



# VCU

Virginia Commonwealth University  
VCU Scholars Compass

---

Theses and Dissertations

Graduate School


---

2017

## Computational Fluid Dynamics in a Terminal Alveolated Bronchiole Duct with Expanding Walls: Proof-of-Concept in OpenFOAM

Jeremy Myers  
*Virginia Commonwealth University*

Follow this and additional works at: <https://scholarscompass.vcu.edu/etd>

 Part of the [Numerical Analysis and Computation Commons](#), [Numerical Analysis and Scientific Computing Commons](#), [Partial Differential Equations Commons](#), and the [Respiratory System Commons](#)

© The Author

---

Downloaded from

<https://scholarscompass.vcu.edu/etd/5011>

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact [libcompass@vcu.edu](mailto:libcompass@vcu.edu).

Copyright ©2017 by Jeremy Moulton Myers  
All rights reserved

# COMPUTATIONAL FLUID DYNAMICS IN A TERMINAL ALVEOLATED BRONCHIOLE DUCT WITH EXPANDING WALLS: PROOF-OF-CONCEPT IN OPENFOAM

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

by

**Jeremy Moulton Myers**

B.S. Virginia Commonwealth University, 2014

B.A. James Madison University, 2009

Director: Dr. Rebecca Segal, Associate Professor  
Department of Mathematics and Applied Mathematics



Virginia Commonwealth University  
Richmond, Virginia  
August 2017

## Acknowledgements

I would like to thank all the people who contributed in some way to the success of this thesis. Foremost, I would like to express great thanks to my thesis advisor, Dr. Rebecca Segal, for an invaluable learning experience that has set my life on a new and interesting trajectory. I want to thank Dr. Laura Ellwein-Fix for serving on my thesis committee, treating me as a peer in our working group in parameter estimation, and advancing my interest in numerical linear algebra. I also want to thank Dr. Angela Reynolds for serving as an unofficial advisor and also as a peer in our parameter estimation working group. I have thought of these three as my mentors for the past year and I look forward to staying in touch for years to come. Special thanks goes to Dr. Rebecca Heise for serving on my thesis committee.

I would like to thank my parents for encouraging me to keep pushing myself.

Last, I would like to thank Tori Hovater for supporting me, keeping my sanity, and showing endless patience. And thanks to our dogs, Briggs and Pungo!

# Table of Contents

<b>Acknowledgements</b>	<b>iii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>Abstract</b>	<b>xi</b>
<b>1 Introduction and Overview</b>	<b>1</b>
1.1 Research Summary . . . . .	3
1.2 Literature Review . . . . .	3
1.2.1 Lung Physiology . . . . .	3
1.2.2 Geometric Lung Models . . . . .	5
1.2.3 In-Vitro Modeling . . . . .	6
1.2.4 Computational Modeling . . . . .	7
1.3 Overview . . . . .	8
<b>2 The Mathematics of Fluid Mechanics and Numerical Analysis</b>	<b>10</b>
2.1 Navier-Stokes Equations . . . . .	11
2.1.1 Reynolds Number . . . . .	12
2.2 The Finite Element Method . . . . .	13
2.2.1 The Weak form of a Boundary Value Problem . . . . .	13

2.2.2	The Galerkin Method . . . . .	18
2.2.3	The Galerkin Finite Element Method . . . . .	20
2.2.4	Solving $KU = F$ with a Krylov method . . . . .	22
2.3	Finite Volume Method . . . . .	23
2.4	Numerical Algorithms for Solving the Navier-Stokes Equations . . . . .	25
2.4.1	SIMPLE . . . . .	26
2.4.2	PISO . . . . .	26
<b>3</b>	<b>Methods</b>	<b>28</b>
3.1	OpenFOAM for Computational Fluid Dynamics . . . . .	29
3.1.1	Overview . . . . .	29
3.1.2	OpenFOAM as a General Computational Engine . . . . .	30
3.1.3	OpenFOAM Case File Structure . . . . .	31
3.2	Case Models . . . . .	33
3.2.1	Duct Models . . . . .	34
3.2.2	Airway Flow Models . . . . .	35
3.3	Mesh Generation with Gmsh . . . . .	39
3.4	Post-Processing in ParaView . . . . .	43
<b>4</b>	<b>Results</b>	<b>44</b>
4.1	Duct Model . . . . .	44
4.1.1	Case A: Steady-State Flow and Case B: Constant Transient Flow . . . . .	44
4.1.2	Case C: Oscillating Transient Flow . . . . .	44
4.2	Airway Model . . . . .	47
4.2.1	Case D: Steady-State Flow and Case E: Constant Transient Flow . . . . .	47
4.2.2	Case F: Oscillating Transient Flow . . . . .	48
4.2.3	Case G: Transient Flow with Expanding Boundary . . . . .	50

<b>5 Discussion</b>	<b>54</b>
5.1 Airway Model Comparisons . . . . .	55
5.1.1 Steady State vs. Transient Flow . . . . .	55
5.1.2 Transient Flow With and Without Periodic Boundary Conditions . . . . .	56
5.1.3 Pressure-Driven Flow with Moving Boundary . . . . .	58
5.2 Challenges . . . . .	60
5.3 Future Work . . . . .	63
5.4 Conclusions . . . . .	64
<b>Bibliography</b>	<b>66</b>
<b>A OpenFOAM Code</b>	<b>70</b>
A.1 Case A . . . . .	70
A.1.1 /0 . . . . .	70
A.1.2 /constant . . . . .	74
A.1.3 /system . . . . .	76
A.2 Case B . . . . .	80
A.2.1 /0 . . . . .	80
A.2.2 /constant . . . . .	82
A.2.3 /system . . . . .	83
A.3 Case C . . . . .	87
A.3.1 /0 . . . . .	87
A.3.2 /constant . . . . .	89
A.3.3 /system . . . . .	90
A.4 Case D . . . . .	94
A.4.1 /0 . . . . .	94
A.4.2 /constant . . . . .	99
A.4.3 /system . . . . .	100

A.5	Case E	105
A.5.1	/0	105
A.5.2	/constant	108
A.5.3	/system	108
A.6	Case F	113
A.6.1	/0	113
A.6.2	/constant	115
A.6.3	/system	116
A.7	Case G	120
A.7.1	/0	120
A.7.2	/constant	124
A.7.3	/system	126
<b>B</b>	<b>Gmsh Code</b>	<b>132</b>
B.1	Duct	132
B.2	Airway-1	133
B.3	Airway-2	134
<b>Vita</b>		<b>135</b>



## List of Figures

1.1	Bronchi, Bronchial Tree, and Lungs, by Arcadian, 2006. Public Domain. [2]	4
1.2	Alveolar Sac Model with 13 alveoli used by Harding and Robinson [16]. Duct diameter, $DD$ , is 0.23 mm. Sac length is 0.816 mm. . . . .	8
3.1	OpenFOAM case file structure for each solver type used in this thesis. . . .	32
3.2	Structured meshes generated in Gmsh by transfinite interpolation for the current study. . . . .	41
3.3	Airway Model Variants . . . . .	42
4.1	Velocity profiles for 2D pipe flow. (a) The steady state solution using <code>simpleFoam</code> converged in 45 iterations. (b) Fully developed flow after 0.1 seconds using <code>icoFoam</code> . (c) - (f) Directed flow profiles for the 2D duct with a sinusoidal periodic boundary conditions at the inlet using <code>icoFoam</code> ; frequency $f = 2$ Hz, amplitude $A = 0.069542$ m/s. . . . .	46
4.2	Steady state solution for the idealized terminal alveolated airway using <code>simpleFoam</code> . The steady state solution converged in 72 iterations. . . . .	48
4.3	Transient laminar solutions on the idealized terminal alveolated Airway- 1 Model using <code>icoFoam</code> . The inlet velocity is uniform 0.0575 m/s in the $x$ -direction. . . . .	49

- 4.4 Transient laminar solutions with periodic boundary conditions defined on the inlet for the idealized terminal alveolated Airway-1 Model using `icoFoam`. The inlet velocity vector was sinusoidal: (0.0575 0 0) m/s with frequency  $f = 100$  Hz. . . . . 51
- 4.5 Transient laminar solutions on the idealized terminal alveolated airway with moving boundary using `pimpleDyMFoam`. The walls of the alveolar sac stretch outward with sinusoidal cell displacement equivalent to a 15.6%; frequency  $f = 10$  Hz, amplitude  $A = 2.34 \times 10^{-5}$  mm. The inlet and outlet were defined on the same edge. Neither inlet nor outlet was provided an initial condition for flow; flow was driven by wall expansion. . . . . 53
- 5.1 Recirculation zones for airway model in the duct center. The vector fields of the transient solution at  $t = 0.01$ s (b) and the steady state solution (c) are highly similar. . . . . 56
- 5.2 Flow patterns in the alveolus and traveling wavefronts for the steady-state (a) and transient (b)-(f) solutions. The steady-state wavefront of "brack-ish" fluid can be seen in 5.2(a) at the bottom and top left of the sac boundary. The wavefront is seen only in the bottom of the duct in (c) and (d). It approaches the alveolus boundary at each additional time step. The wavefront is then visible in the upper portion of (e). Chaotic recirculation is observed for the transient solutions (c)-(f). A saddle point is observed in the center of the sac in (e). . . . . 57
- 5.3 A detailed view of the fluid dynamics at final time  $t = 0.025$ s in the expanding sac airway model. . . . . 59
- 5.4 Sample error messages, which are frequently uninformative. . . . . 62

## List of Tables

3.1	Summary of simulations performed in this thesis. . . . .	34
3.2	Geometry specifications for the Duct and Airway Models. Model dimensions are rounded to simplify mesh construction from Harding and Robinson's Alveolar Sac Model [16]. . . . .	40
3.3	Mesh details for Duct and Airway Models. . . . .	41

# Abstract

## COMPUTATIONAL FLUID DYNAMICS IN A TERMINAL ALVEOLATED BRONCHIOLE DUCT WITH EXPANDING WALLS: PROOF-OF-CONCEPT IN OPENFOAM

by Jeremy M. Myers, M.S.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2017.

Director: Dr. Rebecca Segal, Associate Professor  
Department of Mathematics and Applied Mathematics

Mathematical Biology has found recent success applying Computational Fluid Dynamics (CFD) to model airflow in the human lung. Detailed modeling of flow patterns in the alveoli, where the oxygen-carbon dioxide gas exchange occurs, has provided data that is useful in treating illnesses and designing drug-delivery systems. Unfortunately, many CFD software packages have high licensing fees that are out of reach for independent researchers. This thesis uses three open-source software packages, Gmsh, OpenFOAM, and ParaView, to design a mesh, create a simulation, and visualize the results of an idealized terminal alveolar sac model. This model successfully demonstrates that OpenFOAM can be used to model airflow in the acinar region of the lung under biologically relevant conditions.

