for (const auto& element : elements(lowDimGridView))
263         {
264             auto fvGeometry = localView(*lowDimFvGridGeometry);
265             fvGeometry.bind(element);
266
267             auto elemVolVars = localView(lowDimGridVariables->curGridVolVars());
268             elemVolVars.bind(element, fvGeometry, sol[lowDimIdx]);
269
270             auto elemFluxVars = localView(lowDimGridVariables->gridFluxVarsCache());
271             elemFluxVars.bind(element, fvGeometry, elemVolVars);
272
273             for (const auto& scvf : scvfs(fvGeometry))
274             {
275                 const auto idx = scvf.index();
276
277                 if (!scvf.boundary())
278                 {
279                     FluxVariables fluxVars;
280                     fluxVars.init(*lowDimProblem, element, fvGeometry, elemVolVars, scvf, elemFluxVars);
281                     volumeFlux[idx] = fluxVars.advectiveFlux(0, upwindTerm);
282                 }
283                 else
284                 {
285                     const auto bcTypes = lowDimProblem->boundaryTypes(element, scvf);
286                     if (bcTypes.hasOnlyDirichlet())
287                     {
288                         FluxVariables fluxVars;
289                         fluxVars.init(*lowDimProblem, element, fvGeometry, elemVolVars, scvf, elemFluxVars);
290                         volumeFlux[idx] = fluxVars.advectiveFlux(0, upwindTerm);
291                     }
292                 }
293             }
294         }
295
296     ////////////////////////////////////////////////////////////
