DEVELOPMENT OF TOOLS TO ACCELERATE AND ADVANCE MODELING DISEASE PROGRESSION

By
STEVEN MACRAE RUGGIERO
Bachelor of Science in Chemical Engineering
Rowan University
Glassboro, New Jersey
2016

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
Dec, 2020

DEVELOPMENT OF TOOLS TO ACCELERATE AND ADVANCE MODELING DISEASE PROGRESSION

Dissertation Approved:

Ashlee Ford Versypt

Thesis Advisor

Heather Fahlenkamp

Christopher Fennell

Jindal Shah

ACKNOWLEDGMENTS

My friends, who have been there through my journey up to this point:

- Adam Jlelaty
- Rebecca Cargan
- John McGuire
- Connor Buckley
- Alex Beebe
- Emma Beebe
- Tina Al-Attar
- May "Corvimae" R.
- And all of the wonderful people I've met through D&D

My family, who's patience knows no limits:

- Parents: Holly and Louis Ruggiero
- Sister: Jean Ruggiero
- Grandparents: Graham and Jean Smith

My research group, who have learned alongside of me:

- Advisor: Dr. Ashlee Ford Versypt
- Temitope Benson
- Carley Cook
- Duncan Mullins
- Krutika Patidar

- Haryana Thomas
- Yen Nguyen Edalgo
- Minu Pilvankar
- Blake Bartlett
- Samantha Carpenter
- Kaitlyn Lane
- Cole Strickling
- Troy Adkins
- Albert Cai
- Lindsay Cordier
- Travis Diamond
- Natalie Evans
- Noah Gade
- Grace Harrell
- Caitlin Haug
- John Hayes
- Michele Higgins
- Cory Hopcus
- Kelcie Jenkins
- Jacquelyn Lane
- Matt Linna
- Zach Mauck
- Isabelle Posey
- Meredith Proctor
- Ashlea Sartin
- Susanna Stewart
- Claire Streeter

- Karley White
- Aubrey Wolfe
- Hui Ling Yong
- Anya Zornes

All of the wonderful teachers and professors I met during my times at Union County Magnet High School and Rowan University.

Special thanks to:

- Dr. Buchanan, who convinced me to study chemical engineering.
- Dr. Hesketh and Dr. Staehle, who convinced me to go to graduate school.

The medical professionals who helped me keep my life from falling apart due to ADHD:

- Kara Grace, Shawna Moore, and the rest of the staff at CBSO
- Rachael Lawler
- Melissa Andrade and the rest of the staff at Andrade Medical Services

The funding sources that enabled the research presented here:

- Oklahoma Center for the Advancement of Science and Technology, Project Number: HR17-057
- National Science Foundation, Award Number: 1845117
- National Institutes of Health, Project Number: R35GM133763

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

“The end of learning is the beginning of death.”

Eramir in Path of Exile by Grinding Gear Games

Name: STEVEN MACRAE RUGGIERO

Date of Degree: Dec, 2020

Title of Study: DEVELOPMENT OF TOOLS TO ACCELERATE AND ADVANCE MODELING DISEASE PROGRESSION

Major Field: CHEMICAL ENGINEERING

Abstract: Disease pathology emerges from complex interactions at multiple scales including the cellular level, tissue level, and organ level. Multiscale modeling seeks to capture emergent phenomena by coupling models of differing spatial and temporal scales. Multiscale modeling is a powerful tool for modeling disease progression, but this power comes at the cost of great complexity and development time. The primary objective of this thesis is to develop tools that help make multiscale models in less time. The tools developed in this work are a virtual kidney currently in development, a model converter, and a method for producing simulation geometry.

TABLE OF CONTENTS

Chapter Page

I Introduction **1**
1.1 The Significance of Study 2
1.1.1 Case I: A Generalizable Model 2
1.1.2 Case II: Generalized Tools 2
1.2 Publications Outside of Thesis Scope 3
1.2.1 Mathematical Modeling of Tuberculosis Granuloma Activation 3
1.2.2 Building a MATLAB Graphical User Interface to Solve Ordinary Differential Equations as a Final Project for an Interdisciplinary Elective Course on Numerical Computing. 5
1.3 Research Objectives 6
1.4 Potential Impact 7

II Establishing the Virtual Kidney **9**
2.1 Introduction 9
2.1.1 Motivation 9
2.1.2 The Virtual Kidney 11
2.1.3 Software Used 15
2.2 Agent Based Model 18
2.3 Solute Transport Model 20
2.4 Computational Fluid Dynamics Simulation 23
2.5 Simulation Results 27
2.5.1 Full Transport Model Results 27
2.5.2 Podocyte Model Validation 27
2.5.3 Time Scale Analysis 31
2.6 Conclusions 33

III SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter **34**
3.1 Background 34
3.2 Software Details 39
3.2.1 Summary 39
3.3 Tutorial 42
3.3.1 Interactive Jupyter Notebook 42
3.3.2 Creating a Python Model 42
3.3.3 Exploring a Newly Created Python Model 44
3.3.4 Simulating the Model 46

3.4 Conclusion 47

IV Procedural Generation Methods in Tissue Simulation 48
4.1 Background 48
4.1.1 Procedural Content Generation 48
4.1.2 Motivation 51
4.1.3 Methods Used in Literature 53
4.2 Generation of Cellular Geometry 56
4.2.1 Glomerulus Cross Sections 56
4.2.2 Generalizing the Algorithm 58
4.3 Conclusions 61

V Recommendations for Future Work 62
5.1 Aim I: Establishing the Virtual Kidney 62
5.2 Aim II: SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter 62
5.3 Aim III: Procedural Generation Methods in Tissue Simulation . . . 63

References 64

A Podocyte and Mesangial Cell Models 80

LIST OF TABLES

Table Page

2.1 Tissue Simulation Basis Measurements 19

2.2 Diffusion Coefficients And Radii 22

A.1 Pharmacodynamic parameter values for the podocyte local RAS model for benazepril 82

A.2 Parameters for equations (1.9) and (1.12). 83

LIST OF FIGURES

Figure | Page

2.1 Glomerulus Structure Illustration 10

2.2 Virtual Kidney Conceptual Model 12

2.3 Multiscale Model Guidelines 13

2.4 Tissue Model Initial Geometry 19

2.5 Computational Fluid Dynamics Simulation Geometry 24

2.6 Bowman's Space Mesh 26

2.7 Mesangium and Glomerular Basement Membrane Mesh 27

2.8 Flow Domain Velocity 28

2.9 Full Transport Results 29

2.10 Podocyte Model Results 30

2.11 Glucose Field Response 31

2.12 Ordinary Differential Equations Model Response 32

3.1 SBMLtoODEpy Overview 41

4.1 Rogue Level Generation 49

4.2 Tracks Generated Through Optimization 51

4.3 Early Simulation Reference 52

4.4 Early Simulation Geometry 52

4.5 Glomerulus Microscopy 53

4.6 Step By Step Geometry Generation 57

4.7 Geometry Variety Examples 59

4.8 Graph-based Geometry Generation Reference 60

CHAPTER I

Introduction

Many human diseases cause physiological and morphological symptoms evident at the tissue and organ level. However, these symptoms are the result of interactions at the cellular level, which often involves signaling proteins moving between cells. Furthermore, the actions of the cells are governed by biochemical networks contained inside the cells. Due to this coupling of biological systems at differing spatial and temporal scales, multiscale models are needed to properly predict the evolution of diseases.

Computational modeling is useful for creating multiscale models for multiple reasons. The most straight forward reason is that a computer can find a numerical solution to a highly complex system of equations that have no analytical solution or are time consuming to solve. Ordinary differential equations (ODEs) are good for modeling biochemical networks inside cells. Biochemical network models are typically complex enough to require a numerical solution. Additionally, computational modeling can employ methods such as agent based models (ABMs) that model individual objects or agents that act using a set of rules. A major advantage offered by the variety of models is that an appropriate model type can be matched to the model scale. This feature makes ABMs useful for modeling tissues. Beyond the variety of model types, computational modeling has a second advantage in that multiple models can be coupled together, e.g., the results of a time step of one model can be fed directly into another model. Multiscale models can be developed capitalizing on these two advantages to construct accurate computational models of realistic systems. Despite

these strengths, the biggest drawbacks of multiscale modeling are the complexity and development time required.

1.1 The Significance of Study

The theme of generalizable open source code that makes developing multiscale models easier and faster unifies the aims of this thesis.

1.1.1 Case I: A Generalizable Model

The first branch of this theme discusses the state of a comprehensive virtual kidney under development. The motivation of the virtual kidney is to explore unanswered questions about diabetic kidney disease (DKD) by starting from a healthy state and simulating the kidney's progression to a diseased state. Through modular code design, the disease mechanisms specific to DKD can be swapped out for mechanisms associated with other kidney diseases. As a result, one of the long term goals of the virtual kidney is to serve as a framework that can be generalized to other diseases. Chapter II presents Aim I of this thesis: Establishing the Virtual Kidney.

1.1.2 Case II: Generalized Tools

The second branch is the generalization of methods used to solve problems encountered in Aim I. These problems i) are encountered by other researchers that develop systems biology models, ii) have existing solutions that are insufficient for our needs, and iii) leave a gap in solutions that represents a niche to be filled.

Systems Biology Markup Language (SBML) [1] is a popular standard for storing biochemical models, with thousands available from the BioModels database [2, 3]. While developing the virtual kidney, I explored the possibility of coupling SBML models into our simulations. Multiple converters exist that turn SBML into a variety of programming and modeling languages, but Python [4] is not one of the options

available [5]. Other Python modules that handle SBML models do not suit the needs of the project. Given the use of Python as the programming language of choice in Aim I, its rising use in computational science, and a lack of SBML to Python converters, I created an open source SBML to Python converter. Chapter III presents Aim II of this thesis: SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter.

Early work on the virtual kidney focused on examining the effects of cellular dysfunction at the level of an entire glomerulus. Glomeruli are the initial sites of urine filtration in the kidney and suffer severe damage in a DKD patient. Due to variance in the geometry of cross sections in a single glomerulus, we wanted to produce a wide variety of geometries that represent the glomeruli. The methodology presented in Aim III incorporates the concepts used in game development to create a highly flexible algorithm for generating tissue geometry. Chapter IV presents Aim III of this thesis: Procedural Generation Methods in Tissue Simulation.

Chapter V discusses the future work needed to further these projects beyond their current state.

1.2 Publications Outside of Thesis Scope

I have authored two publications that have not been included in this thesis. The primary reason for their exclusion is that the work was either not generalizable or did not use novel computational techniques.

1.2.1 Mathematical Modeling of Tuberculosis Granuloma Activation

The first paper I was first author on [6] was written by Dr. Minu Pilvankar, Dr. Ford Versypt, and me. The work discussed in the paper added novel equations to a preexisting model of tuberculosis (TB) [7]. These equations modeled the regulation of matrix metalloproteinase-1 (MMP-1), collagen buildup, and bacterial leakage in

a TB granuloma. MMP-1, commonly referred to as collagenase, breaks down the collagen membrane of the granuloma. As the membrane deteriorates, containment of TB bacteria inside the granuloma fails, leading to an active infection.

My contributions to this work were in replicating the model from [7] and implementing the new equations into the model. Recreating the original model proved to be a difficult task. With 13 ordinary differential equations (ODEs), searching for even a single error proved to be arduous. This was compounded by the fact that the original model was not available. Once satisfactory implementation of the original model was completed, the new portion of the model was added into the code.

The other aspect of my work on this project was developing scripts to analyze the model. Our paper had several virtual experiments including a local sensitivity analysis, a gene deletion experiment, and an exploration of the effect of a key variable added to the model. The local sensitivity analysis searched for the model parameters that had the greatest effect on TB leakage out of the granuloma by running the model with a single adjusted parameter. This run was then compared to the results of a basis parameter set to quantify the effect of the adjusted parameter. Gene deletion provided insight into how a single part of biological system behaves with that part removed. For the gene deletion experiments, one of the ODEs in the model was set to 0, and the difference in model behavior was used to infer the role of the variable removed. Parameter k_C is the rate constant for the degredation of collagen. This parameter was specifically targeted for analysis because it was the novel parameter that had the greatest impact on bacterial leakage and was easy to use in tuning the rate of bacterial leakage when studying the effect of leakage on other model variables [6].

While the additions to the TB model were novel, the scripts developed for the project did not solve any new problems from a programmatic perspective. Numerical methods for solving ODE models is one of the oldest and most explored compuational

methods, and the virtual experiments were based on analyses of other models. While this publication falls outside of the scope of this thesis, the experience developing the code for the ODE model and its analysis gave insight used in Aim II. How SBML-toODEpy generates Python model implementations was directly influenced by the needs I had as a researcher during this project.

1.2.2 Building a MATLAB Graphical User Interface to Solve Ordinary Differential Equations as a Final Project for an Interdisciplinary Elective Course on Numerical Computing.

The second paper I helped author [8] was written by Jianan Zhao, Dr. Ford Versypt, and me. In this publication, we detailed a class project of developing a MATLAB [9] graphical user interface (GUI) for solving a system of ODEs. The project served as a capstone for an applied numerical computing class. Topics covered in the course included Python and MATLAB basics, parameter fitting, GUIs, and solving ODEs. The project required the MATLAB program to have a GUI and solve an arbitrary system of ODEs, which meant the project required mastery of most of the concepts covered in the class.

The project requirements stated that students can expect the equations to be entered into their program in an order such that each equation is only dependent on previous equations. This kept the complexity of the project down to a manageable level. My project submission was inspired by Polymath [10], a software program for common numerical methods used in chemical engineering education. One of the features of Polymath is that the equations can be entered in an arbitrary order, and the program will sort out the order to solve the equations. Due to a combination of ambition and sheer stubbornness, I wanted my submission to handle equations entered in an arbitrary order. This added back in the complexity Dr. Ford Versypt tried to keep out of the project.

Sorting the equations took two steps: parsing and then reordering the equations. Parsing the equations made use of regular expressions. Regular expressions are used to define text patterns. For this project, I defined patterns that I expected a user would enter for their equations. The regular expressions identified variable names, differentials, functions, and operators from entered text. When the parsing function finds a matching pattern, it breaks off the matching portion of text, interprets it, and then recursively calls itself with the remaining text as input. The final output of the function is an array that has the dependent variable, independent variable for ODEs, equation type, and a tree representing the variables and order of operations of the right hand side of the equation [8].

Each variable defined by an ODE has an associated initial value. Therefore, the columns associated with variables defined in ODEs are zeroed out immediately. Then an equation with a single 1 in its row is picked, and the column of the variable is zeroed out. This process repeats until no changes occur in the matrix after an iteration. If all elements are zero, the equations are properly defined, and the order of the rows is a valid equation order [8].

Like the TB model discussed in Section 1.2.1, the resulting software pushes no boundaries; it was an educational contribution, and the experience I gained was invaluable in developing Aim II. The most important module in SBMLtoODEpy is the function that generates code from model elements. In that function I rewrote the Gaussian elimination algorithm I developed in [8] to place **Assignment Rules** in the correct order in Aim II.

1.3 Research Objectives

Aim I: Establishing the Virtual Kidney.

- Task 1 - Incorporate a partial differential equations (PDEs) solver and ABM simulator.

- Task 2 - Add ability to incorporate ODEs generated from the results of Aim II.
- Task 3 - Develop PDEs to solve the transport of signaling proteins in the glomerulus.

Aim II: SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter.

- Task 1 - Interpret SBML files to produce intermediate text files.
- Task 2 - Generate Python models.
- Task 3 - Represent biochemical networks in mesangial cells, podocytes, and epithelial cells with ODEs.

Aim III: Procedural Generation Methods in Tissue Simulation.

- Task 1 - Develop a algorithm to generate simulation geometry.
- Task 2 - Parameterize the function to enable creation of varied geometries.
- Task 3 - Outline generalizing the algorithm.

1.4 Potential Impact

The majority of the work presented in this thesis is directly connected to the virtual kidney project. The first aim discusses development of the virtual kidney itself. The second and third aims solve problems encountered in the project. The most immediate and direct impacts of the aims of this thesis will be on DKD research; already Aim II was used to create the Python implementations of podocyte and mesangial cell models used in Aim I.

It is my hope that the aims of this thesis will be useful beyond the context of DKD research. As discussed in Section 1.1, I see the possibility for a wider impact of my work. Development of the virtual kidney will continue beyond my tenure in

Dr. Ford Versypt's lab, and the contributions I have already made will continue to shape its development. The success of Aim I will be measured by the time saved in further development of the virtual kidney and the applicability of my contributions to other forms of kidney research. As Python grows in use by computational modelers, the demand for research oriented Python packages will grow. While other SBML tools exist for Python, SBMLtoODEpy targets a niche unfilled by other packages at the time of writing. The success of Aim II will be measured by the flexibility that generated models provides to other researchers. Procedural generation offers a way to produce a robust variety of geometries with minimal researcher time. The methodology in Aim III incorporates an idea from outside of systems biology in an attempt to improve it. The success of Aim III will be measured by its impact on other researchers' ability to develop simulation geometries.

To me, no success is greater than my work benefiting other researchers.

CHAPTER II

Establishing the Virtual Kidney

2.1 Introduction

2.1.1 Motivation

Diabetic kidney disease (DKD) is the most common form of end stage renal disease [11, 12]. According to the 2019 report published by the United States Renal Data System, 46.9% of patients receiving hemodialysis, peritoneal dialysis, or a transplant for end stage renal disease had diabetes as the primary cause [13]. Its prevalence in the US population is increasing due to increasing prevalence of diabetes [12]. However, DKD is not just a US health issue; DKD incidence has been rising around the globe [11].

The glomerular tissues are the tissues key to understanding DKD. The glomeruli (plural of glomerulus) are structures in the kidney that filter waste products from the bloodstream. Figure 2.1.1 shows the structure and cross section of a glomerulus. Of note are mesangial cells and podocytes. The mesangium, comprised of the mesangial cells and their extracellular matrix (ECM), are located inside clusters of capillaries and provide structure for the glomerular tuft [14]. Podocytes envelope the capillaries with interwoven protrusions known as foot processes. The foot processes are linked via a protein mesh to create a size-exclusion filtration barrier known as the slit diaphragm or slit membrane [15].

Under chronically elevated glucose, the biochemistry inside the mesangial cells changes and leads to several structural changes [16]. The earliest structural changes

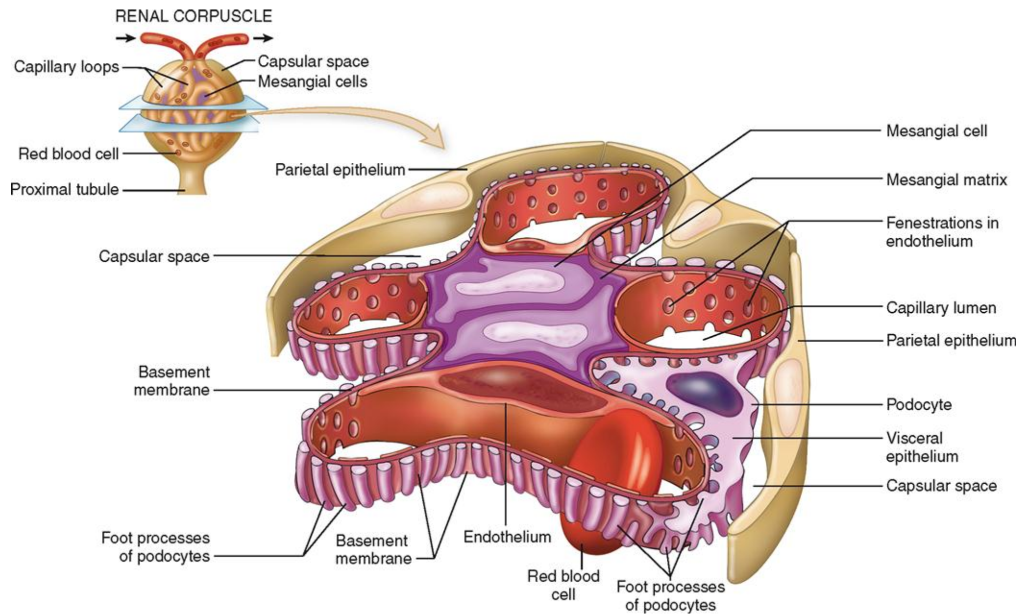


Figure 2.1: Insert shows a glomerulus and the magnified view shows a cross sectional slice of a glomerulus [14]

are the thickening of the glomerular, capilliary, and tubular basement membranes. The next set are mesangial swelling, breaking of linkages between podocytes, and loss of endothelial fenestrations [11]. The process of podocytes losing the interdigitating pattern of their foot processes is known as effacement, which leaves gaps in the filtration barrier [17, 18]. Parallel to effacement is podocyte dedifferentiation. When podocytes dedifferentiate, they lose the ability to form the linkages between foot processes, which in turn further complicates maintenance of the slit diaphram. The net result is that due to the loss of podocytes, proteins are able to penetrate the glomerular filtration barrier and pass into the urine, termed proteinuria [18].