# Chapter 1

## Introduction and Overview

There is much in our physical world where an elegant mathematical description remains to be found. Frequently, these arenas are complex systems with high nonlinearity and many degrees of freedom. We are required to create models and develop techniques that allow us to approximate the world around us. Although these models may be simplifications: idealized and generalized, both physically and conceptually, they can be very accurate on a human scale. Techniques are built around these simulation frameworks with specific instructions so that we can have an answer within an efficient amount of time; it does no service to receive a disorganized solution in one hundred years, let alone one billion.

There is one area in particular where an incomplete mathematical theory compounded by urgent human need drives modeling by approximation and algorithm: fluid dynamics. Fluid dynamics governs complex and chaotic systems and since the mathematics that underlies fluid dynamics—the Navier-Stokes equations—remains incomplete, mathematicians and other researchers work actively to develop the methods by which we analyze these systems. For example, one method may be a scaled mock-up made of plastic, submerged in water, with the fluid flow analyzed by observing a dye and taking photographs—e.g. particle image velocimetry (PIV). Computational Fluid Dynamics

(CFD) is the set of methods where a computer model is created, subdivided into a finite collection of elements or cells, and a routine of computer operations solves the governing equations for each cell and for each time point. The set of solutions at each point in time and space (collectively, "the solution") provides a wealth of data to be analyzed: the magnitudes of pressure and momentum, flux, divergence, and more at each point. We can use this data to visualize our results, as well as to determine the quality and usefulness of the methods themselves, typically by calculating the error, among other measures.

The scientific discipline that lies at the intersection of CFD and mathematical biology has rich and interesting applications in human physiology. Beginning in the 1960s, researchers, in medicine and mathematics alike, have studied airflow in the human lung. What this thesis demonstrates is that a free and open-source CFD software package can be used with other free and open-source geometry-meshing and post-processing software to build and test a human lung model computationally with accurate results.

Mathematicians that might contribute greatly to lung physiology research may be otherwise hindered by the high monetary cost of commercial CFD licenses. Since lung physiology is small subfield in mathematical biology, demonstrating that a functional, scaled model can be created and tested with free and open-source software is essential to opening up research to a broader class of new researchers. Demonstrating that modeling of this kind can be performed without expensive commercial software is important because it may lead to a surge in research into this area.

Before any discussion of the modeling in this thesis, I provide a literature review as a window into the current state of mathematical models of the human lung, with an emphasis toward CFD. I used dimensions from the work of Harding and Robinson [16] to construct the models detailed in Chapter 3, with results of my simulations in Chapter 4.

## 1.1 Research Summary

In this thesis, I first used Gmsh [10] to create a geometry and mesh of an idealized terminal alveolar sac with dimensions taken from the literature. An overview of lung physiology follows below in Section 1.2.1. Two geometric models served 1) to test the capabilities of the software with a simple two-dimensional pipe featuring varying boundary conditions and 2) extend the tests to an idealized lung geometry. I used OpenFOAM [14] to develop seven cases featuring varied boundary conditions to model the physical breathing process computationally. Finally, flow patterns in the geometries were analyzed visually using ParaView [1]. The performance of OpenFOAM was assessed using built-in functions.

## 1.2 Literature Review

Modeling of the human lung began in 1963 with the Type A model of Ewald R. Weibel, MD, a symmetric tree geometry with 23 generations of bronchiole ducts [30]. Modeling has since pursued two different paths: computational models and in-vitro experiments. In this chapter I will give an overview of human lung physiology with attention paid to the pulmonary acinus. Then I will outline past efforts which informed the modeling decisions taken in this thesis. I will describe various model geometries of the pulmonary acinus, the region where lung branching terminates and gas enters and exits the blood. Lastly, I will discuss experimental techniques that have developed our insight into air-flow in the lungs.

### 1.2.1 Lung Physiology

This overview of human lung physiology follows from Cotes's *Lung Function: Physiology, Measurement, and Application in Medicine* [6]. The respiratory system begins with

## Bronchi, Bronchial Tree, and Lungs

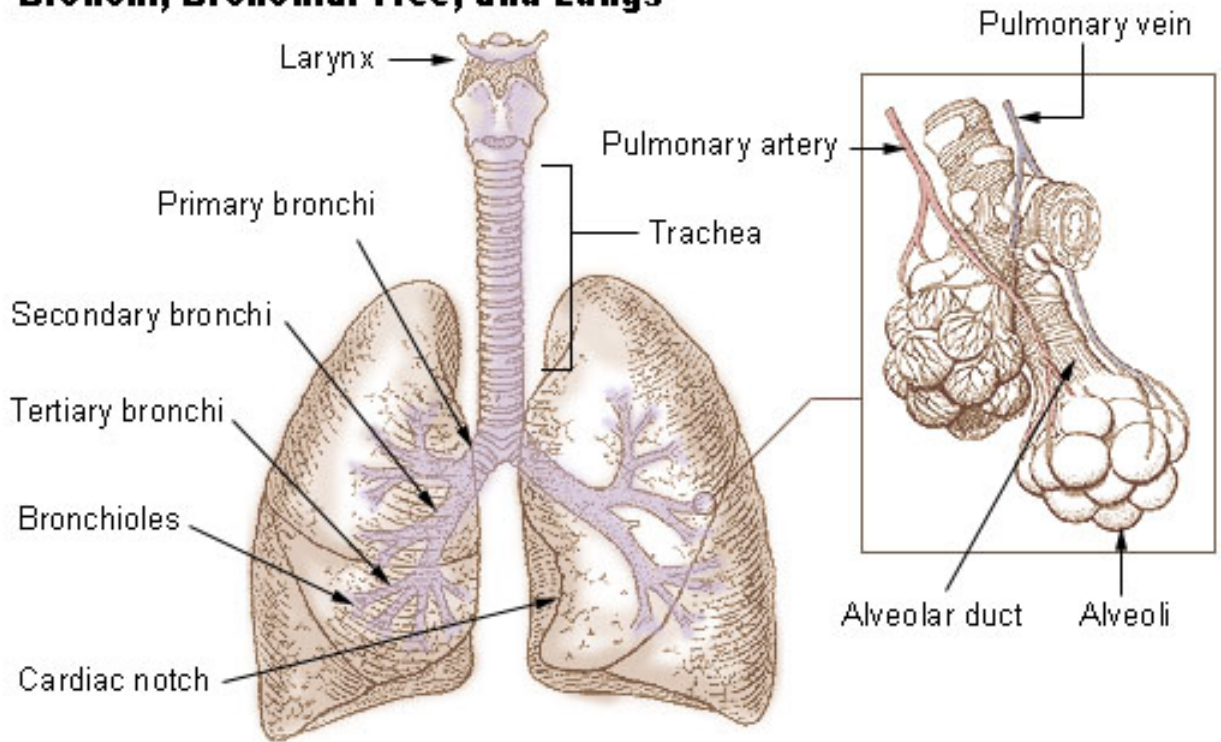


Figure 1.1: Bronchi, Bronchial Tree, and Lungs, by Arcadian, 2006. Public Domain. [2]

the upper airways: the nose, the nasopharynx, and the oropharynx. The goal of these structures is to condition inhaled air. This includes warming the air to near body temperature before air reaches the trachea and filtering out contaminants. Inspired air travels from the upper airway down to through the trachea and into the lungs. The trachea divides into the primary right and left bronchi, a pattern referred to as *branching*. Each bronchus subdivides further for a total of 23 generations in a so-called respiratory tree. See Figure 1.1.

The upper lung (generations 0-16) serve to transport air; the lower lung (generations 17-23) is where the oxygen-carbon dioxide gas exchange takes place. The lower lung is comprised of respiratory bronchioles, alveolated bronchiole ducts, and alveoli. The alveolated bronchiole ducts are covered by 10-30 roughly hexagonal alveoli and terminal alveolar ducts have a single spherical alveolus on the end. On average, a single alveolus is 0.25 mm in diameter and there are  $(200 - 600) \times 10^6$  alveoli in the average human lung.



Respiration is controlled by the respiratory muscles, principally the diaphragm. As the diaphragm contracts, the rib cage expands. At the same time, the intercostal muscles pull the rib cage up, allowing the rib cage to further expand. The expansion of the rib cage causes a pressure drop which draws air into the lungs. Air travels down through the trachea and bronchi to the respiratory bronchioles, alveolated bronchiole ducts, and alveoli of the lower lung. The alveoli are naturally compliant and the alveoli expand as they are filled with air. During inhalation, oxygen particles cross the alveoli walls to enter the bloodstream. During exhalation, this process is reversed. Carbon dioxide particles cross the alveoli walls from the bloodstream and exit the alveolar sacs due to contraction. This contraction of the alveolar sacs is caused by expansion of the diaphragm, which contracts the rib cage, drawing air out of the lungs.

### 1.2.2 Geometric Lung Models

Accurately modeling fluid flow in the terminal bronchiole ducts and alveoli has great implications for human health. Modeling may help researchers understand how lung diseases develop and progress, like emphysema and chronic obstructive pulmonary disorder (COPD). Furthermore, researchers may develop more effective drug delivery methods by analyzing particle deposition patterns in the alveoli based on better models.

Geometric models of alveolated ducts fall into two general types: (1) bronchiole ducts with a single terminal alveolar sac and (2) bronchiole ducts covered with multiple alveoli. Geometry plays an important role for flow patterns inside the alveolus. A circular alveolus was found to be preferred in early works [29]. Several three-dimensional geometries types have been explored more recently: truncated octahedral [18], dodecahedral [9], 19 face polyhedron [23], and spherical [4, 16].

### 1.2.3 In-Vitro Modeling

Ma et. al [22] validate CFD models with in-vitro experiments in three-dimensional bifurcated and alveolated airways. Their predictions from CFD match the measurements obtained from particle image velocimetry and particle tracking velocimetry.

Chhabra and Prasad [4] address the fluid mechanics of the acinar region consisting of alveolated airways (previous, upper (non-alveolated) bifurcating airways) using measurements of flow and dispersion in in-vitro models. Here, they investigate an idealized geometry of a single alveolus mounted on a respiratory bronchiole. Their study assumes that the bronchioles are unlikely to experience expansion and are kept rigid relative to expansion of the alveolus, in contrast to past studies. Their experiment uses an in-vitro model of a single alveolus, expanding and contracting in phase with oscillatory flow through the bronchiole with five distinct cases of increasing complexity. The in-vitro model was a cylindrical tube with a single hole on the top; a thin layer of latex is stretch over this hole. The whole tube was submerged in fluid inside a sealed box. A syringe pump oscillated flow into and out of the bronchiole tube while a separate syringe pump drew fluid from the box to create a negative pressure differential. In this study, the syringes oscillated in phase. Six cases were considered: three combinations of bronchiole flow (no flow, steady unidirectional flow, oscillating flow); and, two options of wall condition (non-deforming and oscillating). The no flow, non-deforming case is considered trivial; data was collected for the remaining cases. To measure velocity fields, particle image velocimetry (PIV) was employed. This paper focuses on dispersion and deposition of massless particles within the alveolus. The main result is for oscillating bronchiole flow over oscillating alveolus: primary and secondary vortices appear, as well as a shear layer at the alveolus mouth. The authors discuss recirculation, convective mixing, and particle deposition for each of the five nontrivial cases. A final observation is that particle transport to the alveolar wall is possible only when the alveolar wall expands and contracts.