297     // setup & solve tracer problem on the same grid
298     ////////////////////////////////////////////////////////////
299
300     //! the problem (initial and boundary conditions)
301     using TracerProblem = typename GET_PROP_TYPE(TracerTypeTag, Problem);
302     auto tracerProblem = std::make_shared<TracerProblem>(lowDimFvGridGeometry);
303
304     // set the flux from the 1p problem
305     tracerProblem->spatialParams().setVolumeFlux(volumeFlux);
306
307     //! the solution vector OKAY
308     SolutionVector x(lowDimGridView.size(0));
309     tracerProblem->applyInitialSolution(x);
310     auto xOld = x;
311
312     //! the grid variables OKAY
```

Figure 12: The 1p-1p-tracer-tracer Code (1)

test_1p_1p_tracer_tracer.cc 〉 nonLinearSolver(assembler, linearSolver, couplingManager)

```
294         }
295
296     ////////////////////////////////////////////////////////////
297     // setup & solve tracer problem on the same grid
298     ////////////////////////////////////////////////////////////
299
300     //! the problem (initial and boundary conditions)
301     using TracerProblem = typename GET_PROP_TYPE(TracerTypeTag, Problem);
302     auto tracerProblem = std::make_shared<TracerProblem>(lowDimFvGridGeometry);
303
304     // set the flux from the 1p problem
305     tracerProblem->spatialParams().setVolumeFlux(volumeFlux);
306
307     //! the solution vector OKAY
308     SolutionVector x(lowDimGridView.size(0));
309     tracerProblem->applyInitialSolution(x);
310     auto xOld = x;
311
312     //! the grid variables OKAY