The two primary metrics used in diagnosing DKD are the estimated glomerular filtration rate (eGFR) and the presence of albumen in the urine, also known as albuminuria [11, 19]. Both measures have problems associated due to uncertainty inherent to producing the measure. The uncertainty with measuring eGFR comes from a high degree of imprecision in the equations used in calculating eGFR. With measuring albuminuria, the uncertainty lies in the day to day variation of the concentration of

albumin present in a patient's urine [19]. Not only is there uncertainty in the measures themselves, but also in the relationship between these measures and tissue level damage [11, 19]. Ultimately, we lack a diagnostic tool that is able to detect DKD early enough and reliably enough [20].

Various modeling, data analysis, and computational techniques have been employed in the field of systems biology to gain a better understanding of DKD. These run the range from organ and multiorgan models, data analysis of data sets produced by omic (genomic, epigenomic, proteomic, etc.) experiments, and tissue level models focusing on the kidney itself [20]. These tissue level models have considered transport in the kidney [21, 22, 23, 24, 25, 26], the efficacy of the kidney as a filter [27, 28, 29], renal bloodflow [30, 31, 32, 33], and the progression of renal interstitial fibrosis [34]. One of the open questions on DKD we seek to answer is "What are the interactions and temporal relationships among podocyte, mesangial, and endothelial cell injuries" [20]?

2.1.2 The Virtual Kidney

The work presented in this chapter is the development of a model that seeks to answer questions about how DKD progression occurs. The overarching goal of the larger project in our lab is to develop a comprehensive virtual kidney. The conceptual model used in developing the virtual kidney is shown in Figure 2.2. This conceptual model leans heavily on two concepts multi-scale modeling and modular code design.

Multiscale modeling analyzes interacting processes that lead to emergent behavior. For example, changes in behavior at the cellular level can cascade into structural changes at the tissue and organ level. Central to the development of multiscale models is that an appropriate model type can be matched to the scale of a process. Figure 2.3 shows different types of models and the biological scales they are effective at. Agent based models (ABMs) are useful for modeling tissues, including the interaction be-

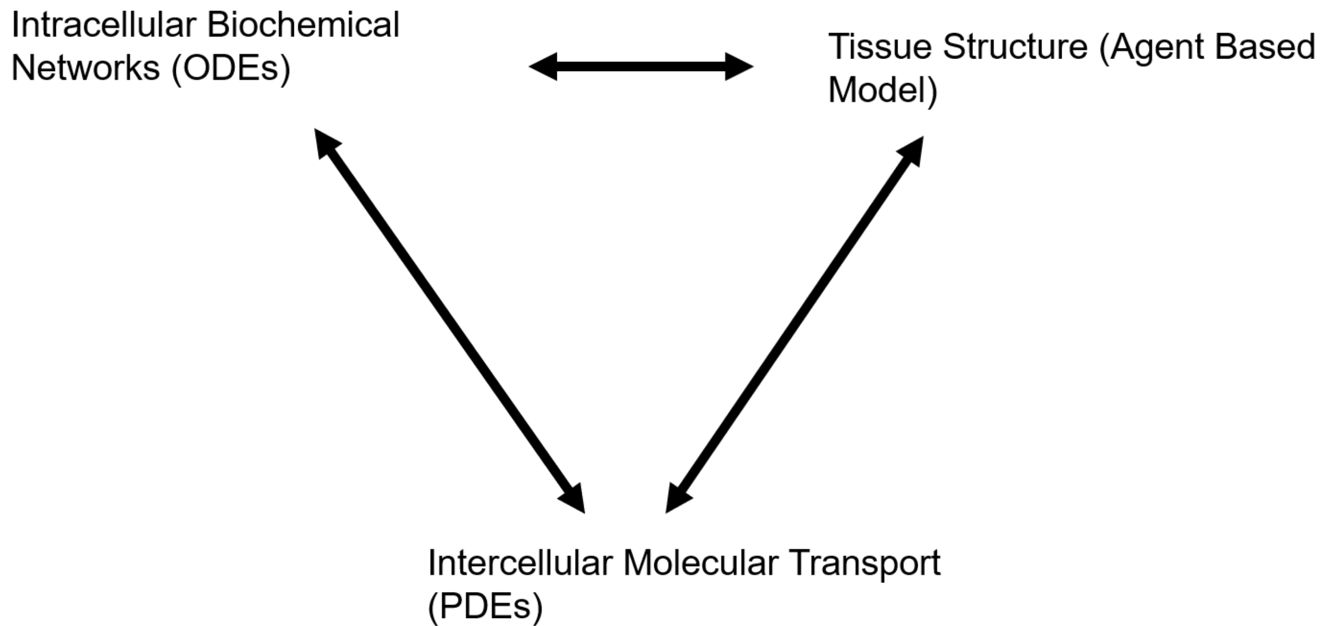


Figure 2.2: The conceptual model used to develop the virtual kidney. Each of the three model types used focuses on a set of phenomena. The scale of the partial differential equations and the agent based model are the same, but the processes they model can not effectively be modeled by the other. The ordinary differential equations scale is that of a single cell. Each model is separate from each other, and coupling the models together is done using each model's interface.

tween swelling mesangium and podocyte effacement. Ordinary differential equations (ODEs) on the other hand are better for modeling biochemical networks inside cells [35]. Multiscale modeling requires coupling together multiple models. For example, the results of an ODE model representing the biochemical networks of a cell can be coupled to the behavior of an agent representing a cell in an ABM.

In modular software design, pieces of a software system that can operate independently should have their implementation separate from the rest of the system, and communicate using a standardized interface [36]. For a module containing an ODE model, the functions that compute differentials and the algorithm used to integrate the equations are part of that model's implementation. If an ABM coupled to and ODE model is independent of how the ODEs are solved, the interface of the ODE model contains two parts: a function that signals the ODE model to step forward

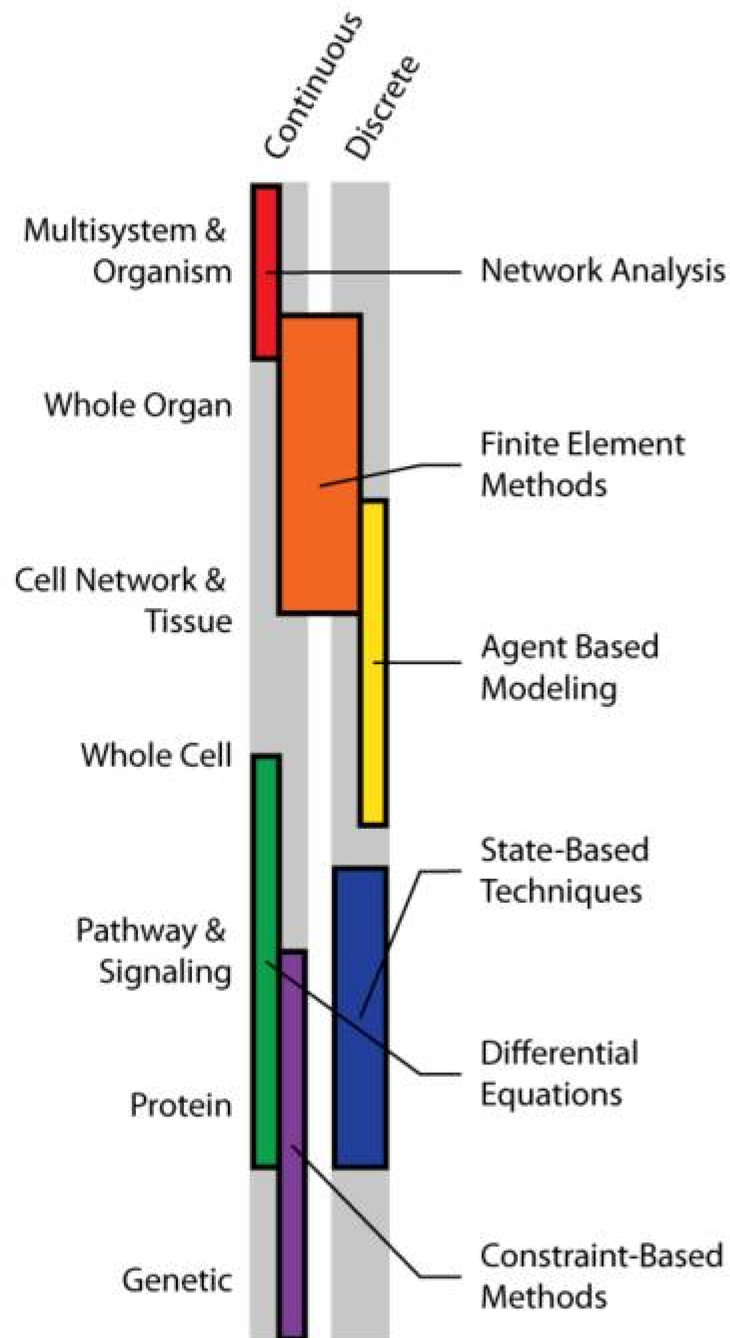


Figure 2.3: Recommended guidelines for appropriate modeling technique based on system scale [35]. While some modeling types may be used in more than one scale, no single type covers the range from the pathway and signaling scale to the tissue scale. Bridging this gap may be done by coupling multiple models together.

in time, and the object members that contain the model solution at the new time. Improvements or complete redesigns of how an ODE model is solved can be made

without considering how other modules are implemented. Or more abstractly, implementation changes that do not affect the module's interface can be made without impacting how other modules function [36].

Modular design principles not only make coupling the models of different scales together easier, they allow generalizing the virtual kidney beyond analysis of DKD. The virtual kidney concept seeks to start from a healthy kidney state and progress towards a diseased state. The diseased state modeled depends on the mechanisms of disease progression implemented in each model. Standardized interfacing enables incorporating models of other types of kidney disease with minimal researcher time invested.

The main factors that drive DKD progression that the model considers are the effects of hyperglycemia on the biochemistry within glomerular cells, how dysregulation in cells propagate to other cells through signaling proteins, and the tissue level damage that emerges from these cells. The biochemistry of interest has two parts. The first is the renin-angiotensin system (RAS) local to the podocytes and mesangial cells [37, 38]. The RAS model captures the dynamics of angiotensinogen (AGT), angiotensin-I (ANG-I), angiotensin-II (ANG-II), and renin. The second is upregulation of transforming growth factor beta 1 (TGF-β1) in mesangial cells that results from dysregulation of the RAS [39, 40, 41, 42, 43]. Elevated levels of TGF-β1 cause inhibition of matrix metalloproteinase-2 (MMP-2) [44]. MMP-2 is responsible for breaking down the extracellular matrix in the mesangium. Two ODE models were developed to capture the effects of glucose on the biochemical networks involved. One model was for the podocytes ((1.1), (1.2), (1.4), and (1.5) in Appendix A), and the other captured the mesangial cells ((1.1), (1.2), (1.4), (1.5), (1.9), and (1.12) in Appendix A). These models were developed by my labmates Dr. Minu Pilvankar in her PhD studies [45, 46, 47] and Ashlea Sartin in her undergraduate research project. The ODE models for the podocytes and mesangial cells that they

developed for our collaborative project are summarized along with their associated parameters in Appendix A. These models are coupled to a solute transport model to capture intercelluar signaling and a tissue structure model to capture emerging tissue damage.

2.1.3 Software Used

CompuCell3D was used to develop the tissue structure model used in this project. At its core, CompuCell3D uses the Glazier-Graner-Hogeweg (GGH) model to drive time evolution of a simulation [48]. The GGH model was developed as an extension of the Potts model [49, 50]. Classically, the Potts model examines a system of spin states in a lattice. The lattice is subdivided by a regular shape, for example into squares, with a spin state contained in each square. The number of unique spin states available depends on the system; a Potts model can be made for a system with an arbitrary number of available spin states. The spin states interact with neighbors, and each interaction has an energy associated with it based on the states of the two spins. The energy of the entire system is the sum of all interactions [51, 52].

In the GGH model, the concept of spin states is equated to cell indexes. Each unique spin state is directly associated with a cell, and a cell occupies the areas of the lattice containing that spin state. The next conceptual modification is the introduction of cell types. Each cell has a cell type, and the energy calculations are based on cell types instead of spin states [49, 50, 48]. It is important to note that in the conceptual framework of the GGH model, the effective energies calculated do not represent physical energy [48].

A classical example of an energy function for a GGH model can be abstractly viewed as

$$H = \sum \text{Contact energy between cells} + \sum \text{Volume constraint energy.} \tag{2.1}$$

More technically, the equation is [49, 50, 48]

$$H = \sum_{neighbors \vec{i}, \vec{j}} \mathbf{J}(\tau(\sigma_{\vec{i}}), \tau(\sigma_{\vec{j}}))(1 - \delta(\sigma_{\vec{i}}, \sigma_{\vec{j}})) + \sum_{\sigma} [\lambda_{vol}(\sigma)(\nu(\sigma) - V_t(\sigma))^2]. \tag{2.2}$$

The first sum is iterated over all interfaces of the lattice. $\mathbf{J}$ is an NxN matrix containing the contact energies for each possible pairing of N cell types and is indexed by the cell types, τ, of the cells in positions $\vec{i}$ and $\vec{j}$, $\sigma_{\vec{i}}$ and $\sigma_{\vec{j}}$ respectively. The contact energy is multiplied by term that zeroes out the energy if $\sigma_{\vec{i}}$ and $\sigma_{\vec{j}}$ are the same cell. The Kronecker delta is

$$\delta(i, j) = \begin{cases} 0 & : \quad i \neq j \\ 1 & : \quad i = j \end{cases} \tag{2.3}$$

which makes the term $(1 - \delta(\sigma_{\vec{i}}, \sigma_{\vec{j}}))$ zero when the same cell occupies $\vec{i}$ and $\vec{j}$. The second sum is iterated over all cells in the simulation. The parameter $\lambda_{vol}(\sigma)$ is the resistance of the cell, σ, to deviation from the cell's target volume, $V_t(\sigma)$. The cell's energy contribution is proportional to the square of the difference between its volume, $\nu(\sigma)$, and its target volume [48].

On top of this style of energy calculation, CompuCell3D uses a Monte Carlo method to move a model forward in time. Despite Monte Carlo steps traditionally having no connection to time stepping, GGH models have empirically shown to have Monte Carlo steps directly proportional to time steps in biological applications [48]. The random move used in the Monte Carlo method is spin flipping. CompuCell3D selects a lattice site and one of its neighbors. Provided that the sites are occupied by different cells, the neighbor is set to be occupied by the cell in the first site chosen. The energy difference between the two states, ΔH, is computed and used to calculate a probability of acceptance [48]. The acceptance function used is the Boltzmann

acceptance function [48, 53]

$$P(\Delta H) = \begin{cases} 1 & : \quad \Delta H \leq 0 \\ e^{-\frac{\Delta H}{T_m}} & : \quad \Delta H > 0 \end{cases} \tag{2.4}$$

where T_m is a parameter that acts to modify the rate of acceptance of a Monte Carlo step [48]. While T_m plays a functionally similar role to temperature, it does not represent the actual temperature in a model. Additionally, since both ΔH and T_m are in arbitrary units, this acceptance function can neglect the Boltzmann constant, which relates temperature and energy. A single Monte Carlo step attempts a random spin flip and accepts or rejects the result a number of times equal to the total number of lattice sites [48].

CompuCell3D's energy calculations are not limited to just contact energy and volume. It supports energy terms that can drive chemotaxis, apply surface area constraints to cells, create links that represent physical connections between cells, and more [54]. This model creates a changing geometry that is not captured by multiphysics or computational fluid dynamics software. Further, CompuCell3D allows users to incorporate Python [4] scripts into their simulations using an application programming interface [48, 55]. This enables coupling a tissue model developed in CompuCell3D with an arbitrary Python based model. While CompuCell3D does have PDE solvers built in, they primarily use the forward Euler method [54]. For this project, the forward Euler method does not provide enough numerical stability or accuracy when handling convection terms. Due to these issues, the transport model was developed using the partial differential equation (PDE) package FyPi [56].

FiPy is a Python based PDE solver that uses the finite volume method (FVM) to solve equations of the form

$$\frac{\partial \rho \Phi}{\partial t} - \left[\nabla \cdot (D_i \nabla)\right]^n \Phi - \nabla \cdot (\mathbf{u}\Phi) - S_\Phi = 0 \tag{2.5}$$

where Φ is the solution variable and the terms from left to right are the transient term, the diffusion term, convection, and source terms. In the transient term, ρ is an arbitrary rate factor. In the diffusion term, D_i, is the diffusivity of Φ and can be either a scalar if the diffusivity is isotropic or a matrix in anisotropic cases. The variable n is the order of the diffusion term. In the convection term, $\mathbf{u}$ is a vector field of a velocity moving Φ. The last term S_Φ abstractly represents processes like reaction or decay that directly add or remove Φ at a location. S_Φ can be a function of Φ. FiPy provides the ability to solve equations like (2.5) by implementing the four terms, meshes, variables, and solvers as Python classes [56]. This means that FiPy has done the majority of the low level coding needed to set up and solve PDEs. The first major advantage of FyPi is that it allows for the rapid development of scripts to solve PDEs without the user getting caught up in the difficult work of implementing the FVM. Since FiPy was developed with the approach of solving arbitrary equations, the resulting Python package is extremely flexible. FiPy offers support for more types of terms than the ones that appear in (2.5) that are easily added into an equation. The classes provided by FiPy are flexible in a way that allows the same problem to be solved in multiple ways, and these classes support coefficients that vary over space, time, or with respect to another solution variable in a system of PDEs [57].

2.2 Agent Based Model

While we are interested in how cellular dysregulation propagates to tissue damage at the scale of the whole glomerulus, we focused on a smaller scale to maintain a finer resolution. Here we investigated a region just large enough to capture the effects of the bloodstream glucose on the mesangial cells, podocytes, and glomerular basement membrane. To this end, the glomerular tissue simulation models a small section of the mesangium adjacent to a capillary. The initial conditions for this simulation are visualized in Figure 2.4, and the measurements used to design the geometry are in

Table 2.1.

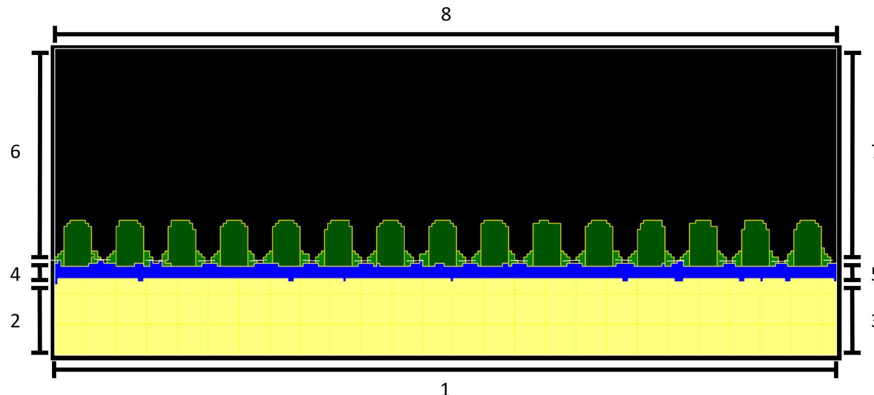


Figure 2.4: Visualization of the tissue model after the first time step. CompuCell3D produces the first image of the simulation after the first Monte Carlo step rather than at the initial condition. The yellow cells represent the mesangium, the blue cells represent the glomerular basement membrane, and the light and dark green cells represent the foot processes. The light green portions of the foot processes serve as anchors for the links between the foot processes. The black area is the medium, which represents the Bowman's space. The numbered labels correspond to the boundaries of the domain.