## 1.2.4 Computational Modeling

Harding and Robinson [16] simulated airflow in an expanding terminal alveolar sac. The common theme of this work is to find convective mixing and recirculation patterns in the terminal pulmonary region by matching simulation parameters to physical measurements; model dimensions were taken from human lung casts and breathing cycles measured from a human volunteer. The geometry, referred to as the Alveolar Sac Model, is representative of those found in generations 19 and below: 12 truncated spherical alveolar sacs spaced evenly but asymmetrically along a duct with a single terminal alveolus added to the end. To solve the Navier-Stokes equations, the authors used a PISO solver\* since the flow is driven by a pressure differential due to the expanding walls (duct kept stationary). For each time step, a user-defined function iteratively expanded the walls to achieve desired volumes; this time step was optimized to agree with the breathing curve of a human volunteer measured *in vivo*. The authors chose to expand their terminal alveolar sac by 15.6% based on a 500-ml tidal volume (TV) and a 3200-ml functional residual capacity (FRC). The expansion was calculated by measuring flow rate, integrating to obtain the change in volume with respect to time, and then scaled. This expansion rate is used in case G in Section 3.2.2. The importance of flow rate ratio in determining recirculation patterns is highlighted.

Muller et. al [23] employ a novel CFD approach to quantify convective mixing. An irreversible flow – a fluid particle entering an alveolus during inhalation does not follow the same path during exhalation – impacts how long a particle remains in the alveolus, which in turn impacts particle deposition. An irreversible and chaotic flow is seen as an efficient feature of convective mixing. This novel technique employs an Eulerian scalar field to track this phenomenon numerically. An Eulerian approach marks an initial volume of fluid numerically, like with a dye. The marked particles are then tracked through the flow. The model geometry is a cylindrical bronchiole duct with a single

---

\*Pressure-Implicit Splitting of Operators. This algorithm is detailed in Section 2.4.2

polyhedral alveolus. Both the bronchiole and alveolus walls are deformed uniformly with the amplitude of the deformation calculated from breathing parameters. The flow rate is assumed to be sinusoidal and in-phase with the wall deformations. The ratio of flow rate in the alveolus to flow rate in the duct has been considered as characterizing alveolar flow, and great attention is paid to quantifying the effects of this ratio. Experiments are run for both single and multiple breath analyses. An advantage cited is that the Eulerian approach minimizes numerical errors.

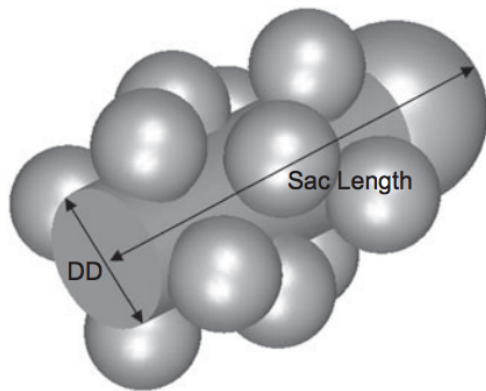


Figure 1.2: Alveolar Sac Model with 13 alveoli used by Harding and Robinson [16]. Duct diameter,  $DD$ , is 0.23 mm. Sac length is 0.816 mm.

Li and Kleinstreuer simulate transient laminar two-dimensional airflow using an in-house validated lattice-Boltzmann method. A CFD method developed to solve the lattice-Boltzmann equation, a lattice-Boltzmann method is a formulation of fluid flow problems that is an alternative to the Navier-Stokes equations which are the classical governing equations in fluid mechanics [21]. The authors in their study varied two different cases: (1) alveolus shape with a moving boundary; and,

(2) asymmetric alveolated duct flow on straight and branching ducts. Geometric asymmetry greatly impacts flow patterns in the alveolar region; in particular, asymmetric alveolus pairing and asymmetric alveolated branching greatly impacts air flow patterns.

### 1.3 Overview

In Chapter 2 I summarize the mathematics of the field. I provide the mathematical foundation for approximating solutions to the Navier-Stokes equations via the finite volume

method. This discussion starts by constructing a weak form of the boundary value problem for Poisson's equation. The argument extends to the Navier-Stokes equations. Then, I explain the theory and implementation of the Galerkin method and I show it is the best approximation to the solution. This leads to a generalization called the finite element method (FEM). When blended with several other concepts, FEM results in the more complex, three-dimensional analogue, the finite volume method (FVM). I conclude Chapter 3 with a discussion of the SIMPLE and PISO algorithms, the workhorses of FVM, for solving the Navier-Stokes equations.

In Chapter 4 I give the details of the methods used in this work. First, I discuss the two model geometries that I constructed for this work. Second, I detail the different numerical experiments that I conducted. I explain how I created the meshes using Gmsh. Next, I give an overview of OpenFOAM for scientific computation. The majority of the chapter is dedicated to describing the set up for each case modeled. I conclude Chapter 4 by briefly discussing ParaView, the software used for visualization and post-processing.

In Chapter 5 I discuss the results of the modeling. Much of the discussion focuses on the validity of the solutions calculated by OpenFOAM and assessment of the accuracy.

Finally, in Chapter 6 I compare results for one model across cases. Additionally, I discuss challenges with using OpenFOAM and where future work is headed.

## Chapter 2

# The Mathematics of Fluid Mechanics and Numerical Analysis

The equations that govern fluid flow are commonly represented by the Navier-Stokes equations, which are discussed in the following section. To date, existence and smoothness of solutions to the Navier-Stokes equations has not been proven. Purely mathematical research is very active—the Clay Mathematical Institute offers a \$1,000,000 prize for a proof or disproof of the existence and smoothness of solutions [8]. Despite this, the Navier-Stokes equations are useful for describing a variety of phenomena in science and engineering and mathematicians have developed advanced methods to approximate solutions to the Navier-Stokes equations based on powerful mathematical concepts.

In this chapter, I derive a current method used widely by the CFD community for approximating solutions to the Navier-Stokes equations: the Finite Volume Method (FVM). This discussion follows Gockenbach's *Understanding and Implementing the Finite Element Method* [11].

The Navier-Stokes equations are a special case of Poisson's equation. Converting Poisson's equation to a weak form permits a unique solution by bounding the domain. The Galerkin Method can be thought of as the computation of an approximation to the

true weak form solution to this boundary value problem; the theory guarantees that this is in fact the best approximation. By extension, the Galerkin Method produces the best approximation to the true strong form solution of Poisson's equation. The Galerkin Method is the conceptual basis for the Finite Element Method (FEM), which uses a collection of piecewise polynomials to subdivide the bounded domain and approximate the integrals and gradients in each subdomain. The above process is demonstrated in Section 2.2. The Finite Volume Method, described in Section 2.3, is an extension of FEM. The Finite Volume Method approximates the flux into and out of each discretization of the domain. In Section 2.4, I outline two current numerical algorithms that use FEM and FVM to approximate solutions to the Navier-Stokes equations in two and three dimensions. This chapter represents the theoretical framework in which OpenFOAM operates. OpenFOAM solvers are written to solve many different CFD cases and OpenFOAM employs libraries of secondary solvers to calculate each piece of the CFD puzzle.

To start the discussion, let's take a quick look at the basics of fluid flow as governed by the Navier-Stokes equations.

## 2.1 Navier-Stokes Equations

The Navier-Stokes equations are the partial differential equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nu \nabla^2 \mathbf{u} = -\nabla w \quad (2.1)$$

$$\frac{\partial p}{\partial t} + \nabla \cdot (p \mathbf{u}) = 0 \quad (2.2)$$

where

$\mathbf{u}$  is flow velocity,

$\nu$  is kinematic viscosity,

$p$  is pressure,

$\rho_0$  is uniform density, and

$$w = \frac{p}{\rho_0}.$$

Equation 2.1 is commonly known as the momentum equation. The term  $(\mathbf{u} \cdot \nabla)\mathbf{u}$  is the convective or inertial term, which describes the flow of the fluid as a whole; the  $\nu\nabla^2\mathbf{u}$  term is the diffusive term, the random motion of the fluid due to spreading [15]. The kinematic viscosity  $\nu$  is the ratio of dynamic viscosity  $\mu$  to (uniform) density  $\rho_0$ . Equation 2.2, commonly known as the continuity equation, determines compressibility of a fluid.\* For a complete derivation of the Navier-Stokes equations, see Chorin and Marsden's *A Mathematical Introduction to Fluid Mechanics* [5].

When solving a partial differential equation analytically or numerically, it is necessary to specify boundary conditions. This ensures that a unique solution exists and that it depends continuously on the initial and boundary conditions. A common boundary condition is  $\mathbf{u} = 0$  on the solid boundary at rest. This is referred to as a "no-slip" boundary condition and it requires that the velocity of the flow field tangent to the wall is zero.

### 2.1.1 Reynolds Number

In the field of fluid mechanics, there are several dimensionless values that allow us to compare different fluid flow systems. One important characteristic value is the dimensionless Reynolds number.

---

\*For incompressible flow,  $\frac{\partial p}{\partial t} = 0$ .



For the geometry of interest, define  $L$  to be the characteristic length (e.g. maximum radius or diameter),  $U$  to be the characteristic–maximum–velocity of the system. The Reynolds number is

$$\text{Reynolds number} \equiv \text{Re} = \frac{LU}{\nu}$$

the ratio of inertial and viscous forces. The Reynolds number is useful for making a distinction between laminar ( $\text{Re} < 1200$ ) and turbulent ( $\text{Re} > 2000$ ) flow. Laminar flow is characterized by smooth flow and corresponds to low Reynolds number. Turbulent flow produces chaotic mixing and unstable flow patterns.

With the basics of fluid mechanics in place, I turn to the theory of approximating solutions to the Navier-Stokes equations.

## 2.2 The Finite Element Method

As mentioned above, FEM uses a weak form of Poisson’s equation to find a best approximation to the true solution of the PDE, which can be extended to solving the Navier-Stokes equations.