313     using GridVariables = typename GET_PROP_TYPE(TracerTypeTag, GridVariables);
314     auto gridVariables = std::make_shared<GridVariables>(tracerProblem,lowDimFvGridGeometry);
315     gridVariables->init(sol[lowDimIdx], oldSol[lowDimIdx]);
316
317     //! get some time loop parameters OKAY
318     //  const auto tEnd = getParam<Scalar>("TimeLoop.TEnd");
319     //  auto dt = getParam<Scalar>("TimeLoop.DtInitial");
320     //  const auto maxDt = getParam<Scalar>("TimeLoop.MaxTimeStepSize");
321
322     //! instantiate time loop
323     //  auto timeLoop = std::make_shared<CheckPointTimeLoop<Scalar>>(0.0, dt, tEnd);
324     //  timeLoop->setMaxTimeStepSize(maxDt);
325
326     //! the assembler with time loop for instationary problem
327     //  using TracerAssembler = FVAssembler<TracerTypeTag, DiffMethod::analytic, /*implicit=*/false>;
328     //  auto assembler2 = std::make_shared<TracerAssembler>(tracerProblem, fvGridGeometry, gridVariables, timeLoop);
329     //  assembler2->setLinearSystem(A, r);
330
331     //! intialize the vtk output module
332      VtkOutputModule<TracerTypeTag> vtkWriter(*tracerProblem, *lowDimFvGridGeometry, *gridVariables, sol[lowDimIdx], tracerPr
333     using VtkOutputFields = typename GET_PROP_TYPE(TracerTypeTag, VtkOutputFields);
334     VtkOutputFields::init(vtkWriter); //!< Add model specific output fields
335     vtkWriter.write(0.0);
336
337
338
339     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
340     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
341     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