Table 2.1: Measurements used in designing the initial conditions of the CompuCell3D simulation. A basis of 0.02 µm per pixel length is used in our simulations.

Parameter	Value	Units	Pixel Lengths	Sources
Mesangium width	5	µm	255[1]	[23]
Mesangium height	0.5	µm	25	[23]
GBM thickness	0.1	µm	5	[23, 58]
Foot process width	0.29	µm	15	[59]
Foot process height	0.3	µm	15	[58]
Nephrin width	390	Å	2	[27, 60]
Simulation space height	-	-	100	-

[1] 250 pixel lengths would capture this measurement better, but 255 was chosen because it is an integer multiple of 17 (foot process width + nephrin width).

2.3 Solute Transport Model

The concentration fields in the simulation are governed by

$$\frac{dC_i}{dt} = \nabla \cdot [D_{i0} K_{Di}(x,y)] \nabla C_i - \nabla \cdot [\vec{v}(x,y) K_{Ci}(x,y)] C_i + f_i(\vec{p}[x,y], \vec{C}) \qquad (2.6)$$

where C_i is the concentration of species i, D_{i0} is the diffusivity of species i through the blood plasma, K_D is diffusive hindrance due to transport through porous media, $\vec{v}$ is the velocity field of the blood plasma, K_C is the convective hindrance of porous media, f_i is a function that solves the biochemical model developed for podocyte cells, $\vec{p}$ is the parameters of the biochemical model, and $\vec{C}$ is the concentrations of all species. The generation and consumption of signaling proteins for each foot process was adapted from the podocyte model described in Appendix A. Inside of the podocytes, $\vec{p}$ contains the appropriate values for the podocyte model. Outside of the cells, the value for each parameter is set to zero, aside from parameters that appear in denominators to avoid divide by zero errors.

Boundary 1 (Figure 2.4) is at the bottom of the mesangium and is assumes to be the interface at an mesangial cell membrane. For glucose, it is assumed to be a no flux boundary. For the other solutes, the concentrations at this boundary are determined by the mesangial cell model. A copy of the mesangial cell model (Appendix A) is instantiated for each face along the bottom. The x dimension is 255 pixels long. This means a total of 255 copies of the mesangial cell model. The glucose concentration for each model is sampled from the center of the pixel adjacent to the model's face. For each time step, the ODE models are progressed forward before solving the PDEs. At each face, the concentration of the non-glucose solutes are set to the corresponding instance of the mesangial cell model. Boundary 2 is at the left side of the mesangium where the mesangium contacts a capillary. The concentration at the capillary bound-

ary is set to bloodstream concentrations for all species. Currently the bloodstream concentrations for solutes other than glucose are assumed to be governed by the same model used for the mesangial cells. This assumption is unrealistic but usable to show the model's current functionality. The right side of the mesangium and both sides of the GBM are boundaries 3, 4, and 5. These boundaries are set as no flux conditions. While these boundaries are symmetry boundaries, we assume that any local fluxes are parallel to the boundaries. Boundaries 6, 7, and 8 for the Bowman's space are set to $\vec{n} \cdot \nabla C_i$, where $\vec{n}$ is the unit vector normal to the face. When the gradient at the boundary is explicitly constrained, FiPy treats the boundary as an inlet or outlet condition as appropriate for the convection terms [57].

Currently, the parameters for energy calculations are arbitrarily chosen. Our end goal is to couple the transport and biochemical models with the parameters used for effective energy calculations in CompuCell3D. The effective energy parameters drive the behavior of the cells and we hypothesize that the coupling will be able to capture the physiological changes of tissues due to dysregulation of biochemical networks. Here, the scope is limited to looking at the chemical crosstalk and transport between cells. In addition to serving as a basis for future inquiry, CompuCell3D is used to represent the cells as physical objects in our simulation space and to visualize solute concentration fields.

For glucose, D_0 was calculated using the Stokes-Einstein equation,

$$D_0 = \frac{k_B T}{6\pi \mu_{solvent} a_i} \tag{2.7}$$

where k_B is the Boltzmann constant, T is the temperature in Kelvin, $\mu_{solvent}$ is the viscosity of the bulk fluid, and a_i is the hydrodynamic radius of the molecule. For the other solutes, their D_0 and a_i were computed using HYDROPRO [61]. HYDROPRO starts its evaluation of hydrodynamic properties by constructing a geometric model from a protein structure. The geometric model is used to simulate the interactions

between the protein and the solvent [61]. HYDROPRO requires the partial specific volume of the protein as input. This measure is not readily available for the species in our models, so it was assumed to be equal to the inverse of density. Data on density are not readily available either, so the density was approximated from the correlation [62]

$$\rho = 1.410 + 0.145e^{-\frac{MW}{13}} \tag{2.8}$$

where ρ is the density in g cm^{-3}, and MW is the molecular weight in kDa. The values of D_{i0} and a_i used in the simulation are in Table 2.2.

Table 2.2: Values of D_{i0} and a_i used for each species. The value of a and method for computing D_0 of glucose are from [23]. For the other species, [62] and [61] were used to compute D_0. The sources for the protein structure files are listed.

Species	D_0	a	Sources
Glucose	853 μm^2 s^{-1}	3.8×10^{-4} μm	[23]
ANG-I	297 μm^2 s^{-1}	1.09×10^{-3} μm	[63]
ANG-II	307 μm^2 s^{-1}	1.05×10^{-3} μm	[63]
AGT	103 μm^2 s^{-1}	3.15×10^{-3} μm	[64]
Renin	91.1 μm^2 s^{-1}	3.22×10^{-3} μm	[65]
TGF-β1	121 μm^2 s^{-1}	3.62×10^{-3} μm	[66]
MMP-2	89.5 μm^2 s^{-1}	2.67×10^{-3} μm	[67]

The value of K_D depends on the medium through which a solute diffuses through and the hydrodynamic radius of the solute. Three media are present in the simulation: the mesangium, glomerular basement membrane, and the Bowman's space. The Bowman's space is not modeled as a porous media, therefore K_D is 1 in the Bowman's space. In the mesangium and glomerular basement membrane, each species has a different value of K_D. K_{Di} is calculated by [23]

$$[K_{Di} = e^{-\pi\phi^b} e^{-0.84 f^{1.09}} \tag{2.9}$$

$$b = 0.174 \ln\left(\frac{59.6}{\lambda_i}\right) \tag{2.10}$$

and

$$f = \left(1 + \lambda_i^2\right) \phi] \tag{2.11}$$

where ϕ is the fraction of volume occupied by fibers in the matrix and λ_i is the ratio of solute radius to fiber radius [23]. For regions occupied by the foot processes, D_i is set to 0 inside of the foot processes except for the pixels where the foot process contact the glomerular basement membrane. This approximates foot processes only secreting into and uptaking from the GBM.

K_C is the convective hindrance caused by a porous media on the solute [68]. The results presented in this chapter uses the same assumptions as the model of [23]; the mesangium has no effect on convection, and the glomerular basement membrane is a perfect filter. This equates to $K_D = 1$ in the mesangium and $K_D = 0$ in the glomerular basement membrane.

2.4 Computational Fluid Dynamics Simulation

The velocity field in (2.6) is solved using a 2D computational fluid dynamics simulation in Star-CCM+ [69] that models the flow through the mesangium, glomerular basement membrane, and Bowman's space. The geometry used in this simulation is shown in Figure 2.5. The properties of the porous media used in the simulation are the same as the work of Hunt et al. [23].

The mesangium and glomerular basement membranes were modeled as porous media, and the flow through the Bowman's space was modeled as laminar flow. For flow through porous media, pressure and velocity are related using

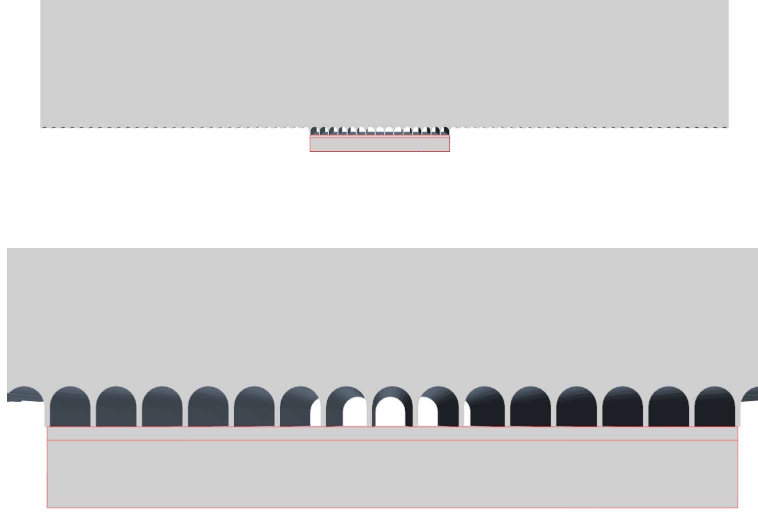


Figure 2.5: Geometry used in the computational fluid dynamics simulation to represent the combined mesangium, glomerular basement, and Bowman's space. The top image shows the full scale of the geometry, and the bottom is a close up of the mesangium and glomerular basment membrane in the computational fluid dynamics simulation. The voids represent the foot processes, which block flow.

$$-\nabla P = \frac{\mu_{solvent}}{k_p}\vec{v_s} + b_{exp}\rho|\vec{v_s}|\vec{v_s} \tag{2.12}$$

where P is the pressure, $\mu_{solvent}$ is the viscosity of the fluid, k_p is the permeability, $\vec{v_s}$ is the superficial velocity, and ρ is the density of the fluid. The term b_{exp} is an experimentally determined term used to capture second order dependence on velocity [69]. For our simulations b_{exp} was set to 0, which reduces (2.12) to Darcy's law. Values of k_p are determined with

$$k_p = \frac{3r^2}{20\phi}(-\ln\phi - 0.931) \tag{2.13}$$

where r is the radius of the fibers of the porous media [23].

This simulation deviates from [23] in two key ways. We assume that the foot processes block fluid flow, only allowing fluid to escape out of the glomerular basement

membrane in narrow channels. The second difference is that instead of modeling the interactions between the flow in the glomerular basement membrane and the Bowman's space with a boundary condition, the CFD simulation explicitly models bulk flow in the Bowman's space. One of the limitations of the current fluid flow model is that it does not account for the effects of oncotic pressure, which was captured in the previous work using the boundary condition coupling conditions in the glomerular basement membrane and the Bowman's space.

Boundary 2 is the inlet to the mesangium is a stagnation inlet with a pressure of 35 mmHg [23]. Boundary 3, opposite of boundary 2, is a symmetry plane. Boundary 1 is a no-slip wall representing the cell membrane of a mesangial cell. Boundaries 4 and 5 are the sides of the GBM and are symmetry planes. The undersides of the podocytes are no-slip walls. In the Bowman's space, the podocytes, including the trimmed cavities in the inlet and outlet regions, are no-slip walls. The inlet, on the left side of the Bowman's space is a velocity inlet with a velocity of 392 μm/s. This was estimated by dividing the average single-nephron glomerular filtration rate of a healthy adult [70] by the average cross section area of the Bowman's space [71]. The outlet of the Bowman's space is a pressure outlet with a pressure of 1.3 mmHg [23]. The surface on top of the Bowman's space is a symmetry plane.

For each of the three regions, an automated meshing operation was used to create a 2D quadrilateral mesh. In the Bowman's space, the base size of the mesh is 0.1 μm, which is reduced to 0.01 μm near the tops of the foot processes. The top image in Figure 2.6 shows the mesh generated in the Bowman's space. In between the foot processes, the base size is further reduced to 2.5×10^{-3} μm. Two prism layers are used along the tops of the foot processes, and three layers are used along the sides of the foot processes. The mesh surrounding the foot processes is the bottom image in Figure 2.6.

In the glomerular basement membrane, the base size of the mesh is 5×10^{-3}

μm. Where the fluid exits the glomerular membrane into the Bowman's space, the base size is reduced to 2.5×10^{-3} μm (bottom image of Figure 2.6). Along the underside of the foot processes, three prism layers are used. The mesangium uses a base size of 5×10^{-3} μm as well and three prism layers along the bottom of the mesangium. Figure 2.7 shows a representative portion of the mesh in the mesangium and glomerular basement membrane.

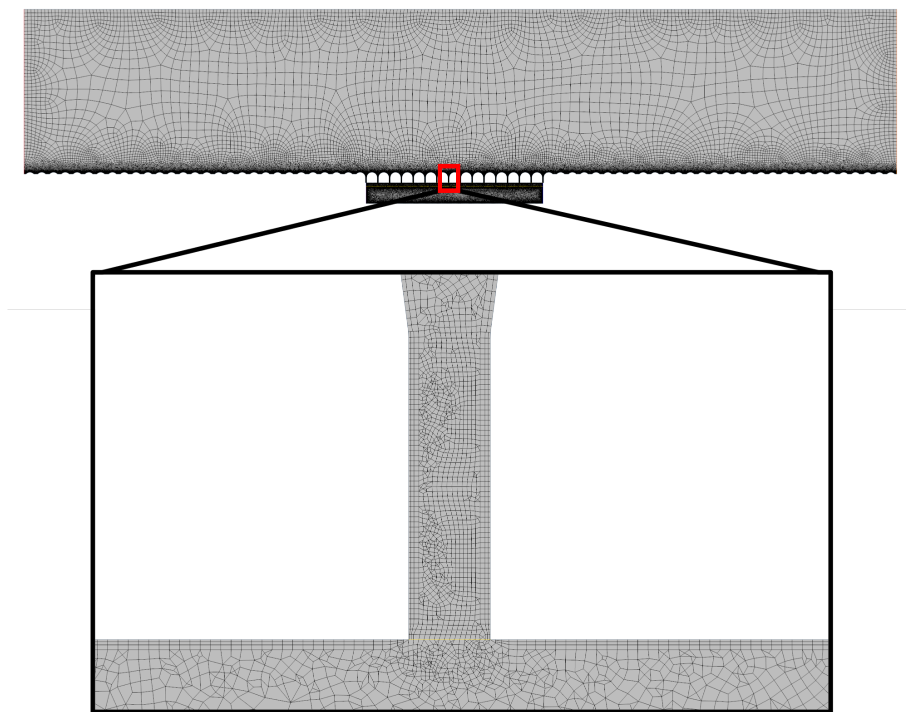


Figure 2.6: The top image is an overview of the mesh generated in the Bowman's space. In the zoomed in region, the base size of the mesh is reduced between the foot processes and in the glomerular basement membrane near the gaps.

A mesh independence study (not shown) was conducted to ensure a sufficiently refined mesh was created. All sizes for each of the three meshes were reduced to 70% of their original values. The average velocity of the fluid flowing through the gaps of the podocytes at 0.1 μm above the glomerular basement membrane was measured. The average velocity through the gaps was 0.974 μm/s on the base mesh and 0.978

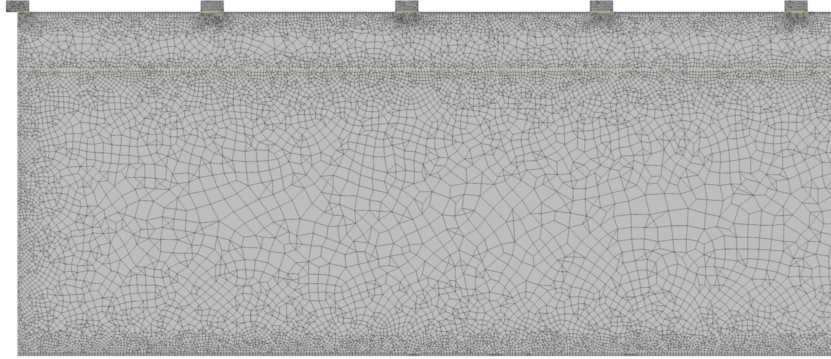


Figure 2.7: Representative regions of the mesh generated for both the glomerular basement membrane and mesangium. Both regions use the same base size for most of their volume.

μm/s on the refined mesh. The percent difference between the two meshes is 0.478%.

The resulting velocity field (Figure 2.8) was sampled and imported into the tissue simulation as the field value for $\vec{v}$ in (2.6).

2.5 Simulation Results

2.5.1 Full Transport Model Results

Figure 2.9 shows the results of the entire transport model simulated within the CompuCell3D simulation. The artifacts in the glucose field are suspected to be related to the meshing of the CFD simulation.

2.5.2 Podocyte Model Validation

A concern when incorporating the biochemical model of the podocytes into (2.6) is that implementation differences in code between sources in (2.6) and their original ODE form could may lead to appreciable differences in behavior. To ensure implementation differences did not affect simulation results of the models for both the podocytes and the mesangium, we compared the results of the Python ODE and PDE implementations of both models. To achieve this, the diffusion and convection terms of (2.6) were removed, glucose was set to a fixed concentration in the simula-

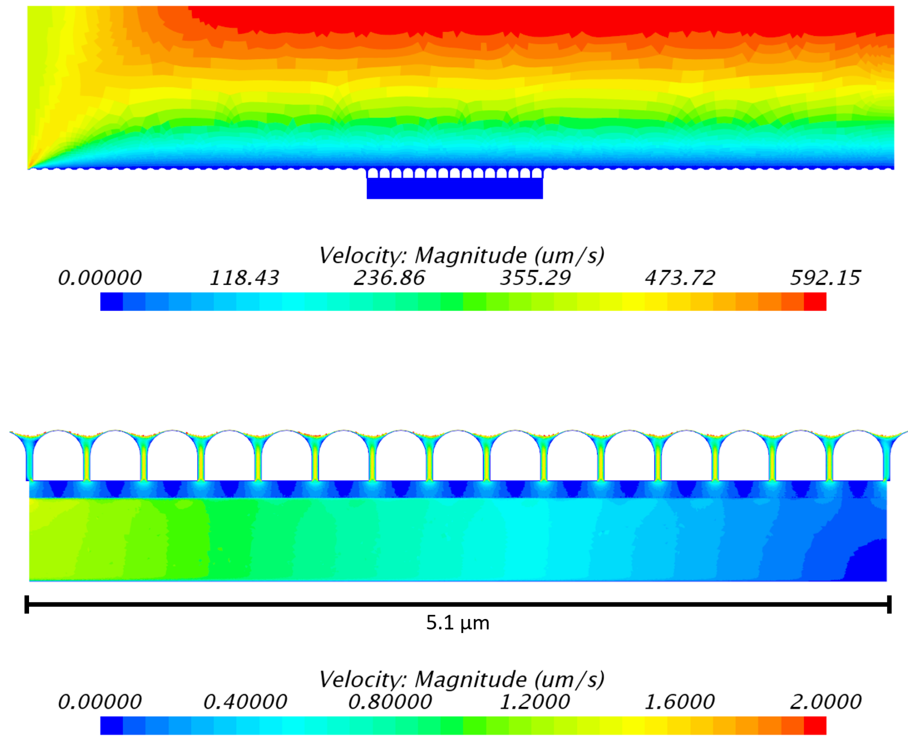


Figure 2.8: Velocity magnitude for the entire flow domain. The bottom image zooms in on the mesangium, glomerular basement membrane, and between the foot processes. The maximum velocity in the bottom image is set to 2 μm/s to highlight the slow moving flow in this region.

tion space, and the simulation was run only allowing the PDEs to progress. We also wanted to determine the effects of diffusive transport on the podocyte model. For this case, diffusion of all species was allowed, including glucose, and the simulation was run only allowing the PDEs to progress. Data was recorded from the center of mass for each cell. Figure 2.10 compares the podocyte model implementations at high and low glucose conditions.