### 2.2.1 The Weak form of a Boundary Value Problem

Consider Poisson’s equation with the solution defined on the boundary:

$$-\nabla \cdot (\kappa \nabla u) = f \quad \text{in } \Omega \quad (2.3)$$

$$u = 0 \quad \text{on } \partial\Omega \quad (2.4)$$

where  $\Omega$  is a bounded domain in  $\mathbb{R}^2$  and  $\partial\Omega$  is the boundary of  $\Omega$ . If  $f$  is continuous, then  $u$  must have two continuous derivatives defined in the closure of  $\Omega$ :  $u \in C_D^2(\bar{\Omega})$ , where  $D$  denotes a Dirichlet problem. Further, if  $u$  is a solution to Equation 2.3 in the

bounded domain  $\Omega$ , then for any function  $v$  defined on  $\Omega$ ,

$$-\nabla \cdot (\kappa \nabla u)v = f v \quad \text{in } \Omega.$$

This implies that the integral equation also holds:

$$\int_{\Omega} -\nabla \cdot (\kappa \nabla u)v = \int_{\Omega} f v.$$

The function  $v$  is known as a *test function* and defines a weight for determining the average of the PDE over  $\Omega$ .

Using Green's identity and the divergence theorem,

$$\begin{aligned} \int_{\Omega} -\nabla \cdot (\kappa \nabla u)v &= \int_{\Omega} \kappa \nabla u \cdot \nabla v - \int_{\partial\Omega} \kappa v \frac{\partial u}{\partial n} \\ &= \int_{\Omega} \kappa \nabla u \cdot \nabla v. \end{aligned}$$

The term  $\int_{\partial\Omega} \kappa v \frac{\partial u}{\partial n}$  disappears since  $u = 0$  on the boundary  $\partial\Omega$ .

The weak form of Equation 2.3 is

$$\int_{\Omega} \kappa \nabla u \cdot \nabla v = \int_{\Omega} f v, \quad (2.5)$$

which holds for some  $u \in C_D^2(\bar{\Omega})$  if and only if  $u$  satisfies Equations 2.3–2.4.

The presence of the Laplacian in Equation 2.3 implies that the solution  $u$  should have two partial derivatives—i.e.  $u$  should be twice differentiable.\* However, in the weak formulation only one partial derivative is necessary.

Toward a definition of a *weak partial derivative* for a function  $u$ , define the *support of  $u$*

---

\*The term  $\nabla^2 u$  requires the second derivative of  $u$ , which increases the number of initial conditions from one to two.

as the closure of the set on which  $u$  is nonzero:

$$\text{supp}(u) = \overline{\{(x, y) \in \mathbb{R}^2 : u(x, y) \neq 0\}}.$$

If  $u$  is defined on  $\Omega$  and the support of  $u$  is a compact-closed and bounded-subset, then  $u$  is *compactly supported* in  $\Omega$ .

Suppose that  $u$  is a real-valued function defined on  $\Omega \subset \mathbb{R}^2$  and that  $u$  is integrable over every compact subset of  $\Omega$ . Let  $C_0^\infty(\Omega)$  denote the space which is the set of all functions that are infinitely differentiable on  $\Omega$  and compactly supported in  $\Omega$ . If there exists another locally integrable function  $g$ , in addition to  $u$ , defined on  $\Omega$  such that

$$\int_{\Omega} g v = - \int_{\Omega} u \frac{\partial v}{\partial x} \quad \text{for all } v \in C_0^\infty(\Omega)$$

then  $u$  is *weakly differentiable* with respect to  $x$  and  $g$  is defined as the *weak partial derivative* with respect to  $x$ .

The motivation for defining the weak partial derivatives of  $u$  may not be immediately clear. It is important to say that it is done with the goal of "relaxing" Equation 2.3 so that the assumptions are as weak as possible. In turn, the "relaxed" PDE is more general and encompasses many more cases, one of which is the Navier-Stokes equations for incompressible flow.

What is important theoretically is that the conditions necessary for a solution to the weak form determine a Sobolev space. Recall that the Euclidean  $L^2$  space is a vector space defined as

$$L^2(\Omega) = \{v : \Omega \rightarrow \mathbb{R} \mid \int_{\Omega} v^2 < \infty\}.$$

If both  $v$  and its derivative  $v'$  are vectors in  $L^2$ , then

$$H^1(\Omega) = \left\{ v \in L^2(\Omega) : \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y} \in L^2(\Omega) \right\}$$

is a Sobolev space.\* For the Dirichlet problem, the constraints on Equations 2.3–2.4 result in the space

$$H_0^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}.$$

Therefore, the weak form boundary value problem of Equations 2.3–2.4 is given as

$$\text{Find } u \in H_0^1(\Omega) \text{ such that } \int_{\Omega} \kappa \nabla u \cdot \nabla v = \int_{\Omega} f v \text{ for all } v \in H_0^1(\Omega). \quad (2.6)$$

### Existence and Uniqueness of Solutions to the Weak Boundary Value Problem

Here I introduce some definitions necessary for proving that unique solutions to the weak boundary problem 2.6 exist.

**Definition 1** If  $V$  is a vector space with a norm, then a linear functional  $\ell$  on  $V$  is a real-valued function that is linear; i.e.  $\ell(\alpha u + \beta v) = \alpha \ell(u) + \beta \ell(v)$ , for all  $u, v \in V$  and  $\alpha, \beta \in \mathbb{R}$ .

**Definition 2** The functional  $\ell$  is continuous at  $u \in V$  if

$$\lim_{v \rightarrow u} \ell(v) = \ell(u) \quad \text{or} \quad \|\ell(u) - \ell(v)\| \rightarrow 0 \quad \text{as} \quad \|u - v\| \rightarrow 0.$$

In other words, it is necessary that the operation of the functional  $\ell$  converges to the solution as the solution itself converges.

Since the linear functional  $\ell$  is defined on the vector space  $V$ , it is natural to define a norm for  $\ell$ :

**Definition 3** The norm of the linear functional  $\ell$  is defined as

$$\|\ell\| \equiv \text{least upper bound } \frac{\|\ell(u)\|}{\|u\|}.$$

It is now possible to define the dual space of  $V$ :

---

\*Each function  $v$  is mapped to a real number using an integral calculation which must be finite.

**Definition 4** The set of all continuous linear functionals on  $V$  forms a normed vector space called the dual space  $V^*$ , and

$$\|\ell\|_{V^*} = \sup \{|\ell(\mathbf{u})| : \mathbf{u} \in V, \|\mathbf{u}\| \leq 1\}.$$

The Riesz Representation Theorem provides the necessary conditions for existence and uniqueness of solutions to the weak boundary value problem.

**Theorem 1 (Riesz Representation Theorem)** Suppose  $V$  is a complete normed vector space. The dual space  $V^*$  can be identified with  $V$  as follows:\*

1. For each vector  $\mathbf{u} \in V$ , the linear function  $\ell$  defined by  $\ell(\mathbf{v}) = (\mathbf{u}, \mathbf{v})$  belongs to  $V^*$  and  $\|\ell\|_{V^*} = \|\mathbf{u}\|_V$ ;
2. For each  $\ell \in V^*$ , there exists a unique  $\mathbf{u} \in V$  such that  $\|\ell\|_{V^*} = \|\mathbf{u}\|_V$  and  $\ell(\mathbf{v}) = (\mathbf{u}, \mathbf{v})$  for all  $\mathbf{v} \in V$ .

Suppose the term  $\int_{\Omega} \kappa \nabla \mathbf{u} \cdot \nabla \mathbf{v}$  on the left hand side of Equation 2.5 is treated as an operator.

Define

$$a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \kappa \nabla \mathbf{u} \cdot \nabla \mathbf{v}.$$

Assume the operator  $a(\cdot, \cdot)$  satisfies the following:

1.  $a(\mathbf{u}, \mathbf{v}) = a(\mathbf{v}, \mathbf{u})$ . [a is symmetric]
2.  $a(\alpha \mathbf{u}, \beta \mathbf{v}) = \alpha a(\mathbf{u}) + \beta a(\mathbf{v})$ . [a is linear]
3.  $a(\mathbf{u}, \mathbf{u}) \geq 0$  ( $a(\mathbf{u}, \mathbf{u}) = 0$  if and only if  $\mathbf{u} = 0$ ). [a is positive definite]
4. There exists  $\alpha > 0$  such that  $a(\mathbf{u}, \mathbf{u}) \geq \alpha \|\mathbf{u}\|^2$  for all  $\mathbf{u} \in V$ . [a is elliptic]

---

\*Part 1 of Theorem 1 provides a connection between the operations and functions on which the operations are being used. Part 2 proposes that an unique solution exists.

5. There exists  $\beta > 0$  such that  $a(u, v) \leq \beta \|u\| \|v\|$  for all  $u, v \in V$ . [a is bounded]

If  $a(\cdot, \cdot)$  satisfies (1 - 5) above, then  $a(u, v) = \ell(v)$  satisfies the Riesz Representation Theorem for all  $v \in V$  and the solution  $u$  exists and is unique. For a complete proof, see [11].

## 2.2.2 The Galerkin Method

The Galerkin Method is a way to compute the best approximation to the true solution  $u$  from a given finite-dimensional space. Let's start organizing this method with the following theorem:

**Theorem 2 (Galerkin Theorem)** *Suppose  $V$  is an inner product space with  $u \in V$  and  $W$  is a finite-dimensional subspace of  $V$ .*

*Then*

1. *There exists an unique  $w \in W$  such that  $\|u - w\| < \|u - z\|$  for all  $z \in W$  and  $z \neq w$ ;*
2. *A vector  $w$  is the best approximation to  $u$  from the subspace  $W$  if and only if  $(u - w, z) = 0$  for all  $z \in W$ .*

The Galerkin Theorem says that there is a projection of  $u$  onto  $W$ , that is closest to  $u$  in  $W$ . Part 2 of Theorem 2 says that  $w$  is the best approximation to  $u$  if and only if  $u - w$  is orthogonal to  $z$ : that is, the error of the approximate solution to the true solution is zero.

It is a standard result in linear algebra that if  $W$  is finite dimensional, then it has a basis  $\{w_1, \dots, w_n\}$  and  $u \in W$  can be represented by  $w = \sum_{j=1}^n \alpha_j w_j$ . If we know the basis  $w$ , we can find the  $\alpha_j$ 's and construct the projection  $w$  on  $u$ .

From Part 2 in Theorem 2 let  $z = w_i$  for each  $i$ .

For  $i = 1, \dots, n$ ,

$$0 = \left( \mathbf{u} - \sum_{j=1}^n \alpha_j \mathbf{w}_j, \mathbf{w}_i \right) \quad (\text{orthogonality})$$

$$0 = (\mathbf{u} - \mathbf{w}_i) - \sum_{j=1}^n \alpha_j (\mathbf{w}_j, \mathbf{w}_i) \quad (\text{linearity})$$

$$\text{So, } (\mathbf{u}, \mathbf{w}_i) = \sum_{j=1}^n (\mathbf{w}_j, \mathbf{w}_i) \alpha_j.$$

Let be the Gram matrix  $G_{ij} = (\mathbf{w}_i, \mathbf{w}_j)$  of inner products,  $b_i = (\mathbf{u}, \mathbf{w}_i)$ , and, if  $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_n\}$  is an orthonormal basis,  $\alpha_i = \frac{(\mathbf{u}, \mathbf{w}_i)}{(\mathbf{w}_i, \mathbf{w}_i)}$ . Then the normal equations which minimize  $\mathbf{u} - \mathbf{w}$  are the solutions of  $G\alpha = \mathbf{b}$ .