342
343     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
344     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
345     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
346
347     ////////////////////////////////////////////////////////////
348     // compute volume fluxes for the tracer model
349     ////////////////////////////////////////////////////////////
350     using Scalar =  typename GET_PROP_TYPE(BulkTypeTag, Scalar);
351     std::vector<Scalar> volumeFlux2(bulkFvGridGeometry->numScvf(), 0.0);
352
353     using FluxVariables =  typename GET_PROP_TYPE(BulkTypeTag, FluxVariables);
354     auto upwindTerm = [](const auto& volVars) { return volVars.mobility(0); };
355     for (const auto& element : elements(bulkGridView))
356     {
357         auto fvGeometry = localView(*bulkFvGridGeometry);
```

Figure 13: The 1p-1p-tracer-tracer Code (2)

test_1p_1p_tracer_tracer.cc 〉 nonLinearSolver(assembler, linearSolver, couplingManager)

```
338
339     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
340     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
341     ////////////////////////////////////////NEW TRACER FLOW////////////////////////////////////////////
342
343     ////////////////////////////////////////NEW TRACER DIFFUSION//////////////////////////////////////////
344     ////////////////////////////////////////NEW TRACER DIFFUSION//////////////////////////////////////////
345     ////////////////////////////////////////NEW TRACER DIFFUSION//////////////////////////////////////////
346