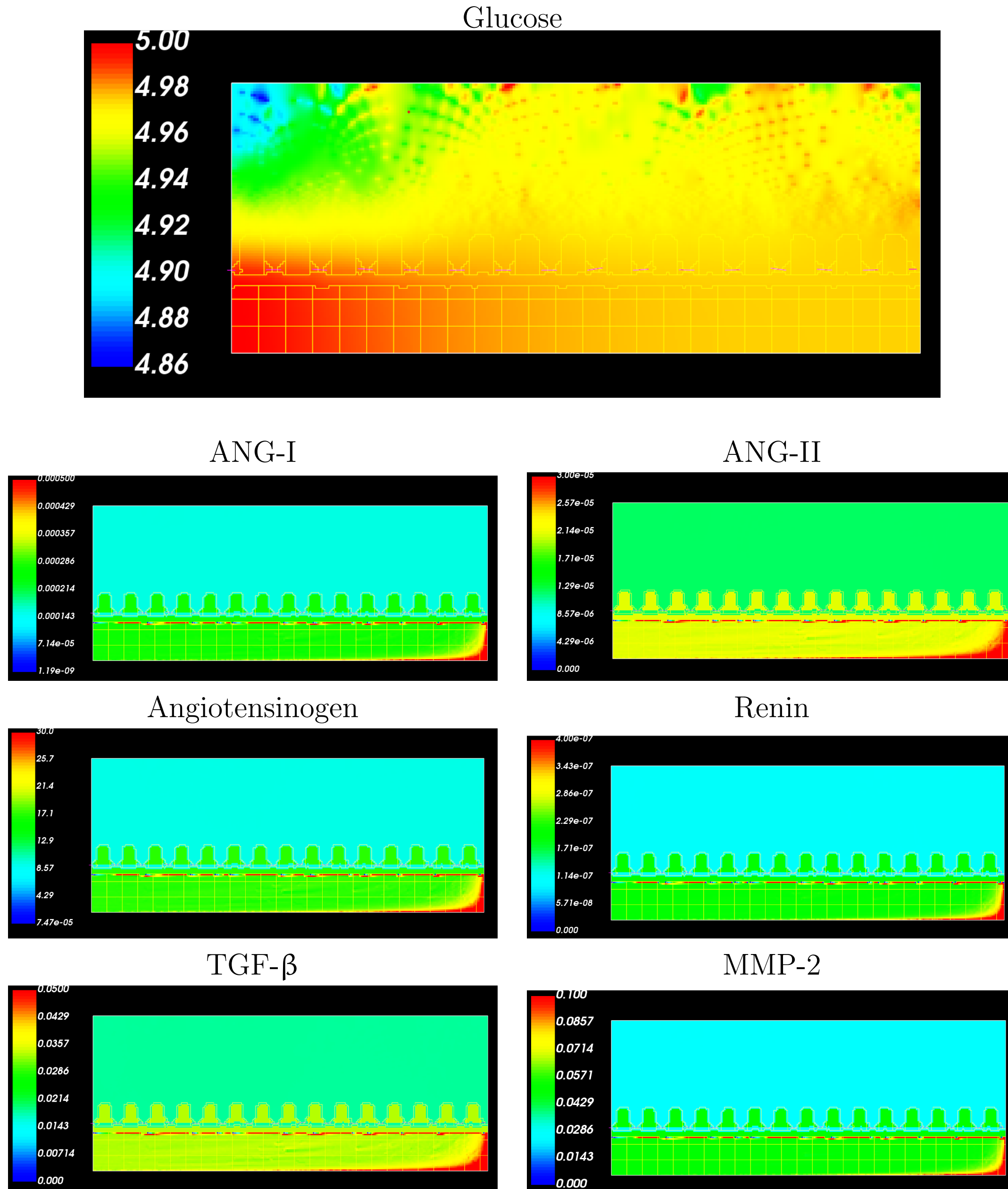


Figure 2.9: Concentration fields (mM) for each the CompuCell3D simulation including all terms of (2.6). The bloodstream glucose concentration is set to 5 mM

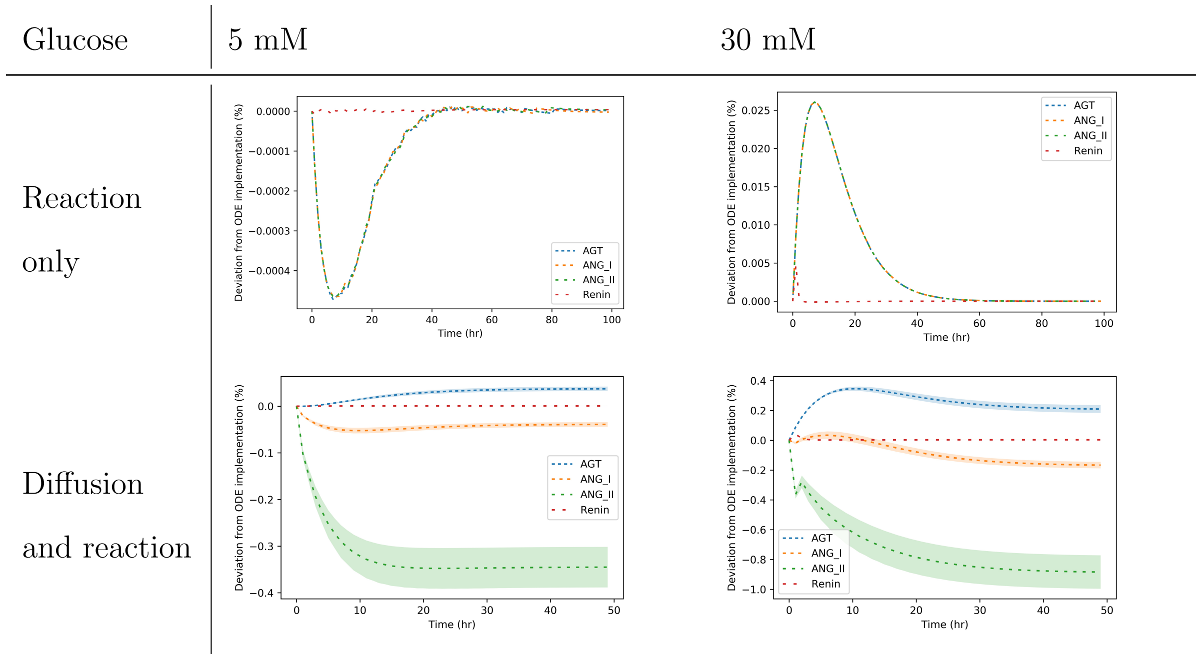


Figure 2.10: Difference of podocyte model behavior between model implementations. The y-axes are the percent differences between the ODE and PDE implementations using the ODE model as the basis. The x-axes are simulation time in hours. While each simulation was run for 500 hours, the graphs generated only show a portion of that simulation's run. Each graph reports the difference as the average for all cells of that type. Each line represents the average concentration of each species across all podocytes in the simulation. In the diffusion and reaction case, the shaded regions represent one standard deviation above and below the average.

Aside from a relatively small transient within the first 50 hours of each, the model implementations appear to be in agreement in the reaction only case. Without convection, the boundary conditions yields a glucose concentration very close to 5 mM through out the simulation space. The small difference in the diffusion and reaction case is not unexpected as a PDE of a well mixed system will have no appreciable difference from an ODE model.

2.5.3 Time Scale Analysis

An analysis of individual model component response to a step change in glucose was conducted. For the diffusion of glucose, a simulation was run with only diffusion enabled. After the first timestep, the blood glucose concentration was changed from 5 mM to 30 mM. Figure 2.11 has the results of this analysis. For the podocyte and mesangial cell models, their ODE implementations were used to see their response to an equal step change in local glucose (Figure 2.12). These processes were chosen to contrast the responses of one of the fastest processes, glucose diffusion, to the slower processes in the model.

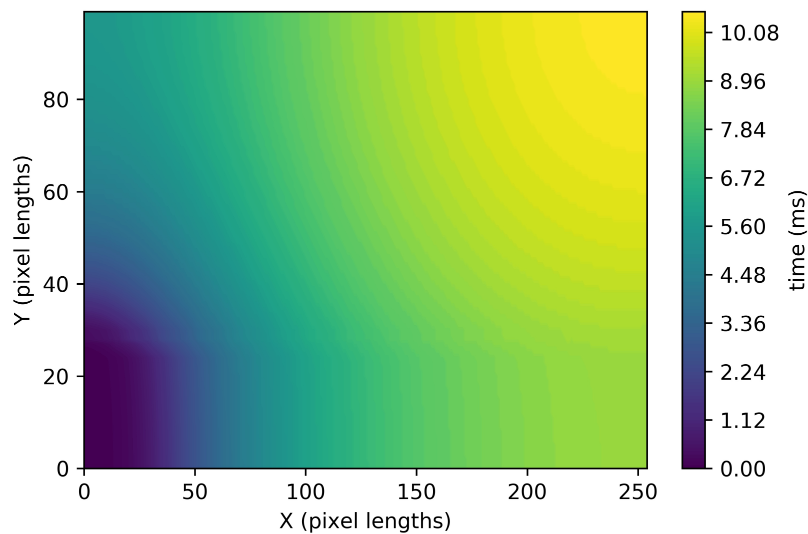


Figure 2.11: The glucose concentration field response to step change in bloodstream glucose. The bloodstream glucose steps from 5 mM to 30 mM. The time reported is how long it takes for the concentration increase at that position to reach 95% of the step size.

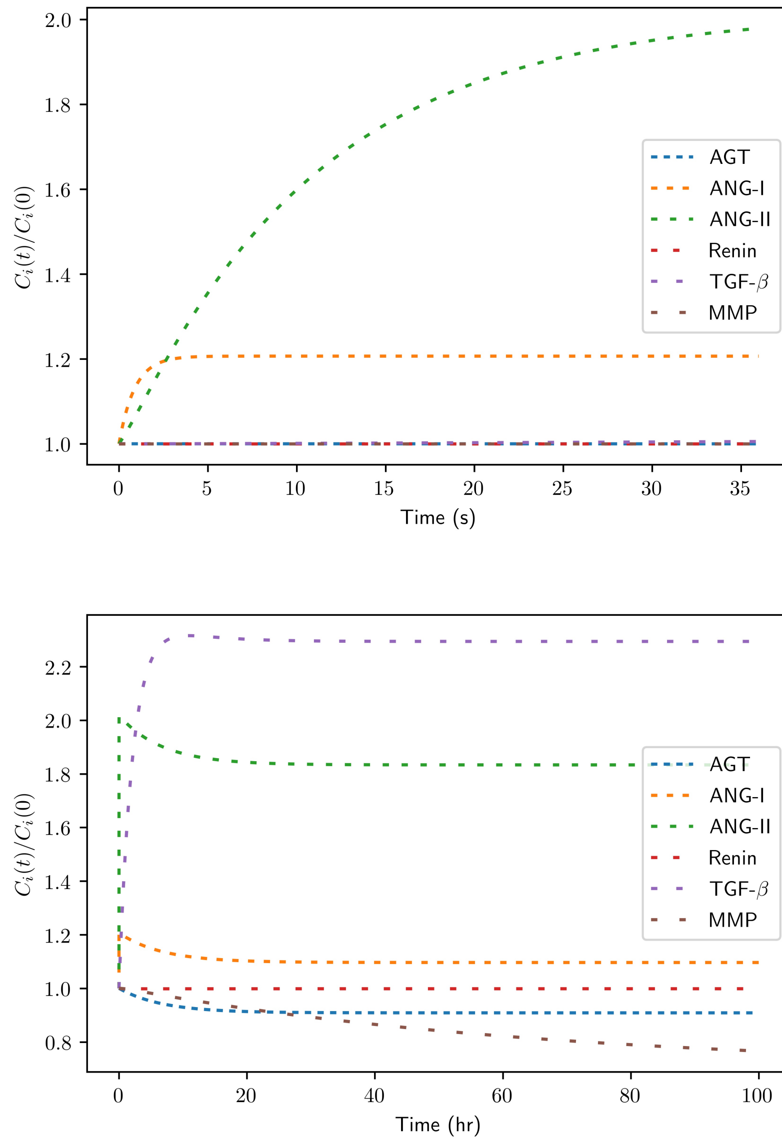


Figure 2.12: The response of the podocyte and mesangial models to a step change in bloodstream glucose. The y-axis is concentration normalized by the initial concentration. The x-axis is time. The top image shows the response over 36 seconds. The bottom image shows the response over 100 hours.

Out of the processes analyzed, the fastest is the diffusion of glucose, occurring on timescales of micro to milliseconds. Reaction of ANG-I and ANG-II occur on time

scales of seconds, and reactions of the rest of the species occur over hours. MMP-2 is on the longest timescale, taking hundreds of hours to approach a steady state.

2.6 Conclusions

In the model's current state, the solute transport model enables considering the effect of space on cellular crosstalk. The tissue geometry has been coupled to the PDEs through the convective and diffusive hinderance terms in (2.6). With the geometry and transport coupled together, I have built a framework for connecting models of kidney diseases progression from a healthy to diseased state.

CHAPTER III

SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter

3.1 Background

While working on Aim I, presented in Chapter II, I wanted to be able to include arbitrary Systems Biology Markup Language (SBML) models as part of the multiscale model. Adding the ability to incorporate SBML models into the virtual kidney will help to expand the applications of the framework because SBML is an intermediary language that facilitates exchanging biochemical reaction network models. A large number of models are already developed into SBML documents. The largest repository of SBML models is the BioModels database [2, 3]. The BioModels database is home to over 8,400 models from literature. These models have been submitted directly by paper authors and database curators [3]. An additional 140,000 models have been submitted to the Biomodels database from the Path2Models project, which automatically generates models from metabolic data [72]. Not all of the models on the BioModels database are curated. For the models that are curated, part of the curation process involves annotating and ensuring correctness of models submitted to the database. The guarantee of correctness means that the curated models may be used to create a test suite for software developed to use SBML models [2]. The large number of available models and ease of use makes incorporating existing SBML models attractive.

SBML helps with model exchange by addressing three main problems. First, since the language is not specific to a single software or simulator, its flexibility enables

researchers to use diverse tools with varying strengths when developing or analyzing a model. Second, by decoupling the model from the software used to build it, a model no longer becomes obsolete if the software does. Third, standardizing how models are defined reduces the confusion that may arise from a researcher working with another researcher's model [1].

While SBML is specifically designed for a wide variety of computational biology models [1], The flexibility of SBML stems from the fact that "it can describe any model of form:

$$\frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{p}) \tag{3.1}$$

where $\mathbf{x}$ is the state vector of the model, and $\mathbf{p}$ is a vector of time-independent parameters" [73]. SBML is so flexible that even models of zombie infections developed with variants of the susceptible infected removed epidemiology model have been described using the standard [74].

The interoperability and usability of SBML is achieved by it being a declarative language as opposed to an imperative language. This means that SBML merely lays out the model components without explicitly implementing how to solve the model. SBML leaves implementing and solving the model to the simulator or tool of the researcher's choice. Furthermore, a researcher need not focus on the details of implementation when they have software previously developed to read, construct, and solve SBML models [73].

SBML at its core is built upon Extensible Markup Language (XML) [1]. XML was developed as a standardized way to store arbitrary data in a document that is human and machine readable [75]. SBML builds upon the XML standard by implementing standards for following model components [76, 77]:

- The **SBML Container** points to the appropriate XML namespace for the SBML model based on the version and level of SBML used in the document. In level 3 SBML, additional packages used to expand upon the base functionality

are declared here.

- The **Model** object is located within the **SBML Container** and serves to contain all other model elements. The base units used in the model may be defined in the **Model** element.

- A **Function Definition** contains a single user-defined function. This function is limited to only use other functions defined within the model and a subset of MathML elements, such as arithmetic operators and logical operators.

- A **Unit Definition** allows a modeler to define units needed for their model from a base set of units specified in the SBML specifications. Note that **Unit Definitions** are completely optional, and SBML does not specify or require a check for unit consistency in a model. It is up to the modeler or the software to ensure units are explicitly accounted for and consistent in their model.

- A **Compartment** represents an arbitrary space in which a **Species** exists. **Compartments** need not be three dimensional; a surface may be represented by a two-dimensional **Compartment**. SBML level 3 even supports **Compartments** with fractional dimensions. Just like with **Unit Definitions**, SBML does not check for unit consistency with respect to **Compartment** dimensionality.

- A **Species** represents "a pool of entities that (a) are considered indistinguishable from each other for the purposes of the model, (b) may participate in **Reactions**, and (c) are located in a specific **Compartment**" [76]. This is a more abstract implementation of a chemical species, and can represent anything that fits the above requirements. For example, a population of rabbits can be a **Species** in a population model represented in SBML.

- A **Parameter** is simply intended to be a value with an identifier. **Parameters**

can be set to be constant or used as variables in your model. Additionally, **Parameters** can have units associated with them, without any unit consistency checks.

- An **Initial Assignment** is an optional way to set any value of a model at $t \leq 0$. Each value may have an initial value specified, but an **Initial Assignment** allows for a value to be set based on a calculation instead of a specified value.
- **Rules** are used to provide additional ways to compute variable values in a model.
  - An **Algebraic Rule** represents an equation of the form

    $$0 = a + b + c + \ldots + z \tag{3.2}$$

    where a through z represent an arbitrary number of model values. Each value is calculated as appropriate to satisfy the **Rule**. SBML provides a set of rules that specifies which variables may not be determined by an **Algebraic Rule**.
  - An **Assignment Rule** represents an equation of the form

    $$a = b + c + \ldots + z. \tag{3.3}$$

    While the mathematical forms of Algebraic and **Assignment Rules** are equivalent in the abstract, they are handled differently. In an **Assignment Rule**, it is assumed that the right hand side of the equation may be used to directly calculate the value of a.
  - A **Rate Rule** represents an equation of the form

    $$\frac{d(a)}{dt} = b + c + \ldots + z. \tag{3.4}$$

The **Rate Rule** is similar to an **Assignment Rule**. The **Rate Rule** is integrated to directly assign the value of a.

- A **Constraint** defines the assumptions that a model is valid under. A **Constraint** comprises of a math term that determines if the **Constraint** is satisfied or not and a message that is displayed to the user if the **Constraint** is no longer satisfied.

- A **Reaction** is "any kind of process that can change the quantity of one or more **Species** in a model" [76]. Similar to a **Species** component, a **Reaction** is an abstract implementation of a chemical reaction. As such, a **Reaction** can represent any process that is appropriate to the model, including transport from one **Compartment** to another.

- An **Event** is an instantaneous effect on the model. **Events** are comprised of a Trigger, an optional Delay, an optional Priority, and a list of assignments that perturbs the model state.

All of these components are supported by SBML level 3, which has the greatest number of features [76]. Each iteration of the SBML standard is denoted by a version and level. Each level represents a increase in features compared to the previous level, while each version is a refinement of the previous version of that level. The three different levels of SBML are continuously developed and are intended to serve a continual purpose [77, 78].

The simulator of use in Aim I is Compucell3D. Compucell3D supports implementing custom Python [4] scripts that can determine cell behaviors [48]. Incorporated into Compucell3D is libRoadRunner, an SBML model simulator. A key strength of libRoadRunner is that it compiles an SBML model directly into machine code [73]. As a result, the implemented model is simulated very efficiently, but this implementation is mostly immutable. Changing the functionality of a model during runtime

has been useful in my research for model analysis [6]. As a result, the immutability of an SBML model implemented by libRoadRunner is a negative that outweighs the benefits of faster execution.