The above can be summarized in the following theorem:

**Theorem 3** *Let  $W$  be a finite dimensional subspace of  $V$  with a basis  $\{\mathbf{w}_1, \dots, \mathbf{w}_n\}$ . Then the best approximation of  $\mathbf{u} \in V$  is the solution of  $G\alpha = \mathbf{b}$ , where  $G_{ij} = (\mathbf{w}_i, \mathbf{w}_j)$  and  $b_i = (\mathbf{u}, \mathbf{w}_i)$ .*

Since we are solving for  $\mathbf{u}$ , the  $b_i$ 's are unknown. It is convenient to recast the problem  $G\alpha = \mathbf{b}$  using the energy inner product  $KU = F$ , where  $K = a(\mathbf{w}_j, \mathbf{w}_i)$  is the *stiffness matrix* and  $F = a(\mathbf{u}, \mathbf{w}_i) = \ell(\mathbf{w}_i)$  is the *load vector*. Hinting toward the finite element method, the goal is to find  $U$  and use the solution to find the  $b_i$ 's. With  $\mathbf{w} = \sum_{i=1}^n v_i \mathbf{w}_i$  as the solution and  $a(\mathbf{w}, v_i) = a(\mathbf{u}, \mathbf{w}_i) = \ell(\mathbf{w}_i)$ , the original problem 2.6 is now:

$$\text{Find } \mathbf{w} \in W \text{ such that } a(\mathbf{w}, \mathbf{v}) = \ell(\mathbf{v}) \quad \forall \mathbf{v} \in W.$$

To summarize,  $\mathbf{w} \in W$  is the best approximation of  $\mathbf{u}$ . If  $\mathbf{u}$  solves the PDE 2.6,\* then  $\mathbf{w} \in W \subset V$  implies that  $a(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{v})$  for all  $\mathbf{w} \in W$ . So, the projection  $\mathbf{w} \in W$  satisfies  $a(\mathbf{w}, \mathbf{v}) = \ell(\mathbf{v})$  for all  $\mathbf{v} \in W$ . In turn,

$$a(\mathbf{u}, \mathbf{v}) - a(\mathbf{w}, \mathbf{v}) = 0 \quad \forall \mathbf{v} \in W$$

$$a(\mathbf{u} - \mathbf{w}, \mathbf{v}) = 0 \quad \forall \mathbf{v} \in W.$$

\*That is,  $a(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{v})$  for all  $\mathbf{v} \in W$

Therefore,  $w = \text{proj}_W u$  is the best approximation of  $u$  under the inner product  $a(\cdot, \cdot)$  since the error of the approximation to the true solution is zero.

### 2.2.3 The Galerkin Finite Element Method

The *Galerkin Finite Element Method* is the Galerkin Method on a subspace,  $W$ , of piecewise polynomials, defined on subdomains of  $\Omega$ . The collection of subdomains is called a *mesh*. Triangular mesh elements do a good job of fitting a given space. When working with fluid flow however, the triangle faces may not match the direction of flow.

Let  $T$  be the interior of a triangular mesh element with its boundary and  $\tau_h$  be a triangulation—a division of the space into triangles—with max triangle diameter  $h$ . It is possible to define a function on each triangle: e.g.  $a_i + b_i x + c_i y$  is a piecewise linear function of degree 1. For a triangle, let  $z_i = (x_i, y_i)$  for  $i = 1, \dots, 3$  represent its corner coordinates in  $\mathbb{R}^2$ . On each  $T_i \in \tau_h$  it is possible to uniquely determine values for  $a_i, b_i, c_i$  using the function values  $z_i$ . If each element of  $\tau_h$  has  $N$  vertices, written  $N_V$ , then the space  $P_h^1$  is the set of continuous linear functions on  $\tau_h$  and is a finite dimensional vector space with dimension  $N_V$ . Furthermore, each function  $v \in P_h^1$  can be represented by a vector  $d = [a_i \quad b_i \quad c_i]^T \in \mathbb{R}^{N_V}$  consisting of the function values at each node.

The space  $P_h^1$  satisfies the conditions of the Riesz Representation Theorem, which implies one can find a unique solution.\*

Define a basis for  $P_h^1$ :

$$v = \sum_{i=1}^{N_V} d_i \Psi_i(x_j, y_j) = v(x_j, y_j) = d_j \quad \text{for} \quad \Psi_i(x_j, y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

where  $\Psi$  is a *Lagrange* or *nodal basis* that is "on" at nodes and "off" at other points. These nodal basis functions contribute information only locally, so the stiffness matrix  $K$  is

---

\*Part 2 of Theorem 1



sparse and computationally inexpensive to solve.

Recall that  $K = a(w_i, w_j)$ . To construct the sparse matrix, convert the shape function, which interpolates the nodes, into the contribution from each basis function:  $w_i = \sum d_i \Psi_i$ .

For example, let's look at triangles  $T_1, T_2$ . For node  $z_1 = (x_1, y_1)$ , the support of the basis function  $\Psi_1$  is  $T_1 \cup T_2$ .  $\Psi_1$  will contribute information only to triangles  $T_1, T_2$ . By the definition  $K = a(w_i, w_j)$  the inner products for  $i, j \in N_V$  will be mostly zeros, producing sparse  $K$ :

$$K_{ij} = \int_{\Omega} \kappa \nabla \Psi_i \cdot \nabla \Psi_j = \sum_{k=1}^t \int_{T_k} \kappa \nabla \Psi_i \cdot \nabla \Psi_j,$$

where  $T_1, \dots, T_t$  is  $\text{supp}(\Psi_i) \cup \text{supp}(\Psi_j)$  and  $\kappa$  is a scalar. Build  $K$  by iterating over the triangles. The load vector  $F$  is computed similarly:

$$F_i = \int_{\Omega} f \Psi_i = \sum_{k=1}^t \int_{T_k} f \Psi_i.$$

Gaussian quadrature is an approach to numerically approximate an integral given some sample data points—or, "abscissa"—where a smart choice of abscissa and weights provides the highest degree of precision possible [3]. It is natural that more sample points provide a more accurate approximation assuming the quadrature rule fits the problem. Gaussian quadrature rules are derived by requiring exact integration of a polynomial up to some degree  $n$ .

For the problem  $KU = F$ , it is necessary then to have a quadrature rule to approximate the integral over each  $T_k$  and the gradient of each  $\Psi_i$ . Since

$$\nabla \Psi_i = \begin{bmatrix} \frac{\partial \Psi_i}{\partial x} \\ \frac{\partial \Psi_i}{\partial y} \end{bmatrix} = \begin{bmatrix} b_i \\ c_i \end{bmatrix}$$

the gradient of each  $\Psi_i$  is constant on each  $T_k$  and

$$\int_{T_k} \kappa \nabla \Psi_i \cdot \nabla \Psi_j = (\nabla \Psi_i \cdot \nabla \Psi_j) \int_{T_k} \kappa dA.$$

Approximating the integral for each triangle,

$$\int_{T_k} \kappa dA \approx A_k \kappa(\bar{x}_k, \bar{y}_k)$$

where  $A_k$  is the area of  $T_k$  and  $(\bar{x}_k, \bar{y}_k)$  is the centroid of triangle  $T_k$ . The load vector is approximated similarly:

$$\int_{T_k} f \Psi_i \approx A_k f(\bar{x}_k, \bar{y}_k).$$

The stiffness matrix  $K$  and load vector  $F$  are constructed by iterating over the triangles and nodes, respectively, and accumulating the results in  $K$  and  $F$ .

## 2.2.4 Solving $KU = F$ with a Krylov method

The construction of  $K$  can be done in a smart way that makes  $K$  symmetric. This allows the solution of  $KU = F$  to be approximated efficiently using an iterative *Krylov method*. These methods calculate an initial residual  $r_0 = F - KU_0$ —a proxy for the error to the true solution  $U^*$  based on an initial guess  $U_0$ —and multiply it against successive powers of a matrix: the *Krylov sequence* is  $\{r_0, r_0K, K^2r_0, \dots, K^{k-1}r_0\}$  and the  $k$ th *Krylov subspace* is the span of this sequence. The residual is then minimized over the Krylov subspace, resulting in the approximate solution to  $F - KU^* = 0$ .

**Definition 5** A matrix  $A$  is said to be symmetric if  $A = A^T$  and positive-definite if  $x^T Ax > 0$  for all  $x \neq 0, x \in A$ .

Since  $K$  is constructed to be symmetric, it is only necessary that  $K$  be positive-definite for use with the Conjugate Gradient (CG) method. If  $K$  is not symmetric, then one should use the Generalized Minimal Residual (GMRES) method or a preconditioner on  $K$ .

**Definition 6** For a sequence  $\{p_n\}$  that converges to a number  $p$ , a quadrature rule converges quadratically if there exists positive constant  $\lambda$  such that

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^2} = \lambda.$$

The condition that  $K$  be positive-definite is ensured by the choice of a quadrature rule that converges quadratically for the integrals over  $T_K$ . For details about Krylov subspace methods and an in-depth analysis on CG, GMRES, and other methods and their extensions to nonlinear problems, see C. T. Kelley's *Iterative Methods for Linear and Nonlinear Equations* [20].

## 2.3 Finite Volume Method

The Finite Volume Method (FVM) is best thought of as a hybrid of finite difference and finite element methods.\*

Assuming conservation of mass, the finite volume problem is to solve

$$0 = \iint_{\Delta C} \nabla \cdot (\kappa \nabla u) \, dA \quad (2.7)$$

where  $\Delta C$  is the change over a cell  $C$ . A description of the method follows in two-dimensions but FVM naturally extends to three-space.

Let  $p$  be a point in the  $x - y$  plane. Define the space in the positive  $x$  direction as East, the space in the negative  $x$  direction as West, the space in the positive  $y$  direction as North, and the space in the negative  $y$  direction as South.

---

\*A finite difference method is a pointwise approximation of the derivative on a regular, orthogonal grid.