347     ////////////////////////////////////////////////////////////
348     // compute volume fluxes for the tracer model
349     ////////////////////////////////////////////////////////////
350     using Scalar =  typename GET_PROP_TYPE(BulkTypeTag, Scalar);
351     std::vector<Scalar> volumeFlux2(bulkFvGridGeometry->numScvf(), 0.0);
352
353     using FluxVariables =  typename GET_PROP_TYPE(BulkTypeTag, FluxVariables);
354     auto upwindTerm = [](const auto& volVars) { return volVars.mobility(0); };
355     for (const auto& element : elements(bulkGridView))
356     {
357         auto fvGeometry = localView(*bulkFvGridGeometry);
358         fvGeometry.bind(element);
359
360         auto elemVolVars = localView(bulkGridVariables->curGridVolVars());
361         elemVolVars.bind(element, fvGeometry, sol[bulkIdx]);
362
363         auto elemFluxVars = localView(bulkGridVariables->gridFluxVarsCache());
364         elemFluxVars.bind(element, fvGeometry, elemVolVars);
365
366         for (const auto& scvf : scvfs(fvGeometry))
367         {
368             const auto idx = scvf.index();
369
370             if (!scvf.boundary())
371             {
372                 FluxVariables fluxVars;
373                 fluxVars.init(*bulkProblem, element, fvGeometry, elemVolVars, scvf, elemFluxVars);
374                 volumeFlux2[idx] = fluxVars.advectiveFlux(0, upwindTerm);
375             }
376             else
377             {
378                 const auto bcTypes = bulkProblem->boundaryTypes(element, scvf);
379                 if (bcTypes.hasOnlyDirichlet())