Another tool useful to researchers who want to implement SBML models is the Systems Biology Format Converter (SBFC). SBFC is directly integrated into the Biomodels database and is used to automatically convert SBML models into the full range of formats it supports [5]. The Octave implementations generated by SBFC are scripts that simply initialize and simulate the model for a certain duration. These scripts do not take advantage of programming concepts like object oriented programming, which would reduce developer time needed to incorporate these models. At the time of writing, SBFC does not support Python.

Other tools exist for working with SBML models, such as COPASI or libSBMLsim, but most tools developed for working with SBML were not designed to be directly integrated into other programs [73].

3.2 Software Details

The work presented in this section includes sections of the paper, code, and documentation that have been peer reviewed and published in [79]. My contributions to this work include designing and writing the code and writing the application programming interface documentation.

3.2.1 Summary

The goal of this aim is to develop a Python package called SBMLtoODEpy to address the limitations detailed in earlier sections by enabling conversion of SBML models into Python classes that can be rapidly incorporated into biomedical systems modeling projects written in Python, such as the project described in Aim I, or used directly in Python. SBMLtoODEpy generates code that uses object oriented programming

to create code that users can write their own code to interface with. The program aims to accelerate construction of multiscale models that import and reuse published SBML models.

In SBMLtoODEpy, each of the model components are extracted using libSBML. LibSBML is a key tool that helps support SBML. It is a software library that has an application programming interface that simplifies reading and editing SBML documents [78]. The model components can be output to a JavaScript Object Notation (JSON) file. JSON is a format that is easier for users to read and directly edit than either the SBML or Python implementations of the model. The extracted model components are used to create a Python file that defines a class that implements the model. A method for the class is generated to solve the model using a wrapper for the LSODA algorithm in the SciPy Python package [80], and the NumPy Python package [81] is also used. To verify that SBMLtoODEpy properly interprets SBML files and converts them into functional differential equations models, we compared the results of SBMLtoODEpy with COPASI, a graphical user interface based platform for simulating SBML models [82], for a set of representative SBML files downloaded from the BioModels Database that were deposited for a selection of systems biology publications [83, 84, 85, 86, 87, 88]. This comparison served purely to verify that SBMLtoODEpy was properly setting up the equations for the model as both COPASI and SBMLtoODEpy rely on ODEPACK to solve differential equations [89]. These files have been included in the SBMLtoODEpy package within the `sbmltoodepy/sbml_files subdirectory` to serve as examples for users.

The code is split into three main roles. The first is the algorithm that uses libSBML to extract the model components from the SBML document. At this point, the extracted data can either be saved to a JSON file or passed to the second algorithm of the package. The second algorithm takes the extracted model components and writes the Python class to a file. This includes ordering **Assignment Rules** such that none

of the **Rules** use undefined values and constructing the system of differential equations that arise from the **Reactions** and **Rate Rules** of the model. SBMLtoODEpy represents the ODEs as

$$\frac{d}{dt}\mathbf{s} = \mathbf{A}\boldsymbol{\xi}(\mathbf{s}) + \mathbf{r}(\mathbf{s}) \tag{3.5}$$

where $\mathbf{s}$ is a vector representing the model state, $\mathbf{A}$ is a matrix of stoichiometric coefficients, $\boldsymbol{\xi}$ is a vector containing the extents of reaction, and $\mathbf{r}$ is a vector containing the rates of change determined by **Rate Rules**. Figure 3.1 illustrates the roles of these algorithms.

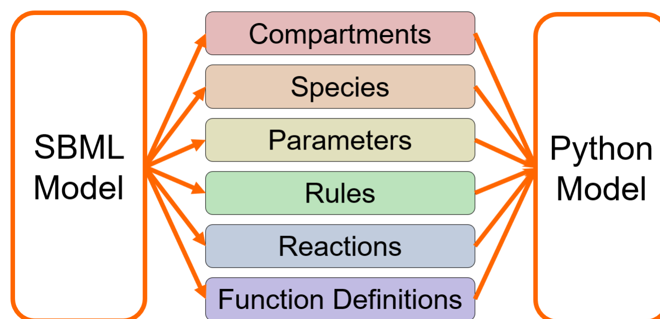


Figure 3.1: The two algorithms implemented in the SBMLtoODEpy package are represented as the two sets of arrows. The first algorithm parses the components of the SBML model (filled boxes) and stores the results. The second algorithm defines Python classes and defines methods to solve the model numerically.

.

The third portion of the code includes the classes and functions that help implement specific features of SBML components, such as **Compartments** and **Parameters**. This includes methods that prevent modifying values that are flagged as constant and that compute the concentration of **Species** based on **Compartment** volume.

3.3 Tutorial

3.3.1 Interactive Jupyter Notebook

An interactive Jupyter notebook working through the installation and a couple of examples can be found at `https://github.com/SMRuggiero/sbmltoodepy/blob/master/SBMLtoODEpy.ipynb?short_path=948bbb4`

An online version that does not require installation of Python can be found at `https://colab.research.google.com/drive/1yBscTNFwgliOUD1OYhdWh-3HvXdj1YUF`

3.3.2 Creating a Python Model

The fastest way to create a Python model using SBMLtoODEpy is to open a Python interpreter in the same folder as the SBML file. Then, run the following lines of Python code using the name of your model.

```
import sbmltoodepy
sbmltoodepy.ParseAndCreateModel("YourModelNameHere.xml")
```

If no errors occur, then a file with the same name as the SBML file but with the extension .py will be created in the same folder. To have a file created with a different name, you can use the outputFilePath keyword argument.

```
sbmltoodepy.ParseAndCreateModel("YourModelNameHere.xml",
                                 outputFilePath = "PythonFile.py")
```

The new Python module defines a class named SBMLmodel. To customize the name of the class, use the className keyword argument.

```
sbmltoodepy.ParseAndCreateModel("YourModelNameHere.xml",
                                 outputFilePath = "PythonFile.py",
                                 className = "ModelName")
```

The last optional keyword argument of ParseAndCreateModel() is jsonFilePath. If specified, a JSON file containing all of the model elements parsed from the SBML

model will be created.

```
sbmltoodepy.ParseAndCreateModel("YourModelNameHere.xml",
                                 jsonFilePath = "YourModelNameHere.
                                 json", outputFilePath = "
                                 PythonFile.py", className = "
                                 ModelName")
```

In reality, creating a Python model with SBMLtoODEpy is a two step process that can be broken up.

```
dataOfModel = sbmltoodepy.parse.ParseSBMLFile("YourModelNameHere.xml
                                 ")
sbmltoodepy.modulegeneration.GenerateModel(dataOfModel, "PythonFile.
                                 py", objectName = "ModelName")
```

The function ParseAndCreateModel serves as a wrapper for two other functions, ParseSBMLFile and GenerateModel. ParseSBMLFile returns an instance of the ModelData class with all of the components pulled from the model. While you can use that instance as input to GenerateModel, there is another feature of SBMLtoODEpy that the ModelData class provides. The ModelData class has a method, DumpToJSON, that generates a JSON file with the model components.

```
sbmltoodepy.dataclasses.ModelData.DumpToJSON(dataOfModel,"
                                 YourModelNameHere.json")
```

A JSON file created from the DumpToJSON method can be used a create a new instance of the ModelData class.

```
newInstance = sbmltoodepy.dataclasses.ModelData.LoadFromJSON("
                                 YourModelNameHere.json")
```

A possible use case for this would be to generate a JSON file from an SBML model, change the JSON file, which is more human readable than an SBML file, and use the changed JSON file to create the Python model.

3.3.3 Exploring a Newly Created Python Model

After generating the Python implementation of a model, you can now import the new class and instantiate it.

```
from PythonFile import ModelName
modelInstance = ModelName()
```

The main components that make up the state of an SBML are **Compartments**, **Parameters**, and **Species**. Each of these are stored in a dictionary that is a member of the model class. These members are named c, p, and s respectively. The key for each dictionary entry is the "id", as defined by the SBML specification, for the component. By printing one of these dictionaries keys, you can see the id of each **Compartment**, **Parameter**, or **Species**.

```
# get the dictionary keys for the IDs of the species in the model
print(modelInstance.s.keys())
# get the dictionary keys for the IDs of the compartments in the
                                            model
print(modelInstance.c.keys())
# get the dictionary keys for the IDs of the parameters in the model
print(modelInstance.p.keys())
```

The individual model elements are represented as instances of an appropriate class. **Parameters** are instances of the `Parameter class`, and the value member contains the **Parameter's** value. **Species** are instances of the `Species` class, and the concentration and amount members are used in calculations. **Compartments** are instances of the `Compartment` class, and the size member contains the size of the **Compartment**.

```
# replace compartmentId with one of the dictionary keys returned
                                            from print(modelInstance.c.keys())
print(modelInstance.c['compartmentId'].size)
# replace parameterId with one of the dictionary keys returned from
```

```
                                   print(modelInstance.p.keys())
print(modelInstance.p['parameterId'].value)
# replace speciesId with one of the dictionary keys returned from
                                   print(modelInstance.s.keys())
print(modelInstance.s['speciesId'].concentration)
print(modelInstance.s['speciesId'].amount)
```

Function Definitions and **Reactions** are stored in a similar manner. Currently, **Rules** and **Initial Assignments** are not handled the same way. **Rate Rules** are bound methods of the model class that are called alongside **Reactions**, and **Assignment Rules** and **Initial Assignments** are implemented through a method of the model class, `AssignmentRules()`. **Algebraic Rules** are not completely supported currently.

In SBML, each model component can have both an id and a name, but only the id is required. Each of the dictionaries containing **Compartments**, **Parameters**, **Species**, **Reactions**, and **Function Definitions** can be searched by the component name using the appropriate method. Note that the original model must not have a blank metadata field for the corresponding entry for the search by name to yield any results. In SBML, there is no guarantee that names are unique (each component in a model does have a unique id). If a single component matches, a list with the component's id and object is returned. If multiple components match, then a nested list is returned. Each element in the list is a list with a component's id and object. Lastly, if no match is found, the behavior of the function depends on the keyward argument suppress. If suppress is False, an exception is raised. If suppress is True, then an empty list is returned.

```
modelInstance.SearchParametersByName('parameter name', suppress =
                                   False)
modelInstance.SearchCompartmentsByName('compartment name', suppress
                                   = False)
```

```
modelInstance.SearchSpeciesByName('species name', suppress = False)
modelInstance.SearchReactionsByName('reaction name', suppress =
                                   False)
modelInstance.SearchFunctionsByName('function name', suppress =
                                   False)
```

The last part of the model's state is simulation time; the current simulation time is stored in the time member of the class. The unit of time in the model is treated as arbitrary by the package. The unit for time is dependent on the units of parameters set by the modeler. This package does not check if the units in a model are consistent.

```
print(modelInstance.time)
```

3.3.4 Simulating the Model

The RunSimulation() method is used to step the ODE model forward in time by timeinterval. The method updates the value of each model component and the total elapsed simulation time over all calls to RunSimulation.

```
modelInstance.RunSimulation(timeinterval)
```

There are two optional keyword arguments, absoluteTolerance and relativeTolerance, which are for tuning the tolerances used to solve the ODE model with LSODA.

```
modelInstance.RunSimulation(timeinterval, absoluteTolerance = 1e-12,
                             relativeTolerance = 1e-6)
```

An important thing to note is that the model only keeps values for the current time. If you wish to graph the evolution of a **Species** with respect to time, the concentration and time will need to be sampled at the appropriate time points.

```
import numpy as np
times = np.zeros(101)
times[0] = modelInstance.time
concentrations = np.zeros(101)
```

```
concentrations[0] = modelInstance.s['speciesId'].concentration
timeinterval = 1
for i in range(100):
  modelInstance.RunSimulation(timeinterval)
  times[i+1] = modelInstance.time
  concentrations[i+1] = modelInstance.s['speciesId'].concentration
```

The results can then be graphed using any method you would like. For graphing directly in python, matplotlib is a good option.

```
import matplotlib.pyplot as plt
plt.plot(times,concentrations)
```

3.4 Conclusion

SBMLtoODEpy was designed to be extremely researcher friendly; researchers are able to convert models with a single function call. Even within a single Python script, a user can convert a model and import it immediately during runtime. The code and documentations are the peer reviewed products of this work. The project is open source and may be freely edited and redistributed. The source code can be pulled from `https://github.com/SMRuggiero/sbmltoodepy/tree/master/sbmltoodepy/sbml_files` [90] in addition to PiPy. SBMLtoODEpy has already been incorporated into my work. In the virtual kidney project described in Chapter II, I instantiated 255 copies of the same model to compute the mesangial cell membrane boundary condition.

CHAPTER IV

Procedural Generation Methods in Tissue Simulation

4.1 Background

4.1.1 Procedural Content Generation

Procedural content generation (PCG) broadly refers to the programmatic creation of game elements. These elements include, but are not limited to, maps, items, characters, graphical elements, and music within the game [91]. One of the earliest games that made heavy use of PCG is the 1984 cult classic Rogue. Rogue places you in the shoes of a novice adventurer delving into a dungeon. If your character dies, the game ends, and you must restart from the beginning. The catch: since the procedural generation used by Rogue to create each floor of the dungeon is random, the layouts of the floors are new each time you play [92]. Figure 4.1 shows the first level from Rogue from two separate attempts.

In their 2013 submission to The Fourth Procedural Content Generation in Games workshop, Compton, Osborn, and Mateas proposed coining the titular term "generative methods" as an umbrella that includes PCG along with other analogous methods from other fields [93]. The working definition proposed for generative methods is "a function which produces artifacts" where an artifact has a structure and a context. If we take a generative method that produces figures in the form of a picture as an example, the figure is the artifact, how the pixels are encoded is the structure, and the article that the figure is in is the context. The primary argument for using the term generative method is to increase the academic visibility of PCG research and

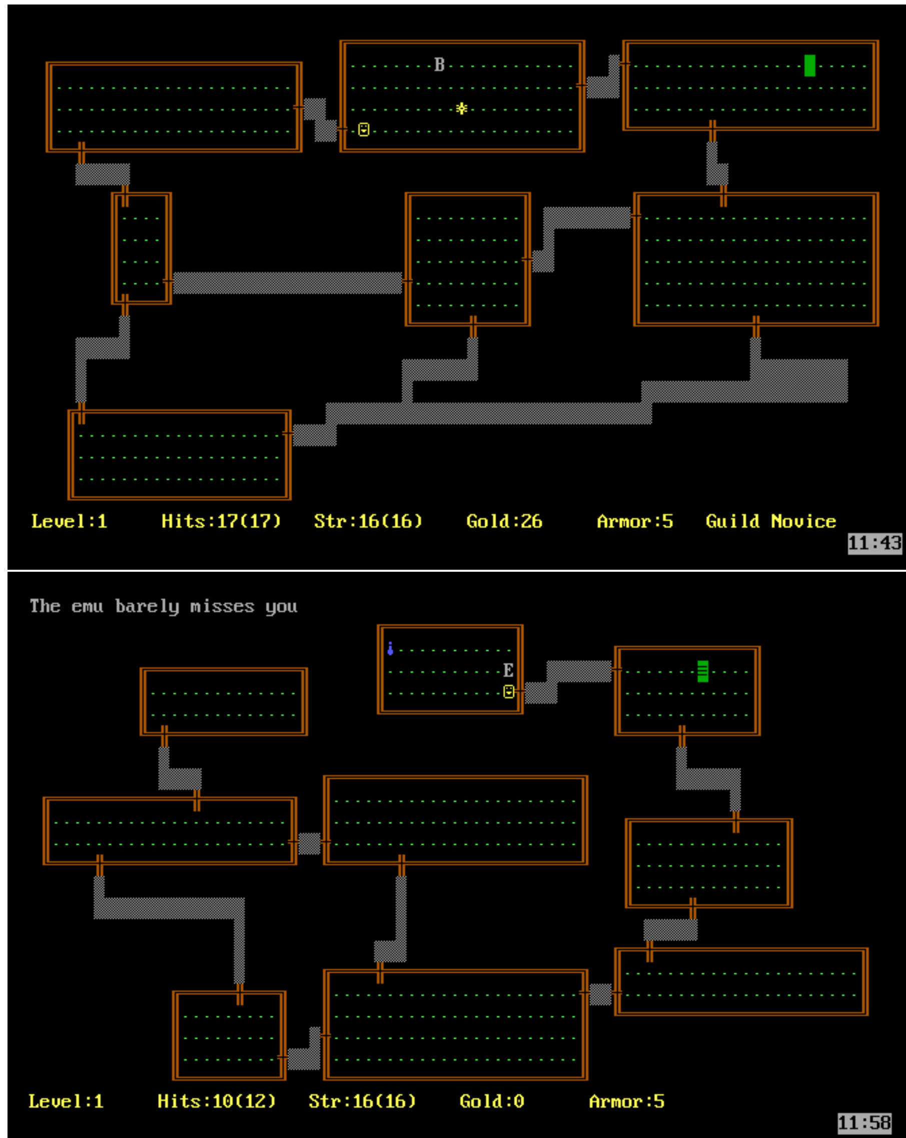


Figure 4.1: Two separate versions of the first level of the game Rogue [92]. Each of the game's elements are represented by a character of text. The brown borders and green periods represent the walls and interior space of rooms, respectively. The gray shaded areas are hallways. The player is the yellow face character, and enemies are represented by gray capital letters: B for a bat in the top image and E for an emu in the bottom image. The stairs going deeper into the dungeon are represented by a flashing ≡ with a green background.

to continue the cross pollination of ideas between fields [93]. PCG borrows solutions from the fields of "AI, computational intelligence, computer graphics, modeling, and discrete mathematics" [94]. Compton, Osborn, and Mateas argue that advances in PCG have little academic visibility to other fields despite the presence of similar techniques in art, architecture, computer aided design, optimization, and more [93].

Within engineering, there are techniques that fit the definition of a generative method. A broad range of engineering challenges from logistics to reactor design can be viewed through the lens of designing a set of problem relevant parameters to find the best design. Using this lens, the task can distilled down into an optimization problem [95]. When solving an optimization problem, an algorithm searches a design space to produce a parameter set that yields an optimal solution [96]. The output of this algorithm has a structure, typically a list of parameter values, and a context, the model used in the optimization problem. In other words, optimization techniques fit the definition of a generative method.

Genetic algorithms are a class of global optimization techniques that attempt to solve an optimization problem by conceptualizing a models parameter as genes in a genome. A genetic algorithm puts a population of trial parameter sets through a process of mimicing mutation, recombination, and natural selection. Populations start across an wide area of the design space and narrow towards the best optima encountered [97, 96, 98, 91, 99]. Two examples of problems that can be solved by genetic algorithms are scheduling and transportation problems [99]. In the context of biomedical engineering, genetic algorithms have shown success in fitting quasi-linear viscoelatic models [100] to data generated from a wide variety of soft tissues [101, 102].

Not only are genetic and evolutionary algorithms good choices for engineering problems with a high number of local minima, they are the methods of choice for search-based PCG [91]. One use case of search-based PCG is the generation of tracks of a racing game tuned to a specific player [103]. In this case, core information for

producing a track is represented as a series of waypoints that form a closed loop. The generator produces the tracks from these waypoints by drawing curves on either side of the loop. Figure 4.2 shows two generated tracks and their waypoints.

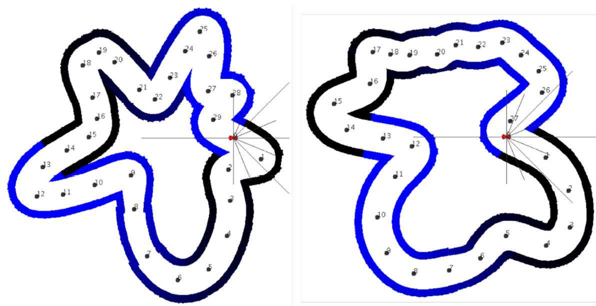


Figure 4.2: Two example tracks generated by the method presented in [103]. These examples were generated based on the play style of one of the authors and serve as an example of the same method producing individualized content for different players.