Then

$$\begin{aligned}
0 &= \iint_{\Delta C} \nabla \cdot (\kappa \nabla u) dA \\
&= \iint \frac{\partial}{\partial x} \kappa \frac{\partial u}{\partial x} dx dy + \iint \frac{\partial}{\partial y} \kappa \frac{\partial u}{\partial y} dx dy
\end{aligned} \tag{2.8}$$

The first term of 2.8 describes the flux in and out of the cell in the  $x$  (E-W) directions. The  $\int dy$  component equals  $A_E = A_W$ , the areas of the east and west faces, respectively. Similarly, the second term of 2.8 describes the flux in and out of the cell in the  $y$  (N-S) directions. The  $\int dx$  component equals  $A_N = A_S$ , the areas of the north and south faces, respectively. It is possible to write the terms in 2.8 as

$$\begin{aligned}
\iint \frac{\partial}{\partial x} \kappa \frac{\partial u}{\partial x} dx dy &= \kappa_E A_E \left( \frac{\partial u}{\partial x} \right)_E - \kappa_W A_W \left( \frac{\partial u}{\partial x} \right)_W \\
\iint \frac{\partial}{\partial y} \kappa \frac{\partial u}{\partial y} dx dy &= \kappa_N A_N \left( \frac{\partial u}{\partial y} \right)_N - \kappa_S A_S \left( \frac{\partial u}{\partial y} \right)_S
\end{aligned}$$

where the flux across each cell face is approximated by

$$\begin{aligned}
\left( \frac{\partial u}{\partial x} \right)_E &= \frac{u_E - u_P}{\Delta x_{EP}} & \left( \frac{\partial u}{\partial x} \right)_W &= \frac{u_P - u_W}{\Delta x_{PW}} \\
\left( \frac{\partial u}{\partial y} \right)_N &= \frac{u_N - u_P}{\Delta y_{PN}} & \left( \frac{\partial u}{\partial y} \right)_S &= \frac{u_P - u_S}{\Delta y_{SP}}
\end{aligned}$$

So 2.8 is equivalent to

$$0 = \kappa_E A_E \frac{u_E - u_P}{\Delta x_{EP}} - \kappa_W A_W \frac{u_P - u_W}{\Delta x_{WP}} + \kappa_N A_N \frac{u_N - u_P}{\Delta y_{NP}} - \kappa_S A_S \frac{u_P - u_S}{\Delta y_{SP}}$$

Grouping the terms,

$$\left( \frac{\kappa_E A_E}{\Delta x_{EP}} + \frac{\kappa_W A_W}{\Delta x_{WP}} + \frac{\kappa_N A_N}{\Delta y_{NP}} + \frac{\kappa_S A_S}{\Delta y_{SP}} \right) u_P = \frac{\kappa_E A_E}{\Delta x_{EP}} u_E + \frac{\kappa_W A_W}{\Delta x_{WP}} u_W + \frac{\kappa_N A_N}{\Delta y_{NP}} u_N + \frac{\kappa_S A_S}{\Delta y_{SP}} u_S$$

is equivalent to the linear system

$$\alpha_P u_P = \alpha_E u_E + \alpha_W u_W + \alpha_S u_S + \alpha_N u_N \quad \text{for each } p, \quad (2.9)$$

where  $\alpha_* = \frac{\kappa_* A_*}{\Delta x_P}$ . To solve the system, find the solution  $u_P$  at each node.

FVM converges quadratically at best. Approximating the flux across each cell face has limitations; it is not guaranteed that a regular or uniform discretization can be generated that would allow for higher order estimates of  $\frac{\partial u}{\partial x}$ ,  $\frac{\partial u}{\partial y}$ . Compared with the finite element method, FVM is relatively fast and flexible but does not allow much control over the accuracy of the scheme. It is flexible in the sense that it can approximate accurately highly irregular domains but does not handle discontinuous solutions very well. In general, it is recommended for flow simulations where the viscosity is not a dominant force. For three-dimensional problems, FEM may provide more accurate solutions but is computationally more expensive. On the other hand, FVM will work faster on 3D problems at the expense of a more accurate solution, though this is typically acceptable.

## 2.4 Numerical Algorithms for Solving the Navier-Stokes Equations

Recall that FVM uses flux equations to discretize a PDE and create finite differences to approximate the solution. Focusing on the Navier-Stokes equations, there are options for solving Equations 2.1-2.2, like pressure-based solvers for incompressible flow and density-based solvers for compressible flow. A general algorithm to solve Equations 2.1-2.2 works as follows: on each cell, (1) start with an initial guess for pressure, (2) solve for velocity, and then (3) correct for pressure; then, iterate on each cell at each time step until the approximate solution converges to the true solution within an acceptable tolerance.

Two numerical algorithms are widely in use in the CFD community today: the SIM-

PLE (Semi-Implicit Method for Pressure-Linked Equations) and PISO (Pressure-Implicit with Splitting of Operators) algorithms. The SIMPLE algorithm is employed for steady-state problems and the PISO algorithm is used for calculating transient flow problems.

### 2.4.1 SIMPLE

The SIMPLE algorithm, a heuristic, is quoted from its creator, Suhas Patankar [27]:

1. *Guess the Pressure field  $p^*$ .*
2. *Solve the (discretized) Momentum equation 2.1 to obtain  $u^*$ .*
3. *Solve for the Pressure correction  $p'$ .*
4. *Calculate Pressure  $p$  from  $p^*$  and  $p'$ .*
5. *Use  $u^*$  to update Momentum  $u$ .*
6. *Solve for other values that may impact the flow: e.g. Temperature, Concentration, Turbulence.*
7. *Set Pressure  $p = p^*$ , return to step 2, and repeat the whole procedure until a converged solution is obtained.*

Step 2 first requires that the internal velocity field be computed from the boundary conditions. Then the mass fluxes at each cell face are approximated and the solution  $u^*$  of Equation 2.9 is computed for each cell. Step 5 corrects the mass fluxes at each cell face from the pressure update. Momentum is updated with this information.

### 2.4.2 PISO

The PISO algorithm, developed by Raad Issa [19], follows the SIMPLE algorithm with added corrector steps:

1. Do SIMPLE algorithm steps 1-3.
2. Repeat steps 4-5 for a prescribed number of times.
3. Do SIMPLE algorithm steps 6-7.

The PISO algorithm is more effective for turbulent flow problems, in general.

# Chapter 3

## Methods

This chapter focuses on the methods used to 1) design and construct a computational mesh, 2) develop a case file for a CFD simulation, and 3) visualize the results of an idealized terminal alveolar bronchiole duct with expanding alveolar wall. Each of the above tasks was completed using freely-available open-source software: Gmsh for mesh generation, OpenFOAM for CFD simulation, and ParaView for post-processing/visualization.

Designing a work flow that incorporates disparate software packages is akin to assembling a 3D puzzle. A small change in one part of the pipeline may require an entire segment to be reworked. Therefore, several cases of increasing complexity were developed to test each mesh type and computational solver culminating in a coherent model that simulates airflow in a terminal alveolar bronchiole duct with biological accuracy.

In Section 3.1, I give a detailed description of OpenFOAM and its use as a general engine for solving a variety of computational problems, including fluid flow. Furthermore, I explain the development of the simulations that model steady-state, uniform, and oscillating flow in the Duct and Airway-1 Models and flow in an expanding Airway-2 Model. (See Section 3.3 below for details about the Duct, Airway-1, and Airway-2 Models). The simulations are summarized in Table 3.1.

In Section 3.3, I provide an overview of Gmsh and give details about the two idealized



mesh models designed for verification and simulation: a simple, 2D bronchiole "Duct Model" and a 2D bronchiole duct with a single terminal alveolus "Airway Model". Two variants of the Airway Model (Airway-1 and Airway-2) are described in Section 3.3.

In Section 3.4 I briefly discuss the post-processing tools of ParaView used in this thesis.

## 3.1 OpenFOAM for Computational Fluid Dynamics

### 3.1.1 Overview

OpenFOAM ("Open source Field Operation And Manipulation") is a free and open-source software toolbox written in C++ for the development of numerical solvers and pre- and post-processing of solutions to continuum mechanics problems, especially problems in computational fluid dynamics (CFD). OpenFOAM derives its enormous potential from the variety of complex solvers, the ability to create new solvers, and its generality. Many solvers and example cases are fluid flow problems in mechanical and aerospace engineering; examples include compressible and incompressible flow, Newtonian and non-Newtonian fluids, stress analysis, and heat transfer. Beyond these conventional applications, OpenFOAM has been developed and cases have been created for combustion, electromagnetics, and financial mathematics problems.

For this project, OpenFOAM version 4.1 was run using the Docker open-platform, version 17.03. Pre-compiled OpenFOAM distributions run in Docker containers have all of the libraries and source code required to run simulations and can therefore be run natively on the common Windows and Mac OSX operating systems.

The main modeling goal was to generate an idealized bronchiole duct with terminal alveolar sac that expands and contracts. Physiologically, the expansion and contraction causes a pressure change that forces the flow of air into and out of the alveoli. As a CFD modeling approach, this type of problem is generally referred to as fluid-surface

interaction (FSI). Two examples include the immersed boundary and the lattice Boltzmann methods. While FSI solvers for OpenFOAM have been developed by independent authors, they are nonstandard and are not distributed with version 4.1. As will be discussed below, this makes the modeling of biological and physiological fluid flow in OpenFOAM a challenge with limited results.

### 3.1.2 OpenFOAM as a General Computational Engine

OpenFOAM comes out of the box with many solvers designed for ordinary differential equations including Euler methods, Rosenbrock methods, Runge-Kutta methods, midpoint and trapezoid-rule methods, and adaptive solvers, although its strength comes from the many libraries written to solve partial differential equations. The PDE solvers in OpenFOAM allow a great deal of user control in approximating the solution for a given problem. The built-in solvers fall into several broad categories: time discretization schemes, field operator schemes (e.g. derivative, gradient, flux, etc.), interpolation schemes, and meshing schemes.

Time discretization of partial differential equations can be tricky. In all but the simplest cases (steady-state flow), one must be careful when choosing an explicit time discretization technique.

The Courant-Friedrichs-Lewy (CFL) condition is a necessary condition for the stability of any difference scheme that features advection [3]. Informally, the CFL condition requires that size of the time step,  $\Delta t$ , be chosen small enough relative to the size of the spatial discretization,  $\Delta x$ . This ensures that the material under transport progress no more than one computational cell per time step. This is measured with the dimensionless *Courant number*. In one dimension, the Courant number is

$$C = \left| \mathbf{u} \frac{\Delta t}{\Delta x} \right| \leq C_{\max},$$

where  $u$  is the magnitude of the velocity. In practice,  $C_{\max} = 1$ . OpenFOAM reports the Courant number by default at each time step, valuable information that lets the user witness the stability of the scheme to allow for adjustments in real-time.

General solvers for field schemes also vary. While there is only one solver to calculate the Laplacian and another to calculate divergence, many options for approximating the gradient are built into OpenFOAM. In particular, there are several flavors each of least-squares methods, Gauss-Newton methods, and a class of surface-normal gradient methods. Additionally, OpenFOAM calls interpolation schemes for general use in meshing and applying the above solvers. Furthermore, OpenFOAM has solvers to calculate curl, flux, surface and volume integrals, averaging functions, compute cell reduction, and smoothers.