380                 {
381                     FluxVariables fluxVars;
382                     fluxVars.init(*bulkProblem, element, fvGeometry, elemVolVars, scvf, elemFluxVars);
383                     volumeFlux2[idx] = fluxVars.advectiveFlux(0, upwindTerm);
384                 }
385             }
386         }
387     }
388
389     ////////////////////////////////////////////////////////////
390     // setup & solve tracer problem on the same grid
391     ////////////////////////////////////////////////////////////
392
393     //! the problem (initial and boundary conditions)
394     using TracerProblem = typename GET_PROP_TYPE(TracerTypeTag, Problem);
395     auto tracerProblem = std::make_shared<TracerProblem>(bulkFvGridGeometry);
396
397     // set the flux from the 1p problem
398     tracerProblem->spatialParams().setVolumeFlux2(volumeFlux2);
399
400     //! the solution vector OKAY
401     SolutionVector x(bulkGridView.size(0));
```

Figure 14: The 1p-1p-tracer-tracer Code (3)

test_1p_1p_tracer_tracer.cc 〉 nonLinearSolver(assembler, linearSolver, couplingManager)

```
388
389     ////////////////////////////////////////////////////////////
390     // setup & solve tracer problem on the same grid
391     ////////////////////////////////////////////////////////////
392
393     //! the problem (initial and boundary conditions)
394     using TracerProblem = typename GET_PROP_TYPE(TracerTypeTag, Problem);
395     auto tracerProblem = std::make_shared<TracerProblem>(bulkFvGridGeometry);
396
397     // set the flux from the 1p problem