A model of the player is then used to simulate how the player might perform on the track. The track was evaluated on the model player's progress on the track, variance across trials, and maximum speed achieved on the track. Starting from a random set of points, the optimization algorithm used was able to design tracks specifically for a single player [103].

4.1.2 Motivation

In an early stage of the DKD research presented in Chapter II, the focus was on examining the effects of cellular dysfunction at the level of an entire glomerulus. To this end we prepared simulations of the cross section of the glomerulus in CompuCell3D [48]. For background information on DKD and CompuCell3D please refer to Chapter II. Figure 4.3 served as a reference for developing the starting point for our early simulations as seen in Figure 4.4.

However, looking at the structure of an actual glomerulus (Figure 4.5), questions about how best to capture the cross section arise. The actual structure of a glomerulus is highly irregular; cross sections of different glomeruli will have varying numbers of

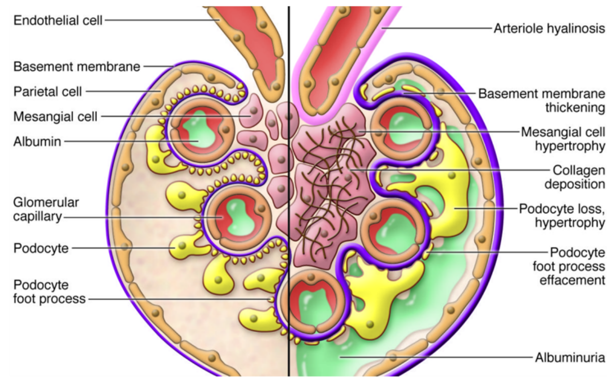


Figure 4.3: An illustrated cross section of a glomerulus adapted from [16]. This illustration served as an initial reference for the first set of initial conditions created.

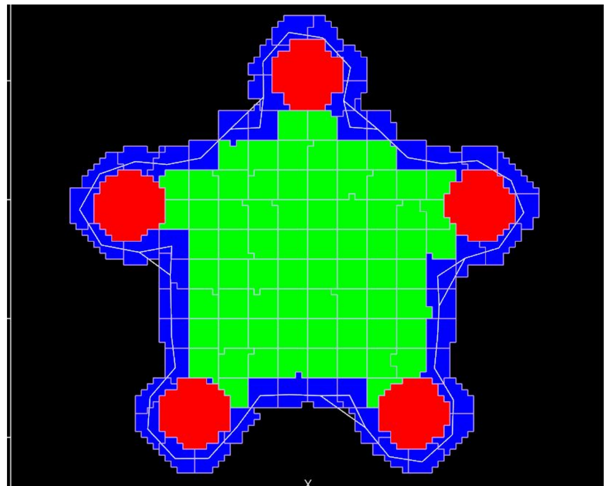


Figure 4.4: The earliest simulation initial conditions that had the capillaries (red) placed and podocytes (blue) properly enveloping the mesangium (green) and capillaries.

capillaries, variance in the positioning of capillaries, differences in the volume of the mesangium relative to the surface, etc. Since patients have thousands of glomeruli and we expect that the structural failure of the glomerulus is directly affected by the structure itself, we need to be able to simulate starting from a variety of initial structures.

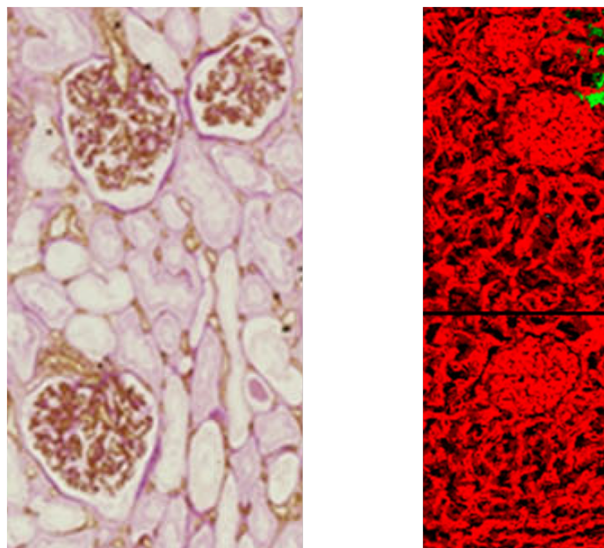


Figure 4.5: Microscopy and 3D reconstructions of renal tissue sampled from patients with mid stage kidney disease without other complications [104]. The spherical structures in both images are glomeruli. The cross sections and surfaces of glomeruli are highly variable but do have patterns.

4.1.3 Methods Used in Literature

CompuCell3D provides some rudimentary functions to researchers [55], but the common ways of initializing CompuCell3D simulations are to use CellDraw, a tool packaged with CompuCell3D, or to sweep through the pixels of the simulation space and to draw cell objects based on coordinates.

CellDraw provides a graphical user interface for drawing the initial conditions of a simulation. It takes the completed drawing and converts it into a plain text format that contains position data for each cell [54]. The main advantage of CellDraw is that it allows a researcher to develop a complicated tissue structure as a starting point

for their simulation with minimal programming knowledge necessary. Its biggest drawback is the large amount of researcher time needed to develop a single starting geometry. Due to the programming knowledge that was expected to be necessary in developing the glomerular simulations, the large number of geometries needed, and the burden on the user's drawing ability, CellDraw was determined to not be a great fit for the project.

Sweeping through all of the pixels in the simulation space has its own set of strengths and weaknesses. Typically the code sweeps along each axis of the simulation space and places cells at each pixel by running a set of calculations. The primary strength is the high level of precision the developer has over the resulting geometry. This method can also be parameterized, allowing the resulting geometry to be tweaked by changing variables and leaving the rest of the code unchanged. But parameterization of this method has severe diminishing returns. As a developer seeks to introduce more complex ways to parameterize their simulation geometry, the resulting code complexity outpaces the complexity introduced into the starting geometry.

As an example, consider placing a podocyte cell one pixel below a capillary, or one pixel lower along the y-axis in the convention of CompuCell3D. Assume that the placement of the capillary has been parameterized as well. If we choose parameters to place a capillary at $(2, 2)$, we will place a podocyte at $(2, 1)$. If we start our sweeps from $(0, 0)$, $(2, 1)$ will be visited first, regardless of which axis is looped along first, and the capillary will not been placed at $(2, 2)$ yet. Of course, the simplest approach is to reverse the direction the y-axis is iterated along. Swapping the y-axis works here, but this is a simplified version of a problem encountered in developing the initial conditions for the glomerular simulations. The actual problem is placing podocytes along the surface of the capillary not in contact with the mesangium, which requires placement above and below the capillary.

A seemingly straight forward approach to consider is checking if a capillary will be placed at $(2, 2)$. Since the placement of the capillary itself has been parameterized, the check at $(2, 2)$ can be as computationally complex as the check in the current pixel. This alone does not make writing of the code considerably more complex. Assuming that checking the pixel is its own function, the researcher could just call the function at (x, y) and $(x, y+1)$. However, parameterizing both the podocyte and capillary can create interactions. Since the capillary is parameterized, its placement might depend on checking the pixel above or to the right of it. Since we started from placing a podocyte relative to an object that has a location that is parameterized, all of these extra checks will need to be done at every pixel visit that could possibly allow a podocyte and not just a specific location. An approach that avoids chaining checks would be to simply sweep through the simulation space twice and place the podocytes on the second pass. The issue is that the same interaction that causes chaining multiple checks leads to needing multiple sets of loops through the entire simulation space.

Looping through the simulation space is attractive because CompuCell3D stores the cell locations in a three-dimensional array, and a nest of three loops is a logical way to access every element. However, we do not actually need or want to visit every element; not every pixel is occupied by a cell. If we resolve where all the cells are before actually placing them, we can skip over the empty elements. Therefore, the starting point for developing our simulation geometry combined conceptual aspects of CellDraw and the method of programmatically evaluating what cell goes in a pixel based on its coordinates.

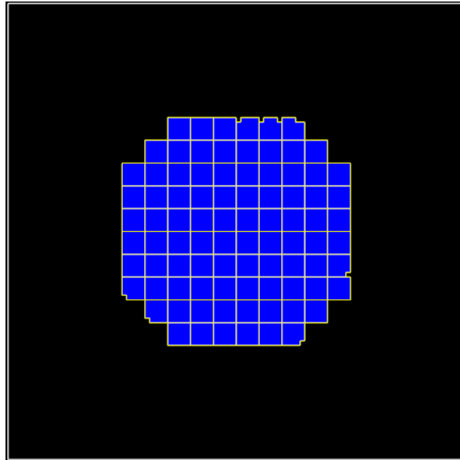
One method from literature on a different scale of systems biology simulation involved generating the structure of a cell or tissue using a generative model [105, 106, 107]. In the context of machine learning, a generative model is trained to estimate the probability distribution of a population [108]. The generative models used in

these methods predict the distribution by producing simulacra of individuals of the population. In these cases it is the simulacra themselves that are useful [106]. These models are trained using massive amounts of data of the population they are trying to generate, typically in the form of annotated microscopy [105, 106, 107]. While data sets containing microscopy of glomeruli are available [109], and machine learning has been applied to studying glomeruli [110], a data set with the level of annotation needed for our purposes is not available. Further, we do not seek to recreate images of glomeruli, we are trying to create a representation that may be used in CompuCell3D.

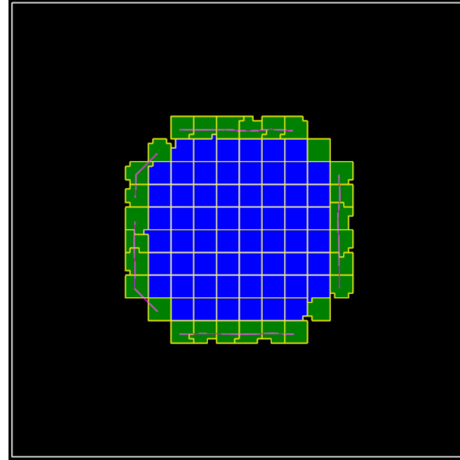
4.2 Generation of Cellular Geometry

4.2.1 Glomerulus Cross Sections

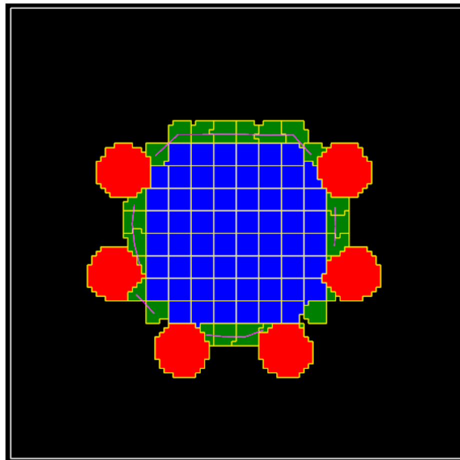
We wanted the ability to statistically sample a population of geometries, produce the geometry rapidly for each simulation, thoroughly parameterize how the cross sections are laid out, and do all of this without the data useful to train a generative model. Inspired by the level generation of games like Rogue [92], an algorithm was developed to generate the cross section of the glomerulus as if drawing part of a level. The algorithm places the cells in our simulation by computing the location of each cell and the exact pixels the cell occupies, and then "paints" them to the array of cells by writing to all elements the cell occupies. Painting the glomerulus was done in four steps. Figure 4.6 shows the geometry after each of the steps.



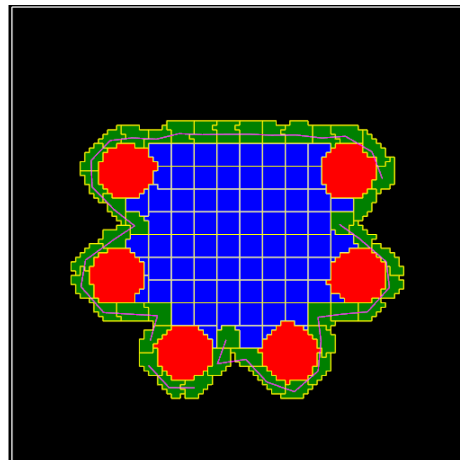
Step One: Mesangium



Step Two: Podocytes



Step Three: Capillaries



Step Four: Capillary Podocytes

Figure 4.6: The simulation initial geometry after each step of the algorithm.

The first step starts by drawing in the mesangium. The method `CreateBlob()` uses the ratio of the mesangium diameter to cell width to calculate a circle diameter. If a diameter of 50 and cell width of 5 is chosen, the resulting circle diameter is 10. Using the midpoint algorithm [111], `CreateBlob()` calculates the points of the circle and optionally fills the circle in. For the mesangium, this circle is filled in. The pixels of the circle are mapped to a cell width by cell width square section of the simulation space, where a cell is drawn for every pixel. To add in the initial set of podocytes, `CreateBlob()` is called again with the fill option disabled. This overwrites the outer

most layer of mesangial cells with podocytes.

The next step is placement of the capillaries. The `CreateCapillaries()` method identifies evenly spaced points in a circle around the center of the glomerulus. At each of those points, it creates a large circular cell in that position using the midpoint algorithm. In this case, instead of expanding the circle to draw multiple cells, the pixels are mapped directly to the simulation space to draw a single cell. This method can also evenly place the capillaries in a specified segment of the circle around the mesangium.

The last step has two parts. The `PlacePodocytesAroundCapillaries()` method starts by cleaning up after the `CreateCapillaries()`. An unintended consequence of `CreateCapillaries()` is that podocytes end up between capillaries and mesangial cells. For each capillary, `PlacePodocytesAroundCapillaries()` finds its neighboring cells and forces them to be mesangial cells. The method then finds the portion of each cell's border in contact with the medium. This exposed region is patched over with a single podocyte cell that is grown outwards in layers one pixel thick. The giant podocyte cell is split up into smaller chunks of a volume similar to the cells placed with the `CreateBlob()` function.

The final version of the algorithm has nine different parameters that can be adjusted to change how the geometry is generated. Examples of geometry produced after parameterizing this procedure are in Figure 4.7. The procedure makes several assumptions about how the cells are placed in relation to each other. These assumptions limit the amount of variability of the geometries produced by the procedure. As a result, I sought to generalize producing the geometry further.

4.2.2 Generalizing the Algorithm

The approach proposed in this section to generalize the generation of simulation geometry is based on the method used to generate outdoor maps in the game Path

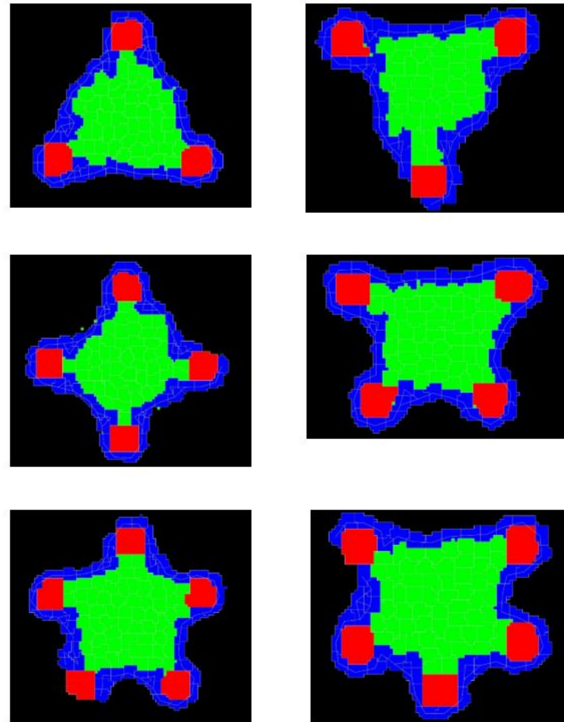


Figure 4.7: Simulations of procedurally generated glomeruli in the preliminary results for Aim II. In the left column, the capillaries are evenly spaced along the entire outside of the mesangium. The right column has an arc at the top excluded, and the capillaries are placed evenly along the included arc. The five capillary case with an excluded arc (bottom right) mimics the glomerulus of Figure 4.3.

of Exile [112]. Just like in Rogue [92], Path of Exile uses procedural generation to create the map of every playable area of the game. Unlike Rogue, Path of Exile uses its method to create variants of the same layout instead of completely randomized layouts. Each area may have more than one layout, but each time a map is created from the same layout, the key objectives are placed in the same general area.

Each layout is represented as a graph. A graph is an abstract network of nodes and edges, and the nodes and edges together represent how objects of a network are connected. Figure 4.8 is an example of one of these graphs and a map generated from it. Nodes are used represent key locations of the area such as exits, story objectives, or intersections. The edges are the paths between these locations and have types associated with them that determine the terrain that make up the paths, including

rivers that cannot normally be walked over. Since the edges are straight lines, nodes can also be used to indicate a bend in the path. The regions are filled in automatically based on the typing of the edges enclosing it [113].

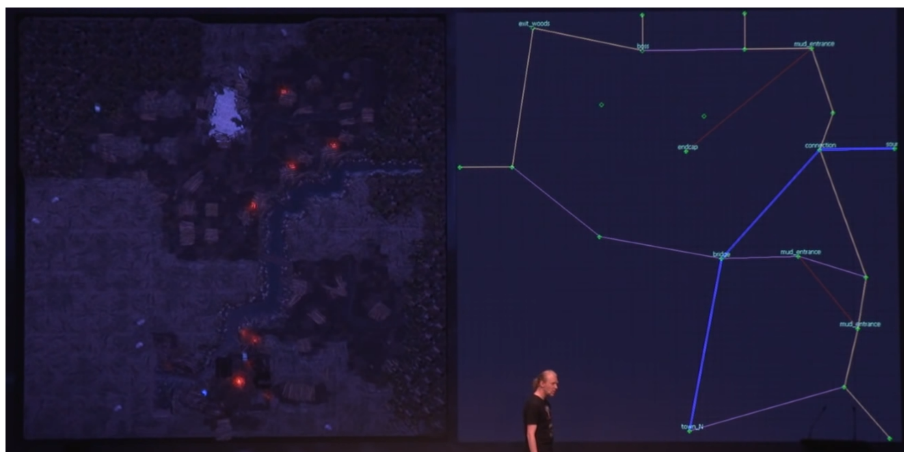


Figure 4.8: An overhead view of a playable area in Path of Exile [112] and the graph used to represent the area's layout presented in [113].

The terrain is built up by piecing together tiles. The tiles are square or rectangular portions of terrain that have been modeled by an artist. These tiles can be manually arranged into “rooms”, which can be prearranged tiles of an outdoor area. Typically rooms are used for the locations denoted by the nodes of the graph but can be included as interesting features dispersed through the map [113].

To translate the preceding discussion about Path of Exile to constructing geometries for our simulation, each of the elements need to be redefined in terms of drawing cells. The graph itself is an abstract representation of the tissue structure. The nodes of the graph represent key features of the tissue or reference points for edges. The edges represent the cells connecting the key features, or thin features of the tissue, such as the glomerular basement membrane. The idea of treating the cells as tiles and puzzling them together is less relevant. The tiles that make up the maps of Path of Exile need special consideration so the edges of each tile match up, allowing the terrain to remain continuous from tile to tile [113]. Instead of placing individual cells

as tiles, painting functions can be used to place cells. The concept of "rooms" is still useful, as manually arranged cells or custom painting functions allows for complex features. An example of a "room" in the glomerulus cross section is the function written to place capillaries and the podocytes around them. A useful extension of this method is to directly consider the regions of the graph. Instead of filling the regions based on the surrounding edges, a generalized painting function can be used to fill the region in with cells.

4.3 Conclusions

The algorithm incorporated into the glomerular tissue simulation is able to produce a variety of geometries useful for on whole glomerulus scale simulations. There are nine parameters in the code that may be adjusted to vary the geometry produced. However, even the current alogrithm is limited in the geometries it can produce. This method may be generalized to produce varied geometries from an arbitrary graph. This generalized algorithm may be a useful tool for other researchers developing tissue simulations. The code for the work discussed in this chapter is available at `https://github.com/ashleefv/GlomSliceGen` [114].