Linear solvers also form a core component of a finite volume computation to solve for pressure  $p$  and velocity  $u$ . Four main types are built-in: pre-conditioned (bi-)conjugate gradient (PCG/BiPCG), explicit diagonal, geometric-algebraic multi-grid (GAMG), and smooth solvers. Among these options, a user can specify preconditioner, solution tolerance, iteration limits, and over-/under-relaxation. These solvers are wrapped into the OpenFOAM implementation of the PISO and SIMPLE algorithms. For practical purposes, OpenFOAM only distinguishes between the variants of these algorithms by the number of corrector steps and update loops specified using the built-in linear solvers.

### 3.1.3 OpenFOAM Case File Structure

Solving a CFD problem in OpenFOAM is done by defining a case directory. There are three required subdirectories in each case file: 1) `/0`, the initial and boundary conditions; 2) `/constant`, which contains mesh information and other physical properties; and 3) `/system`, which allows the user to define simulation controls. See Figure 3.1 for each case file structure diagram, classified by solver.

A case requires the initial and boundary conditions for pressure and velocity to be

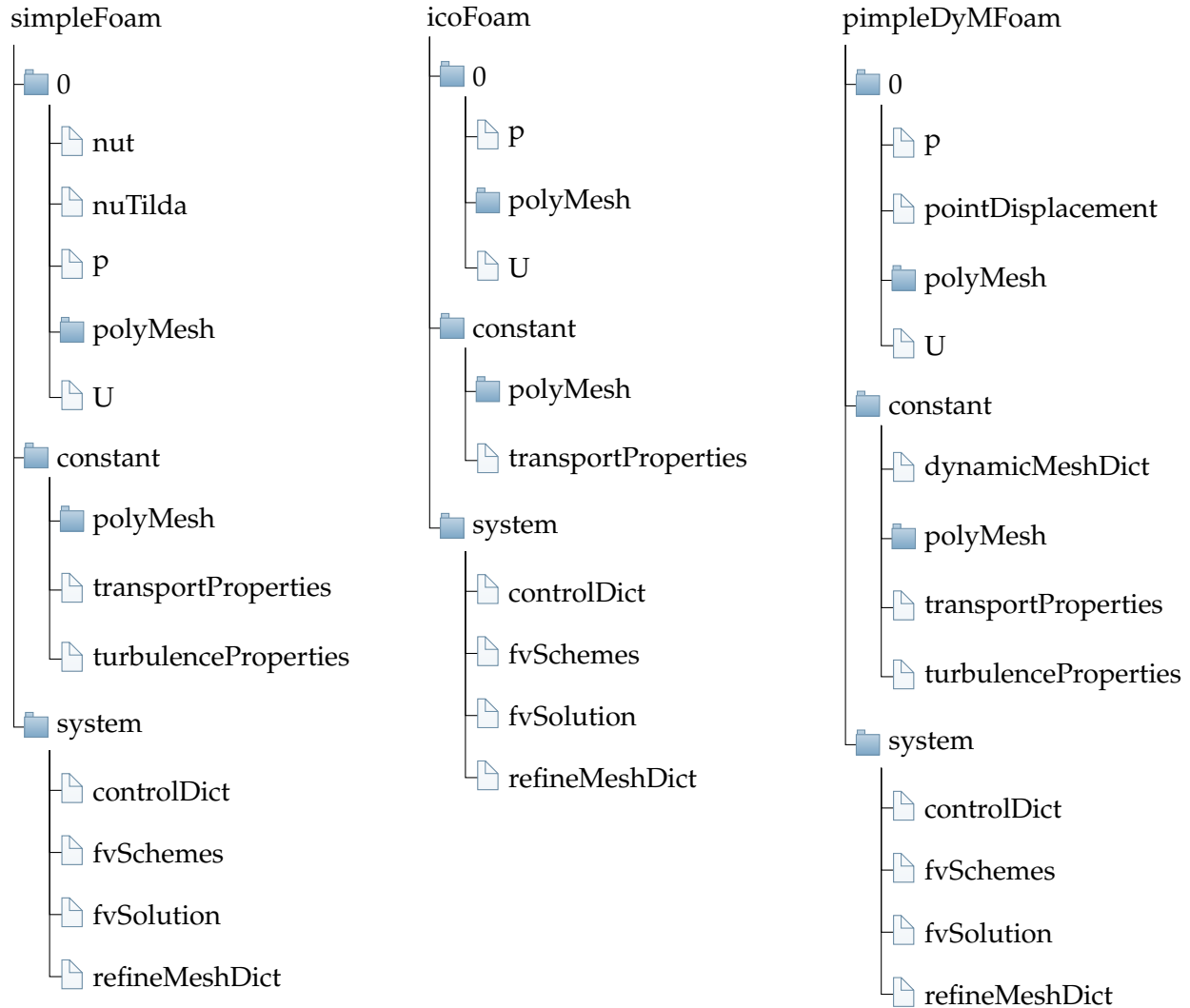


Figure 3.1: OpenFOAM case file structure for each solver type used in this thesis.

defined in the /0 folder.\* These files allow the user to define initial conditions for the internal mesh field(s) and the boundary conditions on each patch face. A comprehensive list of standard boundary conditions are found in Appendix A.4 of [25].

The /constant folder defines the way OpenFOAM interprets the internal and boundary meshes. OpenFOAM populates this directory with geometry information that is indexed and stored through a user-specified OpenFOAM mesh conversion function. The following are currently supported for mesh conversion: ANSYS, CFX 4, DAT, Fluent,

\*For complicated flow simulations, like those featuring transient or turbulent flow, the /0 directory may require additional boundary files. For example, the steady state laminar flow solver, simpleFoam, requires definitions for turbulent viscosity even when used for laminar flow problems.

Gambit, Gmsh, I-Deas, KIVA3v, msh, Netgen, pro-STAR, tetgen, VTK, and OBJ. OpenFOAM is also supplied with a generic mesh generator, blockMesh, which can be used to generate simple graded and curved meshes. This mesh generator is used in conjunction with the snappyHexMesh function to convert a generic geometry file. For this project, I used the conversion utility gmshToFoam to convert Gmsh .msh files to OpenFOAM format.\* Once a mesh file has been converted in OpenFOAM, the /constant folder tells OpenFOAM how to treat specific points and boundaries. It is essential to classify the type for each face, wall, and set of points, and to specify how each wall corresponds to its neighbors. Much of this work is performed by the mesh conversion tool, though the user usually must examine the output and make adjustments. Additionally, parameter values for fluid properties, like transport and turbulence, are specified in this directory.

For all cases, the /system folder requires three dictionaries: controlDict for data input/output control; fvScheme for numerical algorithms; and fvSolution for solution control.

## 3.2 Case Models

In this section, I detail the specific cases used in this project. The OpenFOAM code for each case can be found in Appendix A.

I employed three standard solvers for incompressible flow: simpleFoam is a steady-state solver for turbulent flow, using the SIMPLE algorithm; icoFoam is a transient PISO solver for Newtonian fluids; and pimpleDyMFoam—"PISO-SIMPLE Dynamic Mesh"—a transient solver for turbulent flow of a Newtonian fluid on a moving mesh.

For each of the cases below, kinematic viscosity of air was taken at the body temperature of 37°C, which corresponds to a value of  $\nu = 16.69 \times 10^{-6} \text{ m}^2/\text{s}$ , from [21]. For each case, default discretization schemes were used for time, gradient, divergence, Laplacian,

---

\*A full list of standard mesh conversion utilities can be found in Section 3.6.3 of [14].

Table 3.1: Summary of simulations performed in this thesis.

Case	Model	Flow	Boundary Conditions	Solver
A	Duct	Steady-State	Uniform Inlet	simpleFoam
B	Duct	Transient	Uniform Inlet	icoFoam
C	Duct	Transient	Oscillating Inlet	icoFoam
D	Airway-1	Steady-State	Uniform Inlet	simpleFoam
E	Airway-1	Transient	Uniform Inlet	icoFoam
F	Airway-1	Transient	Oscillating Inlet	icoFoam
G	Airway-2	Transient	Expanding Alveolus Wall	pimpleDyMFoam

interpolation, and surface-normal gradient. The tolerances for the solutions were set to  $10^{-4}$ . For those cases where inlet velocity is defined, the velocity was calculated to force Reynolds number  $Re = 1$ . This choice is supported in the literature: in their in-vitro experiments, Chhabra and Prasad [4] used  $Re = 0.1$  and Ma et al. [22] used  $Re = 0.13$ ; Muller et al. [23] gave a range of  $Re = 5 \times 10^{-3} - 0.5$  in the acinar region; and, Li and Kleinstreuer [21] considered a Reynolds number range in the the alveolar region to be  $0 < Re \leq 11$ .

### 3.2.1 Duct Models

#### Case A: Steady-State Flow

To compute the steady-state solution in the Duct Model, the simpleFoam solver was used. Turbulence parameters, turbulent kinematic viscosity  $\nu_{tilda}$  and turbulence field variable  $\nu_{tilda}$ , from the Spalart-Allmaras [28] model, must be specified since simpleFoam accounts for turbulence. The parameters were set to 0 for laminar flow and turbulence was explicitly defined as 'laminar' in the turbulenceProperties file in the /constant subdirectory. Pressure was supplied with an initial value of atmospheric pressure at the outlet, zero gradient at the inlet, and no-slip boundaries on the walls. Momentum was provided a uniform inlet flow vector (0.069542 0 0) m/s. At the outlet, initial momentum is set to zero gradient; on the walls, momentum is given a no-slip boundary condition.

Since `simpleFoam` is a steady-state solver and does not use time information,  $\Delta T = 1s$  and the maximum number of iterations was 1000.

### **Case B: Constant Transient Flow**

For constant transient flow, the `icoFoam` solver was used. For this solver, it is only necessary to set boundary conditions for pressure and momentum. Pressure was defined with zero gradient at the inlet, atmospheric pressure at the outlet, and no-slip boundary conditions on the walls. Momentum was defined with uniform inlet velocity vector (0.069542 0 0) m/s, zero gradient flow at the outlet, and no-slip boundary conditions on the walls. The case was run for 0.1 seconds,  $\Delta t = 10^{-6}$ . The solution was written every 1000 time-steps.

### **Case C: Oscillating Transient Flow**

For the case with periodic boundary conditions, most of the parameters were inherited from the above case. The necessary changes were written to the momentum file: the inlet was supplied with a periodic boundary condition defined as a uniform fixed value by a uniform sine wave across the boundary. The case was run for 1 second with  $\Delta t = 10^{-6}$ . The velocity amplitude vector was (0.069542 0 0) m/s and the frequency was 2 Hz, which corresponds to a half-second breathing cycle, which was chosen to accelerate computation time due to the small time step. The solution was written every 1000 time-steps.