398     tracerProblem->spatialParams().setVolumeFlux2(volumeFlux2);
399
400     //! the solution vector OKAY
401     SolutionVector x(bulkGridView.size(0));
402     tracerProblem->applyInitialSolution(x);
403     auto xOld = x;
404
405     //! the grid variables OKAY
406     using GridVariables = typename GET_PROP_TYPE(TracerTypeTag, GridVariables);
407     auto gridVariables = std::make_shared<GridVariables>(tracerProblem,bulkFvGridGeometry);
408     gridVariables->init(sol[bulkIdx], oldSol[bulkIdx]);
409
410     //! get some time loop parameters OKAY
411     //  const auto tEnd = getParam<Scalar>("TimeLoop.TEnd");
412     //  auto dt = getParam<Scalar>("TimeLoop.DtInitial");
413     //  const auto maxDt = getParam<Scalar>("TimeLoop.MaxTimeStepSize");
414
415     //! instantiate time loop
416     //  auto timeLoop = std::make_shared<CheckPointTimeLoop<Scalar>>(0.0, dt, tEnd);
417     //  timeLoop->setMaxTimeStepSize(maxDt);
418
419     //! the assembler with time loop for instationary problem
420     //  using TracerAssembler = FVAssembler<TracerTypeTag, DiffMethod::analytic, /*implicit=*/false>;
421     //  auto assembler2 = std::make_shared<TracerAssembler>(tracerProblem, fvGridGeometry, gridVariables, timeLoop);
422     //  assembler2->setLinearSystem(A, r);
423
424     //! intialize the vtk output module
425     VtkOutputModule<TracerTypeTag> vtkWriter2(*tracerProblem, *bulkFvGridGeometry, *gridVariables, sol[bulkIdx], tracerProble
426     using VtkOutputFields = typename GET_PROP_TYPE(TracerTypeTag, VtkOutputFields);
427     VtkOutputFields::init(vtkWriter2); //!< Add model specific output fields