CHAPTER V

Recommendations for Future Work

5.1 Aim I: Establishing the Virtual Kidney

One effect that the transport model neglects is the effect of oncotic pressure on fluid flow. Modeling oncotic pressure requires coupling the solute transport to the CFD simulation. Directly coupling the Star-CCM+ [69] simulation with the CompuCell3D simulation was considerd, but Star-CCM+'s application programming interface made this too onerous of a task. OpenFOAM [115] is a CFD simulator that seems promising to incorporate into the virtual kidney project as an alternative to Star-CCM+.

Using the mesangial model to define solute concentrations in the bloodstream was a stopgap used to produce results that showcased the behavior of the transport model. Using a global renin-angiotensin system model for the bloodstream is recommended.

My work on the tissue model left off at designing the tissue geometry at the start of the simulation. The next step is to model the normal behavior of the cells, how the biochemical and transport models disrupts the cells' behavior, and how this propagates to the structural changes that result from cross talk.

5.2 Aim II: SBMLtoODEpy, An Open Source Systems Biology Markup Language to Python Converter

SBMLtoODEpy currently does not support **Events** and **Constraints**. The next major version update to the package should include supporting these model elements. These additions are on top of the regular maintenance expected as libSBML [78] and the SBML specifications [1] are routinely updated.

5.3 Aim III: Procedural Generation Methods in Tissue Simulation

While a prototype for the generalized algorithm presented in Aim III was started, Aim I diverged from the need for this prototype. Aim I was prioritized and grew to be time demanding to finish the prototype. Continuing this work require developing the prototype into functional software.

References

[1] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. L. Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, and the rest of the SBML Forum, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[2] V. Chelliah, N. Juty, I. Ajmera, R. Ali, M. Dumousseau, M. Glont, M. Hucka, G. Jalowicki, S. Keating, V. Knight-Schrijver, A. Lloret-Villas, K. N. Natarajan, J.-B. Pettit, N. Rodriguez, M. Schubert, S. M. Wimalaratne, Y. Zhao, H. Hermjakob, N. Le Novère, and C. Laibe, "BioModels: Ten-year anniversary," *Nucleic Acids Research*, vol. 43, no. D1, pp. D542–D548, 2015.

[3] M. Glont, T. V. N. Nguyen, M. Graesslin, R. Hälke, R. Ali, J. Schramm, S. M. Wimalaratne, V. B. Kothamachu, N. Rodriguez, M. J. Swat, J. Eils, R. Eils, C. Laibe, R. S. Malik-Sheriff, V. Chelliah, N. Le Novère, and H. Hermjakob, "BioModels: expanding horizons to include more modelling approaches and formats," *Nucleic Acids Research*, vol. 46, no. D1, p. D1248–D1253, 2018.

[4] G. Rossum, "Python reference manual," 1995.

[5] N. Rodriguez, J.-B. Pettit, P. Dalle Pezze, L. Li, A. Henry, M. P. van Iersel,

G. Jalowicki, M. Kutmon, K. N. Natarajan, D. Tolnay, M. I. Stefan, C. T. Evelo, and N. Le Novère, "The systems biology format converter," *BMC Bioinformatics*, vol. 17, no. 1, p. 154, 2016.

[6] S. M. Ruggiero, M. R. Pilvankar, and A. N. Ford Versypt, "Mathematical modeling of tuberculosis granuloma activation," *Processes*, vol. 5, no. 4, p. 79, 2017.

[7] D. Sud, C. Bigbee, J. L. Flynn, and D. E. Kirschner, "Contribution of CD8+ T cells to control of Mycobacterium tuberculosis infection," *The Journal of Immunology*, vol. 176, no. 7, pp. 4296–4314, 2006.

[8] S. M. Ruggiero, J. Zhao, and A. N. Ford Versypt, "Building a MATLAB graphical user interface to solve ordinary differential equations as a final project for an interdisciplinary elective course on numerical computing," *Journal of Computational Science*, vol. 9, no. 1, 2018.

[9] Mathworks, *MATLAB*. `https://www.mathworks.com/products/matlab.html`.

[10] M. Shacham, M. B. Cutlip, and M. Elly, *Polymath*. `https://www.polymath-software.com/`.

[11] R. Z. Alicic, M. T. Rooney, and K. R. Tuttle, "Diabetic kidney disease: challenges, progress, and possibilities," *Clinical Journal of the American Society of Nephrology*, vol. 12, no. 12, pp. 2032–2045, 2017.

[12] I. H. de Boer, T. C. Rue, Y. N. Hall, P. J. Heagerty, N. S. Weiss, and J. Himmelfarb, "Temporal trends in the prevalence of diabetic kidney disease in the united states," *JAMA*, vol. 305, no. 24, pp. 2532–2539, 2011.

[13] United States Renal Data System, "2019 USRDS Annual Data Report: Epidemiology of kidney disease in the United States," tech. rep., National Institutes

of Health, National Institute of Diabetes and Digestive and Kidney Diseases, Bethesda, MD, 2019.

[14] K. Patton and G. Thibodeau, *Anatomy & Physiology - E-Book*. Elsevier Health Sciences, 2014.

[15] H. Kawachi, N. Miyauchi, K. Suzuki, G. D. Han, M. Orikasa, and F. Shimizu, "Role of podocyte slit diaphragm as a filtration barrier (review article)," *Nephrology*, vol. 11, no. 4, pp. 274–281.

[16] K. Reidy, H. M. Kang, T. Hostetter, and K. Susztak, "Molecular mechanisms of diabetic kidney disease," *Journal of Clinical Investigation*, vol. 124, no. 6, pp. 2333–2340, 2014.

[17] W. Kriz, I. Shirato, M. Nagata, M. LeHir, and K. V. Lemley, "The podocyte's response to stress: the enigma of foot process effacement," *American Journal of Physiology-Renal Physiology*, vol. 304, no. 4, pp. F333–F347, 2013. PMID: 23235479.

[18] C. J. May, M. Saleem, and G. I. Welsh, "Podocyte dedifferentiation: a specialized process for a specialized cell," *Frontiers in Endocrinology*, vol. 5, p. 148, 2014.

[19] K. R. Tuttle, G. L. Bakris, R. W. Bilous, J. L. Chiang, I. H. De Boer, J. Goldstein-Fuchs, I. B. Hirsch, K. Kalantar-Zadeh, A. S. Narva, S. D. Navaneethan, *et al.*, "Diabetic kidney disease: a report from an ADA Consensus Conference," *American Journal of Kidney Diseases*, vol. 64, no. 4, pp. 510–533, 2014.

[20] J. C. He, P. Y. Chuang, A. Ma'Ayan, and R. Iyengar, "Systems biology of kidney diseases," *Kidney International*, vol. 81, no. 1, pp. 22–39, 2012.

[21] W. M. Deen, M. P. Bohrer, and B. M. Brenner, "Macromolecule transport across glomerular capillaries: application of pore theory," *Kidney international*, vol. 16, no. 3, pp. 353–365, 1979.

[22] A. Edwards, "Modeling transport in the kidney: investigating function and dysfunction," *American Journal of Physiology-Renal Physiology*, vol. 298, no. 3, pp. F475–F484, 2010.

[23] S. E. Hunt, K. D. Dorfman, Y. Segal, and V. H. Barocas, "A computational model of flow and species transport in the mesangium," *American Journal of Physiology-Renal Physiology*, vol. 310, no. 3, pp. F222–F229, 2016.

[24] A. T. Layton, "Mathematical modeling of kidney transport," *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, vol. 5, no. 5, pp. 557–573, 2013.

[25] S. R. Thomas, "Kidney modeling and systems physiology," *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, vol. 1, no. 2, pp. 172–190, 2009.

[26] A. M. Weinstein, "Mathematical models of renal fluid and electrolyte transport: acknowledging our uncertainty," *American Journal of Physiology-Renal Physiology*, vol. 284, no. 5, pp. F871–F884, 2003.

[27] M. C. Drumond, B. Kristal, B. D. Myers, W. M. Deen, *et al.*, "Structural basis for reduced glomerular filtration capacity in nephrotic humans.," *Journal of Clinical Investigation*, vol. 94, no. 3, pp. 1187–1195, 1994.

[28] H. Layton, J. M. Davies, G. Casotti, and E. J. Braun, "Mathematical model of an avian urine concentrating mechanism," *American Journal of Physiology-Renal Physiology*, vol. 279, no. 6, pp. F1139–F1160, 2000.

[29] D. E. Oken, S. R. Thomas, and D. C. Mikulecky, "A network thermodynamic model of glomerular dynamics: application in the rat," *Kidney International*, vol. 19, no. 2, pp. 359–373, 1981.

[30] J. Arciero, L. Ellwein, A. N. Ford Versypt, E. Makrides, and A. T. Layton, "Modeling blood flow control in the kidney," in *Applications of Dynamical Systems in Biology and Medicine*, pp. 55–73, Springer, 2015.

[31] R. Feldberg, M. Colding-Jorgensen, and N. Holstein-Rathlou, "Analysis of interaction between TGF and the myogenic response in renal blood flow autoregulation," *American Journal of Physiology-Renal Physiology*, vol. 269, no. 4, pp. F581–F593, 1995.

[32] K. M. Hallow, A. Lo, J. Beh, M. Rodrigo, S. Ermakov, S. Friedman, H. de Leon, A. Sarkar, Y. Xiong, R. Sarangapani, *et al.*, "A model-based approach to investigating the pathophysiological mechanisms of hypertension and response to antihypertensive therapies: extending the Guyton model," *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 306, no. 9, pp. R647–R662, 2014.

[33] A. Just, "Mechanisms of renal blood flow autoregulation: dynamics and contributions," *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 292, no. 1, pp. R1–R17, 2007.

[34] W. Hao, B. H. Rovin, and A. Friedman, "Mathematical model of renal interstitial fibrosis," *Proceedings of the National Academy of Sciences*, vol. 111, no. 39, pp. 14193–14198, 2014.

[35] J. Walpole, J. A. Papin, and S. M. Peirce, "Multiscale computational models of complex biological systems," *Annual Review of Biomedical Engineering*, vol. 15, pp. 137–154, 2013.

[36] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Transactions on software Engineering*, no. 3, pp. 259–266, 1985.

[37] E. Márquez, M. Riera, J. Pascual, and M. J. Soler, "Renin-angiotensin system within the diabetic podocyte," *American Journal of Physiology-Renal Physiology*, vol. 308, no. 1, pp. F1–F10, 2015.

[38] J. C. Q. Velez, A. M. Bland, J. M. Arthur, J. R. Raymond, and M. G. Janech, "Characterization of renin-angiotensin system enzyme activities in cultured mouse podocytes," *American Journal of Physiology-Renal Physiology*, vol. 293, no. 1, pp. F398–F407, 2007.

[39] R. V. Durvasula and S. J. Shankland, "Activation of a local renin angiotensin system in podocytes by glucose," *American Journal of Physiology - Renal Physiology*, vol. 294, no. 4, pp. F830–F839, 2008.

[40] T.-H. Yoo, J.-J. Li, J.-J. Kim, D.-S. Jung, S.-J. Kwak, D.-R. Ryu, H. Y. Choi, J. S. Kim, H. J. Kim, S. H. Han, J. E. Lee, D. S. Han, and S.-W. Kang, "Activation of the renin-angiotensin system within podocytes in diabetes," *Kidney International*, vol. 71, no. 10, pp. 1019–1027, 2007.

[41] T. M. Wendt, N. Tanji, J. Guo, T. R. Kislinger, W. Qu, Y. Lu, L. G. Bucciarelli, L. L. Rong, B. Moser, G. S. Markowitz, *et al.*, "RAGE drives the development of glomerulosclerosis and implicates podocyte activation in the pathogenesis of diabetic nephropathy," *The American Journal of Pathology*, vol. 162, no. 4, pp. 1123–1137, 2003.

[42] K. Fukami, S. Ueda, S.-I. Yamagishi, S. Kato, Y. Inagaki, M. Takeuchi, Y. Motomiya, R. Bucala, S. Iida, K. Tamaki, *et al.*, "AGEs activate mesangial TGF-

β–smad signaling via an Angiotensin II type I receptor interaction," *Kidney International*, vol. 66, no. 6, pp. 2137–2147, 2004.

[43] S. Chen, J. S. Lee, M. Iglesias-De La Cruz, A. Wang, A. Izquierdo-Lahuerta, N. K. Gandhi, F. R. Danesh, G. Wolf, and F. N. Ziyadeh, "Angiotensin II stimulates α3 (iv) collagen production in mouse podocytes via TGF-β and vegf signalling: implications for diabetic glomerulopathy," *Nephrology Dialysis Transplantation*, vol. 20, no. 7, pp. 1320–1328, 2005.

[44] R. Singh, N. Alavi, A. K. Singh, and D. J. Leehey, "Role of angiotensin ii in glucose-induced inhibition of mesangial matrix degradation.," *Diabetes*, vol. 48, no. 10, pp. 2066–2073, 1999.

[45] M. R. Pilvankar, *Mechanistis Computational Modeling For Pathophysiology and Pharmacology: Case Studies of Diabetic Kidney Disease, Latent Tuberculosis, and Opioids*. PhD thesis, Oklahoma State University, 2019.

[46] M. R. Pilvankar, M. A. Higgins, and A. N. Ford Versypt, "Mathematical model for glucose dependence of the local renin-angiotensin system in podocytes," *Bulletin of Mathematical Biology*, vol. 80, no. 4, pp. 880–905, 2018.

[47] M. R. Pilvankar, H. L. Yong, and A. N. Ford Versypt, "A glucose-dependent pharmacokinetic/pharmacodynamic model of ace inhibition in kidney cells," *Processes*, vol. 7, no. 3, p. 131, 2019.

[48] M. H. Swat, G. L. Thomas, J. M. Belmonte, A. Shirinifard, D. Hmeljak, and J. A. Glazier, "Multi-scale modeling of tissues using CompuCell3D," *Computational Methods in Cell Biology*, vol. 110, p. 325, 2012.

[49] F. Graner and J. A. Glazier, "Simulation of biological cell sorting using a two-dimensional extended potts model," *Physical Review Letters*, vol. 69, no. 13, p. 2013, 1992.

[50] J. A. Glazier and F. Graner, "Simulation of the differential adhesion driven rearrangement of biological cells," *Physical Review E*, vol. 47, no. 3, p. 2128, 1993.

[51] R. B. Potts, "Some generalized order-disorder transformations," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 48, pp. 106–109, Cambridge University Press, 1952.

[52] F.-Y. Wu, "The Potts model," *Reviews of Modern Physics*, vol. 54, no. 1, p. 235, 1982.

[53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[54] M. H. Swat, J. Belmonte, T. Sego, and J. Glazier, *CompuCell3D Reference Manual*. Indiana University and The Biocomplexity Institute, 2020. `https://compucell3dreferencemanual.readthedocs.io/en/latest/index.html`.

[55] M. H. Swat, J. Belmonte, and J. Glazier, *Python Scripting Manual for CompuCell3D*. Indiana University and The Biocomplexity Institute, 2017. `https://pythonscriptingmanual.readthedocs.io/en/4.1.1/index.html`.

[56] J. E. Guyer, D. Wheeler, and J. A. Warren, "Fipy: Partial differential equations with python," *Computing in Science & Engineering*, vol. 11, no. 3, pp. 6–15, 2009.

[57] J. E. Guyer, D. Wheeler, and J. A. Warren, *FiPy: A Finite Volume PDE Solver Using Python*. NIST, 2020. `https://www.ctcms.nist.gov/fipy/`.

[58] K. Ichimura, N. Miyazaki, S. Sadayama, K. Murata, M. Koike, K.-i. Nakamura, K. Ohta, and T. Sakai, "Three-dimensional architecture of podocytes revealed

by block-face scanning electron microscopy," *Scientific Reports*, vol. 5, p. 8993, 2015.

[59] I. Grgic, C. R. Brooks, A. F. Hofmeister, V. Bijol, J. V. Bonventre, and B. D. Humphreys, "Imaging of podocyte foot processes by fluorescence microscopy," *Journal of the American Society of Nephrology*, vol. 23, no. 5, pp. 785–791, 2012.

[60] V. Ruotsalainen, P. Ljungberg, J. Wartiovaara, U. Lenkkeri, M. Kestilä, H. Jalanko, C. Holmberg, and K. Tryggvason, "Nephrin is specifically located at the slit diaphragm of glomerular podocytes," *Proceedings of the National Academy of Sciences*, vol. 96, no. 14, pp. 7962–7967, 1999.

[61] A. Ortega, D. Amorós, and J. G. De La Torre, "Prediction of hydrodynamic and other solution properties of rigid proteins from atomic-and residue-level models," *Biophysical Journal*, vol. 101, no. 4, pp. 892–898, 2011.

[62] H. Fischer, I. Polikarpov, and A. F. Craievich, "Average protein density is a molecular-weight-dependent function," *Protein Science*, vol. 13, no. 10, pp. 2825–2828, 2004.

[63] G. A. Spyroulias, P. Nikolakopoulou, A. Tzakos, I. P. Gerothanassis, V. Magafa, E. Manessi-Zoupa, and P. Cordopatis, "Comparison of the solution structures of angiotensin i & ii: Implication for structure-function relationship," *European Journal of Biochemistry*, vol. 270, no. 10, pp. 2163–2173, 2003.

[64] A. Zhou, R. W. Carrell, M. P. Murphy, Z. Wei, Y. Yan, P. L. Stanley, P. E. Stein, F. B. Pipkin, and R. J. Read, "A redox switch in angiotensinogen modulates angiotensin release," *Nature*, vol. 468, no. 7320, pp. 108–111, 2010.

[65] Y. Imaeda, H. Tokuhara, Y. Fukase, R. Kanagawa, Y. Kajimoto, K. Kusumoto, M. Kondo, G. Snell, C. A. Behnke, and T. Kuroita, "Discovery of tak-272: a

novel, potent, and orally active renin inhibitor," *ACS Medicinal Chemistry Letters*, vol. 7, no. 10, pp. 933–938, 2016.

[66] A. P. Hinck, S. J. Archer, S. W. Qian, A. B. Roberts, M. B. Sporn, J. A. Weatherbee, M. L.-S. Tsang, R. Lucas, B.-L. Zhang, J. Wenker, *et al.*, "Transforming growth factor β1: three-dimensional structure in solution and comparison with the x-ray structure of transforming growth factor β2," *Biochemistry*, vol. 35, no. 26, pp. 8517–8534, 1996.

[67] E. Morgunova, A. Tuuttila, U. Bergmann, M. Isupov, Y. Lindqvist, G. Schneider, and K. Tryggvason, "Structure of human pro-matrix metalloproteinase-2: activation mechanism revealed," *Science*, vol. 284, no. 5420, pp. 1667–1670, 1999.

[68] K. B. Kosto and W. M. Deen, "Hindered convection of macromolecules in hydrogels," *Biophysical Journal*, vol. 88, no. 1, pp. 277–286, 2005.

[69] Siemens PLM Software, *STAR-CCM+*, 2004. `https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html`.

[70] A. Denic, J. Mathew, L. O. Lerman, J. C. Lieske, J. J. Larson, M. P. Alexander, E. Poggio, R. J. Glassock, and A. D. Rule, "Single-nephron glomerular filtration rate in healthy adults," *New England Journal of Medicine*, vol. 376, no. 24, pp. 2349–2357, 2017.

[71] A. Tobar, Y. Ori, S. Benchetrit, G. Milo, M. Herman-Edelstein, B. Zingerman, N. Lev, U. Gafter, and A. Chagnac, "Proximal tubular hypertrophy and enlarged glomerular and proximal tubular urinary space in obese subjects with proteinuria," *PLoS One*, vol. 8, no. 9, p. e75547, 2013.

[72] F. Büchel, N. Rodriguez, N. Swainston, C. Wrzodek, T. Czauderna, R. Keller, F. Mittag, M. Schubert, M. Glont, M. Golebiewski, M. van Iersel, S. Keat-

ing, M. Rall, M. Wybrow, H. Hermjakob, M. Hucka, D. B. Kell, W. Müller, P. Mendes, A. Zell, C. Chaouiya, J. Saez-Rodriguez, F. Schreiber, C. Laibe, A. Dräger, and N. Le Novère, "Path2Models: large-scale generation of computational models from biochemical pathway maps.," *BMC Systems Biology*, vol. 7, p. 116, Nov 2013.

[73] E. T. Somogyi, J.-M. Bouteiller, J. A. Glazier, M. König, J. K. Medley, M. H. Swat, and H. M. Sauro, "libRoadRunner: a high performance SBML simulation and analysis library," *Bioinformatics*, vol. 31, no. 20, pp. 3315–3321, 2015.

[74] P. Munz, I. Hudea, J. Imad, and R. J. Smith?, "When zombies attack!: mathematical modelling of an outbreak of zombie infection," *Infectious Disease Modelling Research Progress*, vol. 4, pp. 133–150, 2009.

[75] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, *et al.*, "Extensible markup language (XML) 1.0," 2000.

[76] M. Hucka, F. T. Bergmann, C. Chaouiya, A. Dräger, S. Hoops, S. M. Keating, M. König, N. Le Novère, C. J. Myers, B. G. Olivier, *et al.*, "The systems biology markup language (SBML): language specification for level 3 version 2 core release 2," *Journal of Integrative Bioinformatics*, vol. 16, no. 2, 2019.

[77] S. M. Keating and N. Le Novère, *Supporting SBML as a Model Exchange Format in Software Applications*, pp. 201–225. Totowa, NJ: Humana Press, 2013.

[78] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka, "LibSBML: an API library for SBML," *Bioinformatics*, vol. 24, no. 6, pp. 880–881, 2008.

[79] S. M. Ruggiero and A. N. Ford Versypt, "SMBLtoODEpy: A software program for converting SBML models into ODE models in Python," *Journal of Open Source Software*, vol. 4, no. 41, p. 1643, 2019.

[80] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[81] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[82] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer, "COPASI: a COmplex PAthway SImulator," *Bioinformatics*, vol. 22, no. 24, pp. 3067–3074, 2006.

[83] N. Borisov, E. Aksamitiene, A. Kiyatkin, S. Legewie, J. Berkhout, T. Maiwald, N. P. Kaimachnikov, J. Timmer, J. B. Hoek, and B. N. Kholodenko, "Systems-level interactions between insulin–EGF networks amplify mitogenic signaling," *Molecular Systems Biology*, vol. 5, p. 256, 2009.

[84] A. C. Guyton, T. G. Coleman, and H. J. Granger, "Circulation: Overall regulation," *Annual Review of Physiology*, vol. 34, pp. 13–44, 1972.

[85] E. J. Kerkhoven, F. Achcar, V. P. Alibu, R. J. Burchmore, I. H. Gilbert, M. Trybiło, N. N. Driessen, D. Gilbert, R. Breitling, B. M. Bakker, and M. P. Barrett, "Handling uncertainty in dynamic models: The pentose phosphate pathway in Trypanosoma brucei," *PLoS Computational Biology*, vol. 9, no. 12, p. e1003371, 2013.

[86] K. Smallbone and B. M. Corfe, "A mathematical model of the colon crypt capturing compositional dynamic interactions between cell types," *International Journal of Experimental Pathology*, vol. 95, no. 1, pp. 1–7, 2014.

[87] H. V. Waugh and J. A. Sherratt, "Macrophage dynamics in diabetic wound dealing," *Bulletin of Mathematical Biology*, vol. 68, no. 1, pp. 197–207, 2006.

[88] Z. Zi, Z. Feng, D. A. Chapnick, M. Dahl, D. Deng, E. Klipp, A. Moustakas, and X. Liu, "Quantitative analysis of transient and sustained transforming growth factor-β signaling dynamics," *Molecular Systems Biology*, vol. 7, no. 1, p. 492, 2011.

[89] A. C. Hindmarsh, "ODEPACK, a systematized collection of ODE solvers," in *Scientific Computing: Applications of Mathematics and Computing to the Physical Sciences* (R. S. Stepleman, M. Carver, R. Peskin, W. F. Ames, and R. Vichnevetsky, eds.), pp. 55–64, New York: North-Holland, 1983.

[90] S. M. Ruggiero and A. N. Ford Versypt, *SMBLtoODEpy: A software program for converting SBML models into ODE models in Python*, 2019. https://github.com/SMRuggiero/sbmltoodepy.

[91] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. Springer, 2016.

[92] M. Toy, G. Wichman, K. Arnold, and J. Lane, *Rouge*. A.I. Design, 1984. https://archive.org/details/msdos_Rogue_1983.

[93] K. Compton, J. C. Osborn, and M. Mateas, "Generative methods," in *The Fourth Procedural Content Generation in Games workshop, PCG*, vol. 1, 2013.

[94] J. Togelius, J. Whitehead, and R. Bidarra, "Guest editorial: Procedural content generation in games," *IEEE Transactions on Computational Intelligence and AI in Games*, no. 3, pp. 169–171, 2011.

[95] G. C. Onwubolu and B. Babu, *New Optimization Techniques in Engineering*, vol. 141. Springer, 2013.

[96] E. M. Hendrix, G. Boglárka, *et al.*, *Introduction to Nonlinear and Global Optimization*, vol. 37. Springer, 2010.

[97] B. P. Buckles and F. Petry, *Genetic Algorithms.* IEEE Computer Society Press, 1992.

[98] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. MIT press, 1992.

[99] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization.* New York: John Wily and Sons, 2000.

[100] Y.-C. B. Fung, “Stress-strain-history relations of soft tissues in simple elongation,” *Biomechanics Its Foundations and Objectives*, pp. 181–208, 1972.

[101] M. Kohandel, S. Sivaloganathan, and G. Tenti, “Estimation of the quasi-linear viscoelastic parameters using a genetic algorithm,” *Mathematical and Computer Modelling*, vol. 47, no. 3-4, pp. 266–270, 2008.

[102] H.-Y. Ling, P.-C. Choi, Y.-P. Zheng, and K.-T. Lau, “Extraction of mechanical properties of foot plantar tissues using ultrasound indentation associated with genetic algorithm,” *Journal of Materials Science: Materials in Medicine*, vol. 18, no. 8, pp. 1579–1586, 2007.

[103] J. Togelius, R. De Nardi, and S. M. Lucas, “Towards automatic personalised content creation for racing games,” in *2007 IEEE Symposium on Computational Intelligence and Games*, pp. 252–259, IEEE, 2007.

[104] N. Uesugi, Y. Shimazu, T. Aoba, K. Kikuchi, and M. Nagata, “High-resolution three-dimensional digital imaging of the human renal microcirculation: An aid to evaluating microvascular alterations in chronic kidney disease in humans,” *Pathology International*, vol. 65, no. 11, pp. 575–584, 2015.

[105] L. Han, R. F. Murphy, and D. Ramanan, “Learning generative models of tissue organization with supervised gans,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 682–690, IEEE, 2018.

[106] R. F. Murphy, "Building cell models and simulations from microscope images," *Methods*, vol. 96, pp. 33–39, 2016.

[107] D. Svoboda, O. Homola, and S. Stejskal, "Generation of 3D digital phantoms of colon tissue," in *International Conference Image Analysis and Recognition*, pp. 31–39, Springer, 2011.

[108] T. Jebara, *Machine Learning : Discriminative and Generative*. Kluwer international series in engineering and computer science ; SECS 755, Boston: Kluwer Academic Publishers.

[109] G. Bueno, M. M. Fernandez-Carrobles, L. Gonzalez-Lopez, and O. Deniz, "Glomerulosclerosis identification in whole slide images using semantic segmentation," *Computer Methods and Programs in Biomedicine*, vol. 184, p. 105273, 2020.

[110] J. Gallego, A. Pedraza, S. Lopez, G. Steiner, L. Gonzalez, A. Laurinavicius, and G. Bueno, "Glomerulus classification and detection based on convolutional neural networks," *Journal of Imaging*, vol. 4, no. 1, p. 20, 2018.

[111] J. R. Van Aken, "An efficient ellipse-drawing algorithm," *IEEE Computer Graphics and Applications*, vol. 4, no. 9, pp. 24–35, 1984.

[112] Grinding Gear Games, *Path of Exile*. Grinding Gear Games, 2013. `https://www.pathofexile.com/`.

[113] R. Abraham, *Procedural World Generation in Path of Exile*. Grinding Gear Games, 2019. `https://www.youtube.com/watch?v=EXnoHTqO7TE`.

[114] S. M. Ruggiero and A. N. Ford Versypt, *GlomSliceGen*, 2020. `https://github.com/ashleefv/GlomSliceGen`.

[115] H. Jasak, A. Jemcov, Z. Tukovic, *et al.*, "OpenFOAM: A C++ library for complex physics simulations," in *International Workshop on Coupled Methods in Numerical Dynamics*, vol. 1000, pp. 1–20, IUC Dubrovnik Croatia, 2007.

[116] A. N. Ford Versypt, G. K. Harrell, and A. N. McPeak, "A pharmacokinetic/pharmacodynamic model of ACE inhibition of the renin-angiotensin system for normal and impaired renal function," *Computers & Chemical Engineering*, vol. 104, pp. 311–322, 2017.

[117] A. Lo, J. Beh, H. De Leon, M. K. Hallow, R. Ramakrishna, M. Rodrigo, A. Sarkar, R. Sarangapani, and A. Georgieva, "Using a systems biology approach to explore hypotheses underlying clinical diversity of the renin angiotensin system and the response to antihypertensive therapies," in *Clinical Trial Simulations* (H. H. C. Kimko and C. C. Peck, eds.), pp. 457–482, New York: Springer, 2011.

APPENDIX A

Podocyte and Mesangial Cell Models

The work presented in this chapter Dr. Minu Pilvankar's podocyte model [47] and Ashlea Sartin's model of TGF-β1 and MMP-2 regulation in the mesangial cell. These are the biochemical models incorporated into the work presented in ChapterII.

The mass balance for AGT is

$$\frac{d[\text{AGT}]}{dt} = k_{\text{AGT}} - c_{\text{Renin}}[\text{AGT}] - \frac{\ln 2}{h_{\text{AGT}}}[\text{AGT}] \tag{1.1}$$

where k_{AGT} is the constant production rate of AGT, c_{Renin} is the glucose-dependent rate parameter for renin-catalyzed conversion of AGT to ANG I, and h_{AGT} is the half-life for the degradation of AGT [46].

The mass balance for renin is

$$\begin{aligned}\frac{d[\text{Renin}]}{dt} = {} & s_{\text{Renin}} + k_f([\text{ANG II}]_0 - [\text{ANG II}])\left(1 - \frac{[\text{ANG II}]_0 - [\text{ANG II}]}{f}\right) \\ & - \frac{\ln 2}{h_{\text{Renin}}}[\text{Renin}]\end{aligned} \tag{1.2}$$

where s_{Renin} is the constant source of renin in the absence of feedback, the second term is the influence of ANG II negative feedback on renin production, $[\text{ANG II}]_0$ is the initial concentration of ANG II, k_f and f are parameters for the feedback, and h_{Renin} is the half-life for the degradation of renin [116]. The renin source term is computed at steady state by

$$s_{\text{Renin}} = \frac{\ln 2}{h_{\text{Renin}}}[\text{Renin}]_0 \tag{1.3}$$

where $[\text{Renin}]_0$ is the initial concentration of renin [116].

The mass balance for ANG I is

$$\begin{aligned}\frac{d[\text{ANG I}]}{dt} = {} & c_{\text{Renin}}[\text{AGT}] + k_{\text{Renin}}([\text{Renin}] - [\text{Renin}]_0) - c_{\text{ACE}}[\text{ANG I}] \\ & - (k_{\text{NEP}} + k_{\text{ACE2}})[\text{ANG I}] - \frac{\ln 2}{h_{\text{ANG I}}}[\text{ANG I}]\end{aligned} \tag{1.4}$$

where the first term represents the glucose-dependent renin-catalyzed contribution to the production of ANG I from AGT, the second term represents the change to ANG I synthesis from AGT due to the feedback of ANG II on renin with rate constant k_{Renin}, the third term is the ACE-catalyzed conversion of ANG I to ANG II that has a glucose-dependent rate parameter c_{ACE}, the fourth term is the consumption of ANG I

to form ANG-(1-7) and ANG-(1-9) with the glucose-independent rate parameters k_{NEP} and k_{ACE2}, respectively, and $h_{\mathrm{ANG\,I}}$ is the half-life for the degradation of ANG I. The first and second terms in (1.4) both consider the production of ANG I from AGT catalyzed by renin. The first term is from glucoseRASpodocytes and includes glucose dependence in the absence of the ANG II-renin feedback mechanism. The second term is from ACEInhibPKPD to incorporate the effects of the feedback mechanism on changing the production rate [46, 116].

The mass balance for ANG II is

$$\frac{d[\mathrm{ANG\ II}]}{dt} = c_{\mathrm{ACE}}[\mathrm{ANG\ I}] - (c_{\mathrm{AT1}} + k_{\mathrm{AT2}} + k_{\mathrm{APA}} + k_{\mathrm{ACE2}})[\mathrm{ANG\ II}] - \frac{\ln 2}{h_{\mathrm{ANG\,II}}}[\mathrm{ANG\ II}] \tag{1.5}$$

where the first term is the production of ANG II in the presence of ACE that is subject to inhibition I by the drug; the second term is the consumption of ANG II as it converts into downstream peptides and binds to AT1 and AT2 receptors; c_{AT1} is the glucose-dependent rate parameter for binding of ANG II to AT1 receptor; k_{AT2}, k_{APA}, and k_{ACE2} are the glucose-independent rate parameters for binding of ANG II to the AT2 receptor and conversion of ANG II to ANG III and ANG-(1-7), respectively; and $h_{\mathrm{ANG\,II}}$ is the half-life for degradation of ANG II [46, 116].

Glucose dependency is included in the parameters designated by c_x, where $x =$ Renin, ACE, and AT1. In glucoseRASpodocytes [46] these parameters were found to be the most sensitive and were each considered to be a linear function of glucose. The glucose-dependent parameters are calculated using [46]

$$c_{\mathrm{Renin}} = a_{\mathrm{Renin}}[\mathrm{GLU}] + b_{\mathrm{Renin}} \tag{1.6}$$

$$c_{\mathrm{ACE}} = a_{\mathrm{ACE}}[\mathrm{GLU}] + b_{\mathrm{ACE}} \tag{1.7}$$

$$c_{\mathrm{AT1}} = a_{\mathrm{AT1}}[\mathrm{GLU}] + b_{\mathrm{AT1}} \tag{1.8}$$

where [GLU] is the concentration of glucose and the coefficients a_x and b_x in (1.6)–(1.8) are obtained from parameter estimation detailed in [46].

TGF-β production is upregulated by an increase in the concentration of angiotensin II. This change in TGF-β with respect to time is described by

$$\frac{d[\mathrm{TGF}]}{dt} = B_T + \frac{k_1([\mathrm{ANGII}] - [\mathrm{ANGII}]_{NG})}{([\mathrm{ANGII}] - [\mathrm{ANGII}]_{NG}) + a_1} - k_T[\mathrm{TGF}] \tag{1.9}$$

where [TGF] is the concentration of TGF-β, B_T is the baseline production rate of TGF-β, k_1 is the rate constant of angiotensin II stimulated production of TGF-β, $[\mathrm{ANGII}]_{NG}$ is the concentration of Angiotensin-II at normal glucose levels, a_2 is the half-saturation constant of the same process, and k_T is defined as

$$k_T = \frac{ln(2)}{h_T} \tag{1.10}$$

Table A.1: Pharmacodynamic parameter values for the podocyte local RAS model.

Parameter	Value	Units	Sources
k_{AGT}	2.27×10^{6}	nmol/L/h	[46]
k_{Renin}	6.44×10^{4}	h^{-1}	[116]
k_{NEP}	0.583	h^{-1}	[46]
k_{ACE2}	0.382	h^{-1}	[46]
k_{AT2}	25.1	h^{-1}	[46]
k_{APA}	43.6	h^{-1}	[46]
h_{AGT}	10.0	h	[46, 117]
h_{Renin}	0.250	h	[116]
$h_{\text{ANG I}}$	1.72×10^{-4}	h	[46, 117]
$h_{\text{ANG II}}$	5.00×10^{-3}	h	[46, 117]
a_{Renin}	5.47×10^{-4}	L/mmol/h	[46]
b_{Renin}	6.14×10^{-2}	h^{-1}	[46]
a_{ACE}	0.889	L/mmol/h	[46]
b_{ACE}	16.3	h^{-1}	[46]
a_{AT1}	2.55	L/mmol/h	[46]
b_{AT1}	46.4	h^{-1}	[46]
k_f	4.92×10^{-5}	h^{-1}	[116]
f	5.05×10^{-4}	nmol/L	[116]
$[\text{AGT}]_0$	1.70×10^{4}	nmol/L	[46]
$[\text{Renin}]_0$	2.06×10^{-4}	nmol/L	[116]
$[\text{ANG I}]_0$	0.271	nmol/L	[46]
$[\text{ANG II}]_0$	2.10×10^{-2}	nmol/L	[46]
$[\text{ANG II}]_{0,\text{ sys NRF}}$	1.65×10^{-2}	nmol/L	[116]
$[\text{ANG II}]_{0,\text{ sys IRF}}$	2.05×10^{-2}	nmol/L	[116]

where h_T is the half-life of TGF-β. B_T is defined as

$$B_T = k_T[\text{TGF}]_{NG} \tag{1.11}$$

where $[\text{TGF}]_{NG}$ is the concentration of TGF-β at normal glucose levels.

The upregulation of TGF-β in turn causes an down regulation of MMP represented by

$$\frac{d[\text{MMP}]}{dt} = B_M + \frac{k_2}{[\text{TGF}] + a_2} - k_M([\text{MMP}]) \tag{1.12}$$

where [MMP] is the concentration of MMP, B_M is the baseline production rate of MMP, k_2 is the rate constant of TGF-β inhibited MMP production, a_2 is the half-saturation constant of TGF-β inhibition, and k_M is defined as

$$k_{[\text{MMP}]} = \frac{\ln(2)}{h_M} \tag{1.13}$$

where h_M is the half-life of MMP. B_M is defined as

$$B_M = k_M[\text{MMP}]_{NG} - \frac{k_2}{[\text{TGF}]_{NG} + a_2} \tag{1.14}$$

where $[\text{MMP}]_{NG}$ is the concentration of MMP at normal glucose levels.

Table A.2: Parameters for equations (1.9) and (1.12).

Parameter	Value	Units	Sources
B_T	$4.55 * 10^{-4}$	mmol/L/h	-
k_1	$3.18 * 10^{-2}$	mmol/L/h	-
a_1	$1.08 * 10^{-5}$	mmol/L	-
k_T	0.462	h^{-1}	-
h_T	1.5	h	[44]
B_M	$4.55 * 10^{-4}$	mmol/L/h	-
k_2	$3.70 * 10^{-6}$	$mmol^2/L^2/h$	-
a_2	$-2.04 * 10^{-2}$	mmol/L	-
k_M	$1.51 * 10^{-2}$	h^{-1}	-
h_M	46	h	[44]

VITA

STEVEN MACRAE RUGGIERO

Candidate for the Degree of

Doctor of Philosophy

Thesis: DEVELOPMENT OF TOOLS TO ACCELERATE AND ADVANCE MODELING DISEASE PROGRESSION

Major Field: Chemical Engineering

Biographical:

Education:

- Completed the requirements for the degree of Doctor of Philosophy with a major in Chemical Engineering at Oklahoma State University in Dec 2020.
- Received a Bachelors of Science in Chemical Engineering at Rowan University, Glassboro, New Jersey in May 2016.

Honors and Awards:

- Robert L. Robertson Jr. Endowed Fellowship Award, OSU, 2019
- Walter Kolb Graduate Studies Scholarship, OSU, 2018