428     vtkWriter2.write(0.0);
429
430
431
432     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
433     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
434     ////////////////////////////////////////NEW TRACER DIFFUSION////////////////////////////////////////////
435
436
437
438
439     // time loop
440     timeLoop->start();
441     while (!timeLoop->finished())
442     {
443         // set previous solution for storage evaluations
444         assembler->setPreviousSolution(oldSol);
445
446         // solve the non-linear system with time step control
447         nonLinearSolver.solve(sol, *timeLoop);
448
449         // make the new solution the old solution
450         oldSol = sol;
451         bulkGridVariables->advanceTimeStep();
```

Figure 15: The 1p-1p-tracer-tracer Code (4)

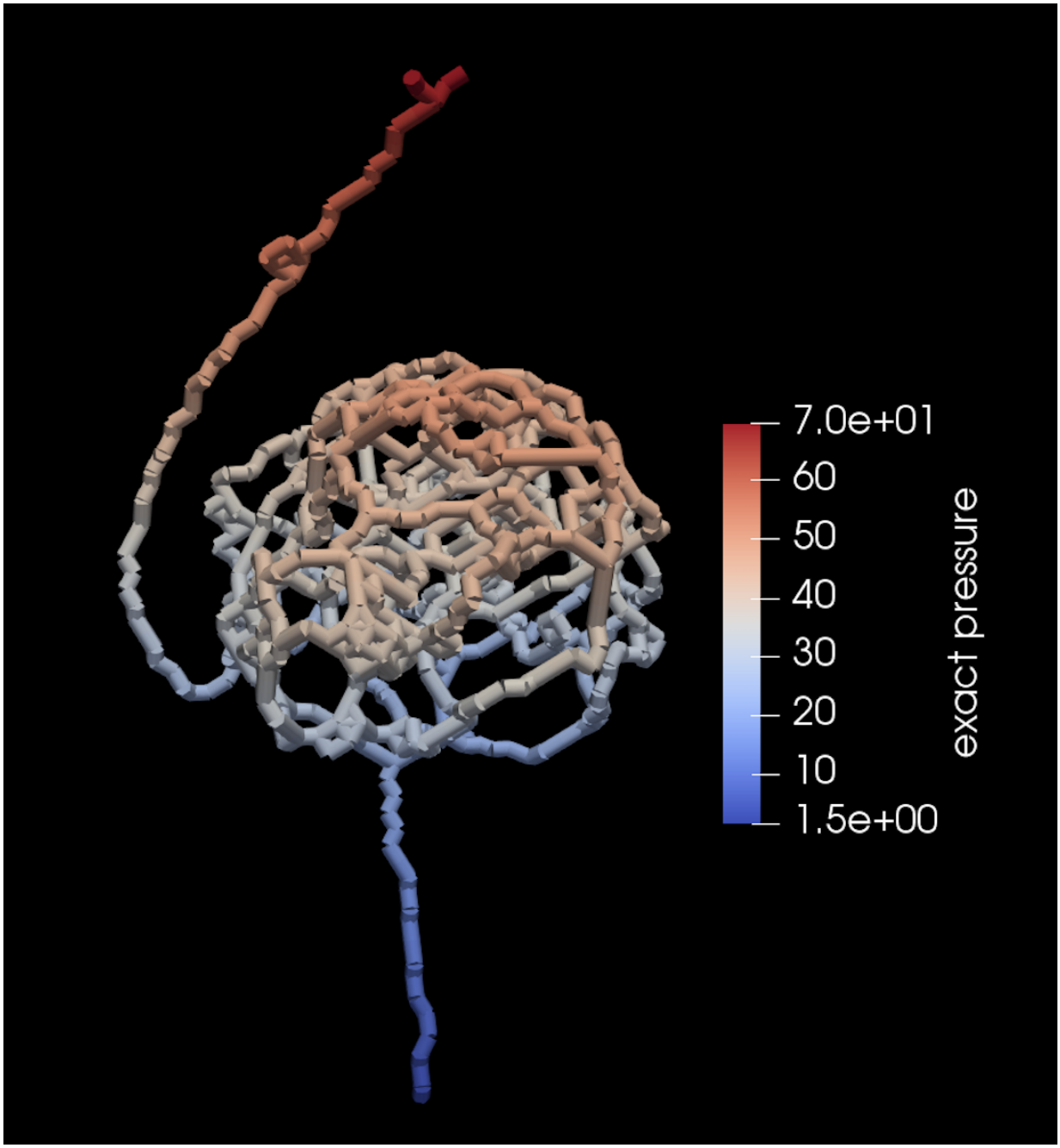

Figure 16: Pressure Field in Glomerulus 1

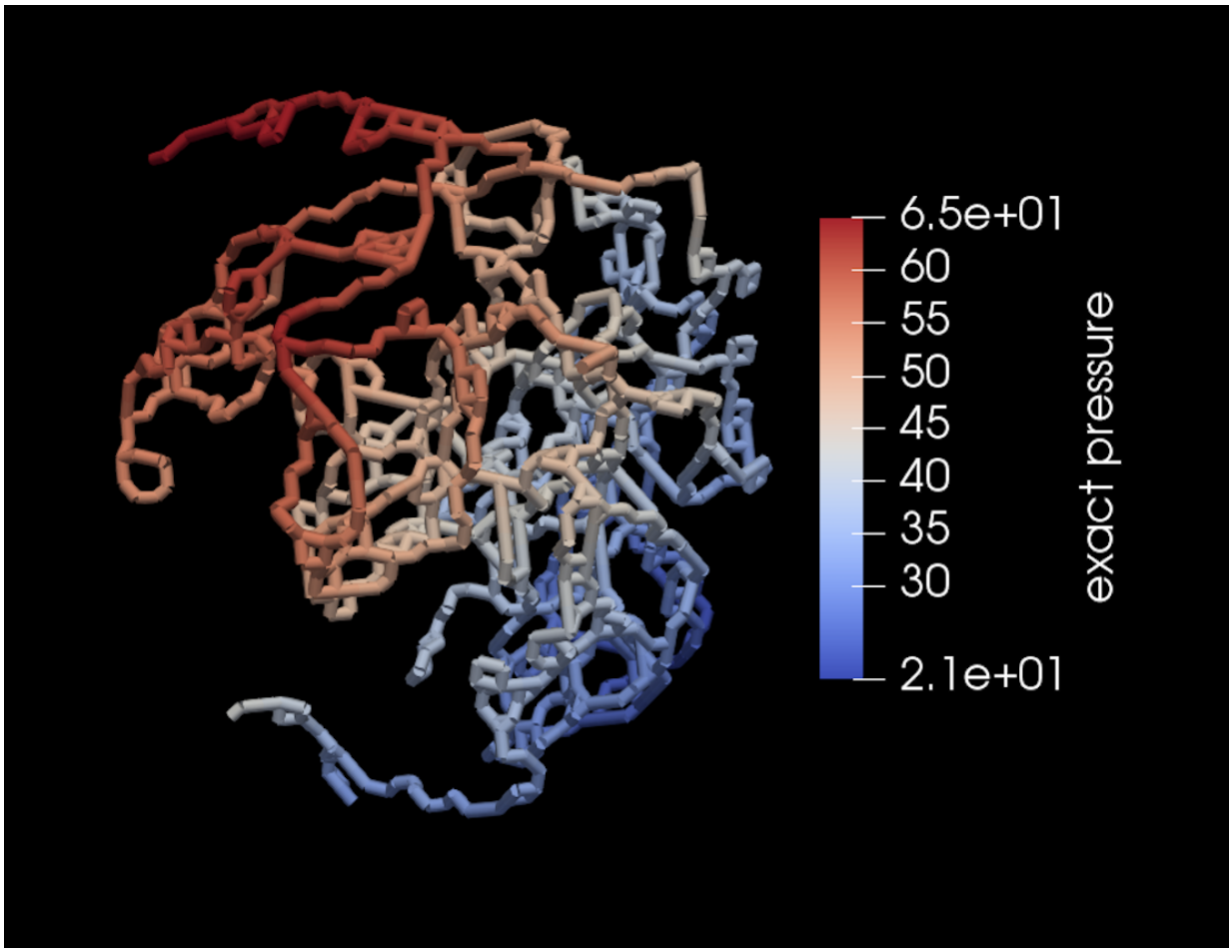

Figure 17: Pressure Field in Glomerulus 2

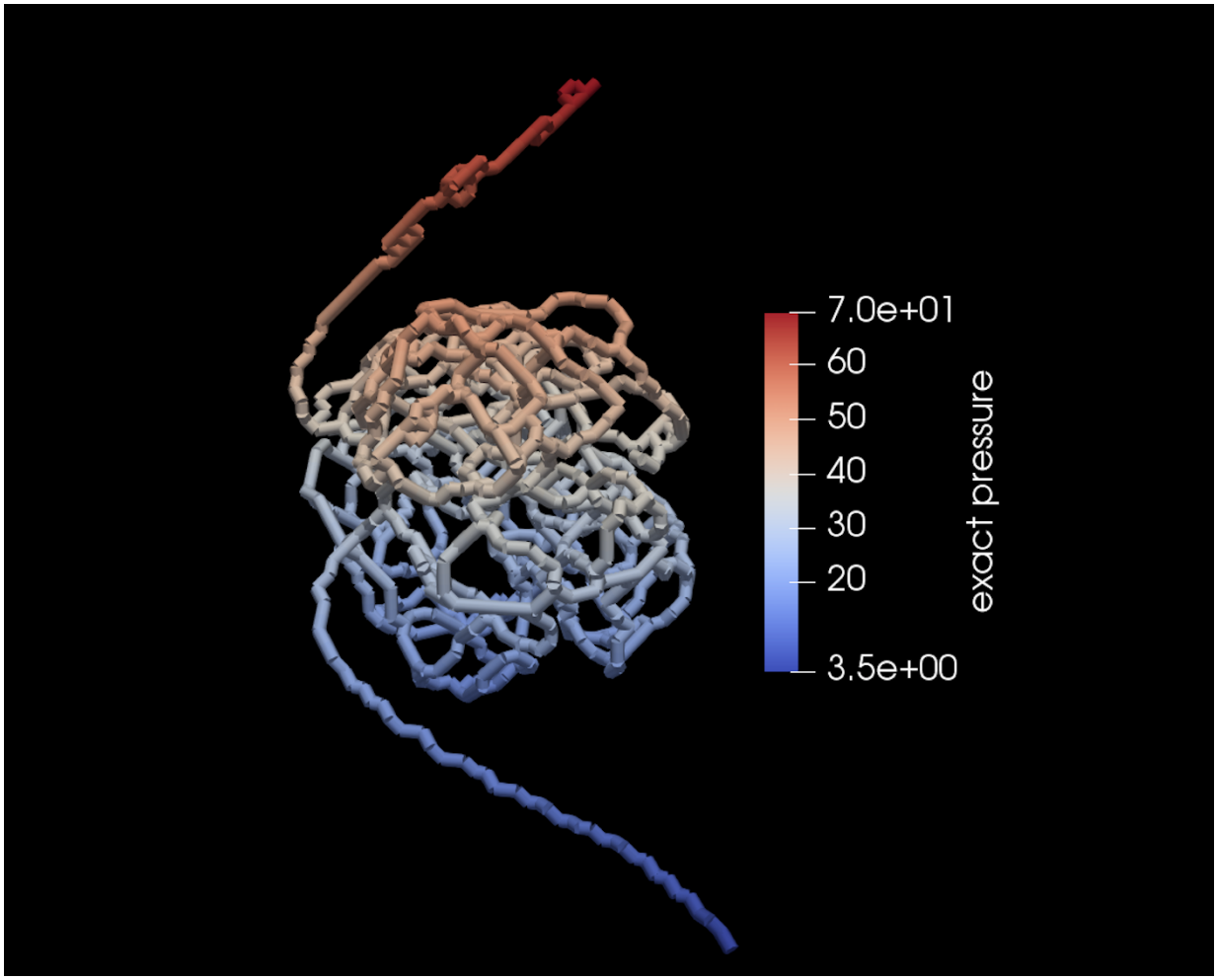

Figure 18: Pressure Field in Glomerulus 3

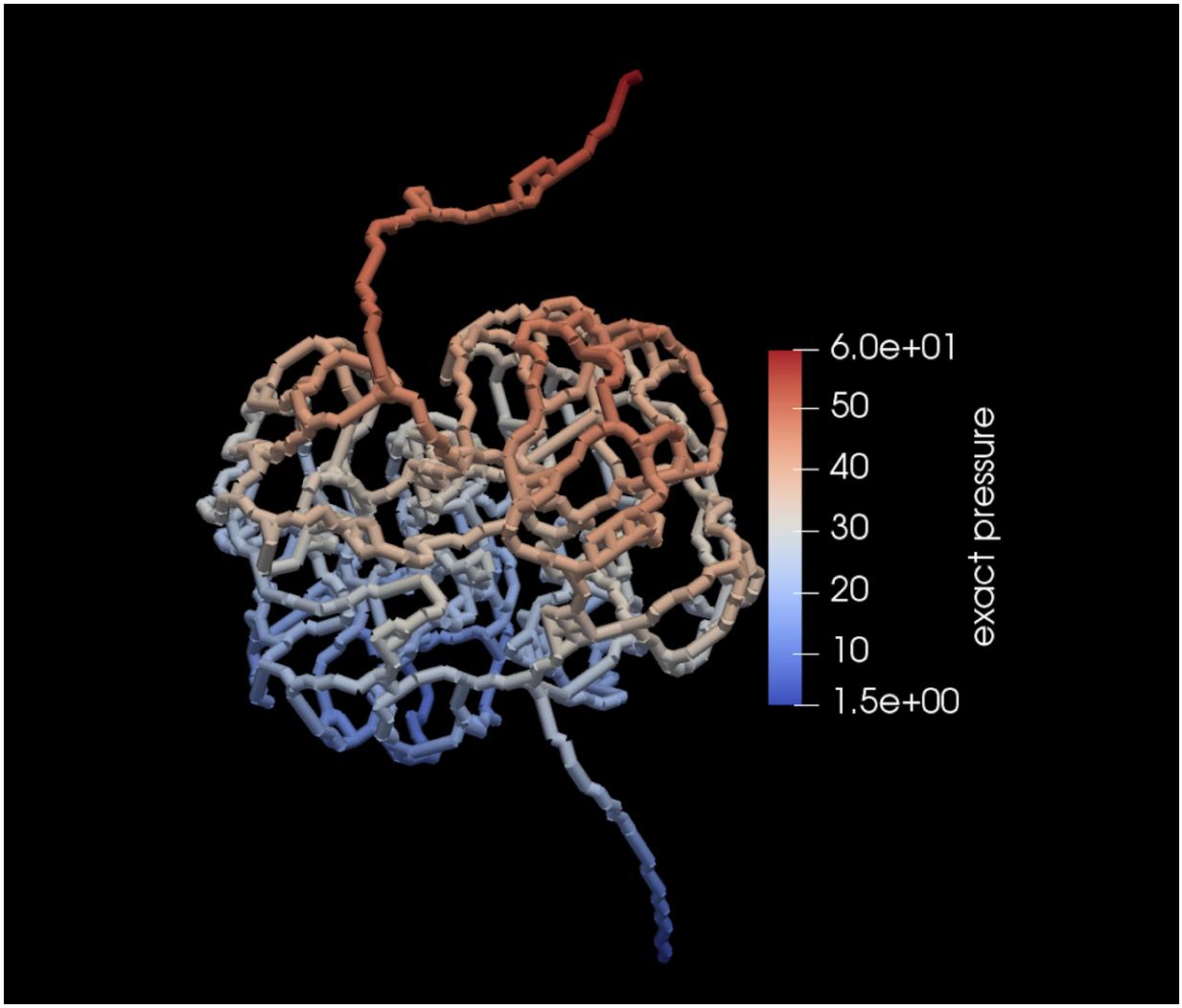

Figure 19: Pressure Field in a Glomerulus 4

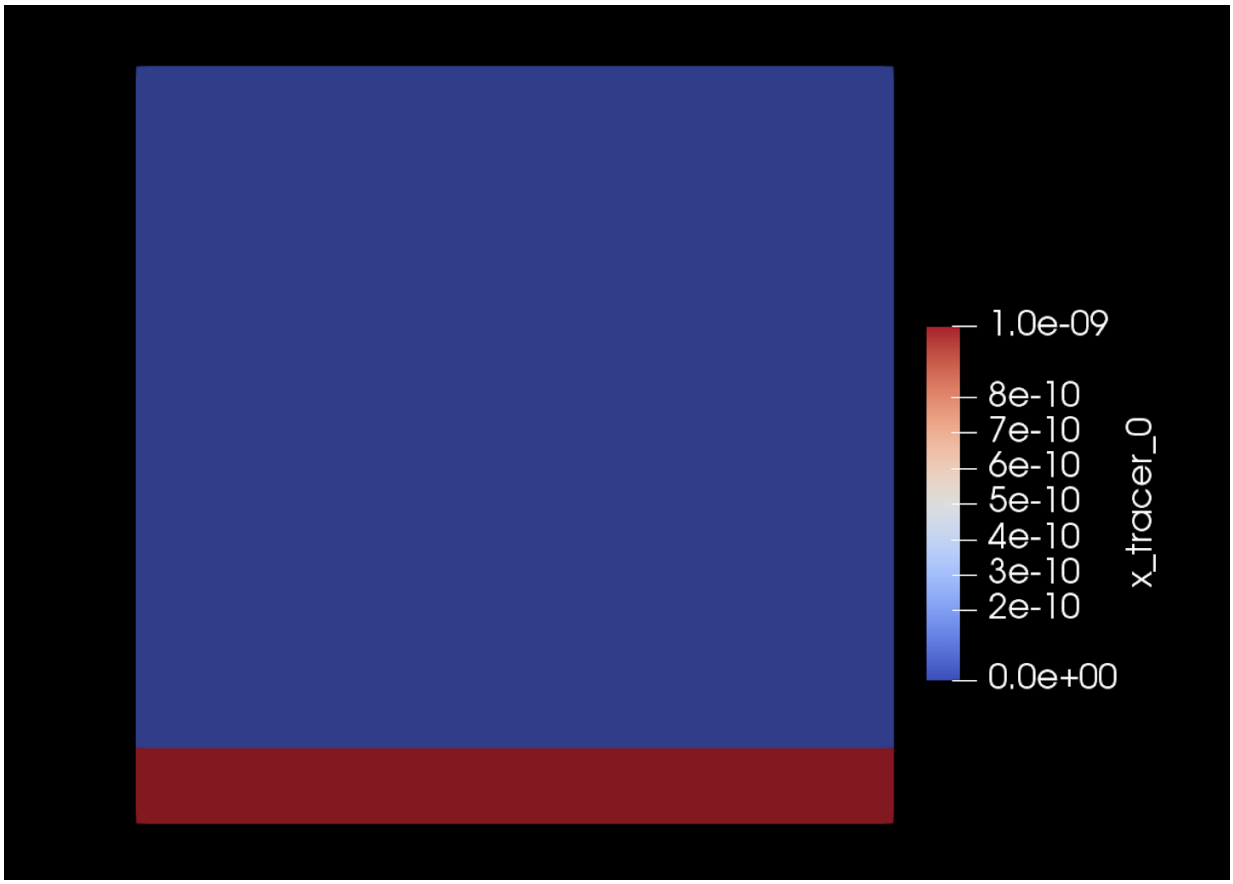

Figure 20: $DuMu^x$ Tracer Concentration Simulation for a Rectangular Geometry (Time Step 1)

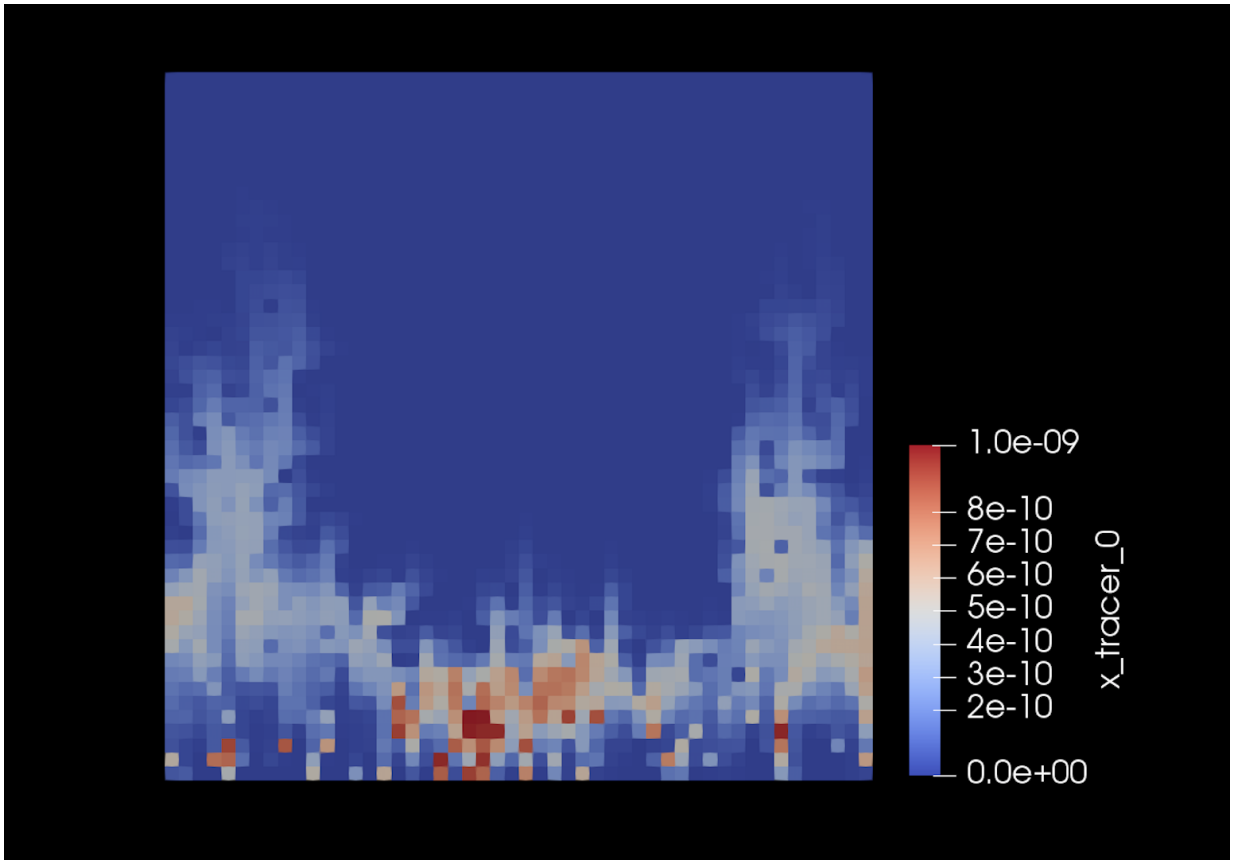

Figure 21: $DuMu^x$ Tracer Concentration Simulation for a Rectangular Geometry (Time Step 10)

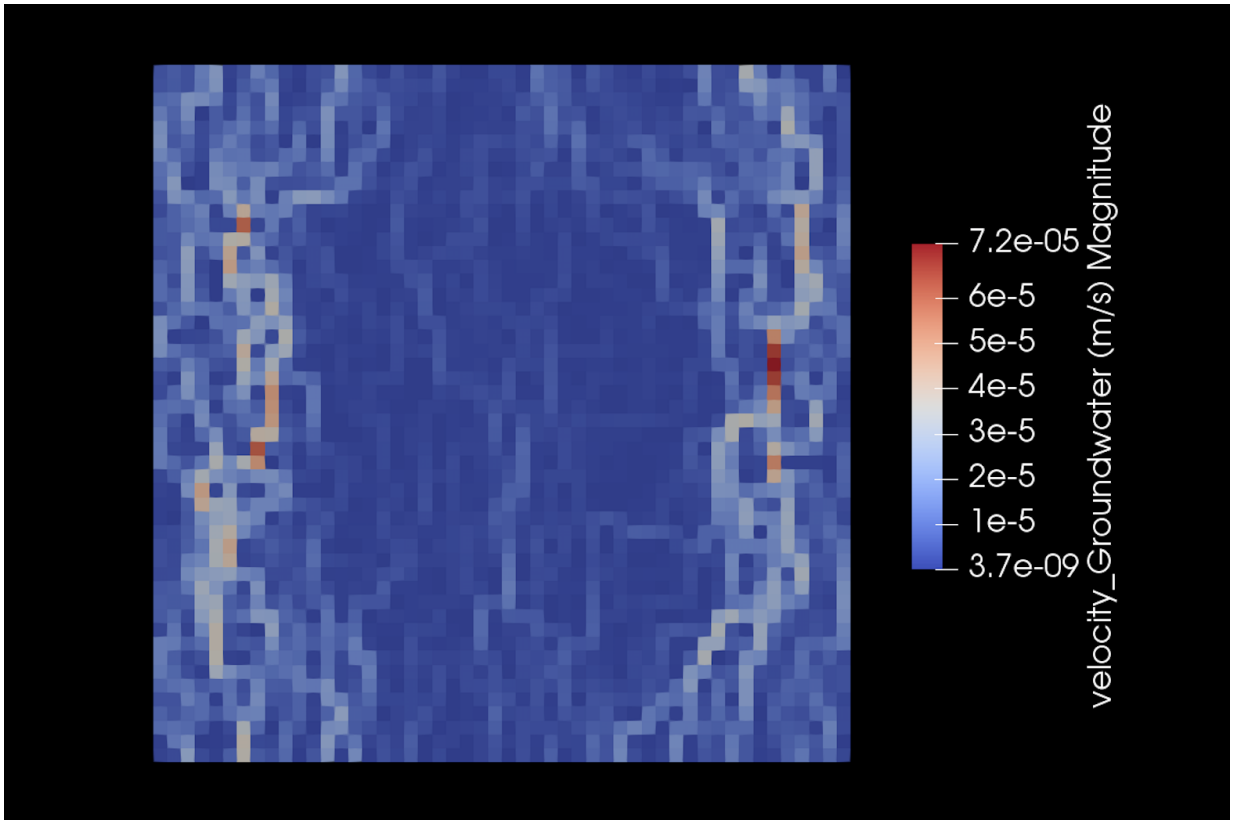

Figure 22: $DuMu^x$ Tracer Velocity Simulation for a Rectangular Geometry