## **3.2.2 Airway Flow Models**

### **Case D: Steady-State Flow**

I used `simpleFoam` with the Airway-1 Model to compute a steady-state solution. Momentum was provided a fixed inlet vector (0.0575 0 0) m/s with zero gradient at the

outlet. Pressure was set with zero gradient at the inlet and fixed atmospheric pressure at the inlet. The walls were supplied with a no-slip boundary condition. The maximum iteration count was 10,000 iterations.  $\Delta = 1$ .

### **Case E: Constant Transient Inlet Flow with Fixed Boundary**

The icoFoam solver was used with the Airway-1 Model for unidirectional, laminar flow. Uniform inlet flow was set as a fixed velocity vector (0.0575 0 0) m/s. With velocity defined at the inlet, it is standard practice to specify pressure at the outlet; here, pressure at the outlet was set to atmospheric. No-slip boundary conditions were specified for the walls of the geometry.

The case was run for 0.01 seconds,  $\Delta t = 10^{-8}$ . The solution was written every 10 time-steps. In controlling the PISO algorithm, 2 corrector steps for pressure and momentum were used, as well as 2 explicit non-orthogonal correctors of the Laplacian term in the pressure updates.

### **Case F: Oscillating Transient Inlet Flow with Fixed Boundary**

All of the settings for the case of oscillating inlet flow for Airway-1 Model with static boundary were identical to those in case E. The exception is in the definition of the momentum boundary conditions. Here, the inlet velocity was defined as a uniform fixed value across the inlet, where the fixed value was calculated as a sine wave with amplitude vector (0.0575 0 0) m/s. Due to the slow speed of calculation, the case was run for 0.01 seconds. Thus, frequency was set to  $f = 100$  Hz to calculate one complete instance of inflow with returning outflow due to the small time step required for numerical stability.

### **Case G: Transient Flow with Moving Boundary**

This model features the idealized airway model wherein the boundaries of the alveolus sac oscillate and the duct walls remain fixed. This case was run with the Airway-2 Model.



The boundary of the alveolus mesh was divided into four parts: top, right, and bottom, were defined as moving walls; the left alveolus wall was treated as a zeroGradient patch so that no flux is calculated at any point on that wall; fluid simply passes through. The left wall of the duct was left as one continuous wall, defined later as an inlet/outlet patch.

What differentiates this case from all others is the moving boundary. The alveolus wall expands at a prescribed rate along an input vector to cause a pressure-drop that draws fluid into the geometry. Therefore, it is essential to define how the mesh moves. This is done by specifying: 1) mesh motion type; 2) the solver of the cell motion equations; and 3) the case-specific initial conditions that determine direction and magnitude of cell motion.

Mesh motion was defined using the dynamicMeshDict dictionary located in the /constant subdirectory. The natural choice for the mesh solver is displacementLaplacian of the dynamicMotionSolverFvMesh class. This solves the Laplacian on each cell center to calculate motion displacement. (An alternative is to solve the Laplacian for motion velocity, but the input parameters are very sensitive and tuning is difficult.) The coefficients of the Laplacian are calculated for each of the three expanding alveolus faces using diffusion. Here, quadratic inverseDistance was chosen as a robust method, though in future applications directional diffusivity could be used to incorporate the surface normal for each face.

I used the cellDisplacement solver in the fvSolution dictionary, located in /system subdirectory to solve the mesh motion equations. The default option is to use a preconditioned conjugate gradient (PCG) method with a Simplified Diagonal-based Incomplete Cholesky (DIC) preconditioner. Here I should mention that the tolerance for this solver as well as for p, p correctors, U, and UFinal were all set to  $10^{-4}$ , also defined in the fvSolution dictionary.

The initial conditions for the solver are defined in the pointDisplacement file lo-

cated in the /0 subdirectory. This file provides input to the displacementLaplacian solver, which then computes motion by the quadratic inverseDistance method mentioned above. Here, the specific alveolus wall motion is defined. I chose the type oscillatingDisplacement; the displacement oscillates by  $d = A \sin(\omega x)$ , where A is the amplitude of the displacement wave and  $\omega = 2\pi f$ , with f the frequency in Hz. An amplitude corresponding to a 15.6% increase in alveolar sac area was chosen to be consistent with [16], which saw a 15.6% expansion of the alveolar sac volume. See Section 1.2.4 above for details.  $\omega$  was set to 62.8318, corresponding to a 0.1s breathing cycle. Although this is not biologically accurate, it was chosen to speed up computation.

The internal pressure field was set to 0 relative to the atmosphere. The inlet/outlet was given the pressure boundary condition inletOutlet from Appendix A.4 of standard boundary conditions in the OpenFOAM User Guide [14]:

This boundary condition provides a generic outflow condition, with specified inflow for the case of return flow.

In practice, it is typical to define pressure 0 relative to atmospheric for generic outflow condition, and specify 0 relative pressure for inflow. The remaining walls were given a zeroGradient boundary condition. Momentum U was initialized with uniform velocity vector (0 0 0) m/s. The inlet/outlet patch is defined with type pressureInletOutletVelocity; again from Appendix A.4 in [14]:

This velocity inlet/outlet boundary condition is applied to pressure boundaries where the pressure is specified. A zero-gradient condition is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the patch-face normal component of the internal-cell value.

The alveolar walls are of type movingWallVelocity, a no-slip boundary condition for moving walls. This was done to ensure that the fluid is pulled in by the wall movement

as it would happen physiologically. The top and bottom duct walls are given a no-slip boundary condition.

The solver `pimpleDyMFoam` is a variant of the `pimpleFoam` solver, an incompressible, transient solver with generic turbulence modeling. Then `turbulenceProperties` is set to `laminar`; no turbulence variables need definition. The dictionary `fvSolution` specifies the solvers for the components of the PISO-SIMPLE algorithm. For the PISO-SIMPLE algorithm, `nCorrectors` sets the number of times the pressure and momentum components are corrected for each time step. In this case, `nCorrectors` was set to 2. The non-orthogonal correction of the Laplacian was set to 2 correctors. Lastly, the number of times the solver loops over the entire system per time-step is set by `nOuterCorrectors`; here it was 2. Case controls are stipulated in `controlDict`. The case was run for 0.025 seconds,  $\Delta t = 10^{-8}$ . The solution was written every 1000 time steps.

### 3.3 Mesh Generation with Gmsh

Gmsh is a free and open-source three-dimensional finite element mesh generator with built-in pre- and post-processing facilities [10]. Gmsh provides users the option to build geometries and generate meshes either via a simple graphical user interface or by editing text files written in the Gmsh scripting language. A major advantage of the scripting language approach is that geometries can be drafted quickly with place-holders for certain dimension values and rescaled easily by quick changes to the code. Elementary objects like points, lines, and arcs are used to define surfaces which are in turn used to define volumes to be meshed. The default mesh generator produces unstructured meshes composed of triangles in two dimensions and a combination of tetrahedra and hexahedra in three dimensions using the Delaunay triangulation algorithm [7]. Structured meshes are generated by transfinite interpolation algorithm [12].

In this work, (2) two-dimensional structured meshes were created using Gmsh 2.16.0.

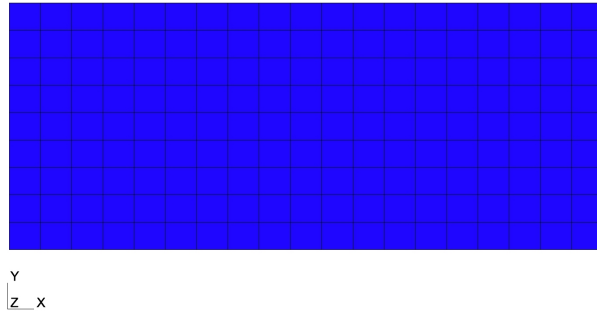
Table 3.2: Geometry specifications for the Duct and Airway Models. Model dimensions are rounded to simplify mesh construction from Harding and Robinson's Alveolar Sac Model [16].

	Current Study		Literature Comparison
	Duct Model	Pipe Model	Alveolar Sac Model (2010)
Duct Length (mm)	0.58	0.58	0.583
Duct Diameter (mm)	0.24	0.24	0.23
Alveolus Radius (mm)	–	0.15	0.145

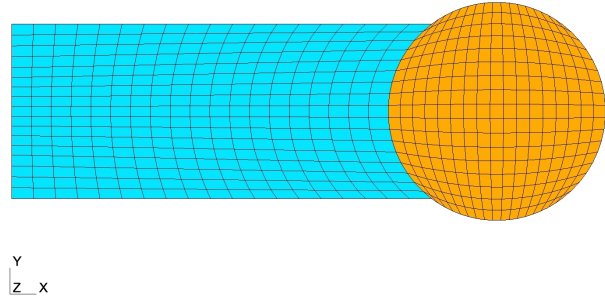
The first, a pipe-like bronchiole Duct Model, was constructed to show that meshes created in Gmsh can be used with several different OpenFOAM case solvers: `simpleFoam` for steady-state flow and `icoFoam` for transient laminar flow with periodic boundary conditions. In order to test model parameters, case set up, and the effect of different OpenFOAM solvers, a simple 2D duct geometry was constructed and a structured mesh generated. The Airway Model was created as an idealized terminal alveolated bronchiole duct and a structured mesh was generated. The dimensions of the Duct and Airway Models are rounded from the dimensions of Harding and Robinson's Alveolar Sac Model [16]. The dimensions of the current study compared to the Alveolar Sac Model are given in Table 3.2.

The Duct Model is a vertical cross section of an axisymmetric pipe representing a bronchiole duct with patch faces "inlet", "outlet", "top" and "bottom", and "frontAndBack". Since OpenFOAM is a FVM solver, it was necessary to extrude both models by some small amount to create a volume. The "frontAndBack" patch therefore refers to the two patch faces in the z-plane with depth 0.01 mm. Specifications made in the OpenFOAM case file signal that this patch can be ignored so that flow is solved in 2D.

The Airway Model was constructed as an idealized terminal alveolated airway, an extension of the Duct Model. This geometry was built for use with three solvers: `simpleFoam` for steady-state flow, `icoFoam` for constant and periodic transient flow, and `pimpleDyMFoam` for pressure-driven transient flow with a moving boundary. This model



(a) Duct Model.



(b) Airway Model.

Figure 3.2: Structured meshes generated in Gmsh by transfinite interpolation for the current study.

features a structured mesh on a polar coordinate system. This construction was observed to tolerate larger time steps than the unstructured prism-based meshes. The geometry was constructed starting with the alveolus such that the right edge of the duct is a concave arc.

A summary of mesh details is given in Table 3.3.

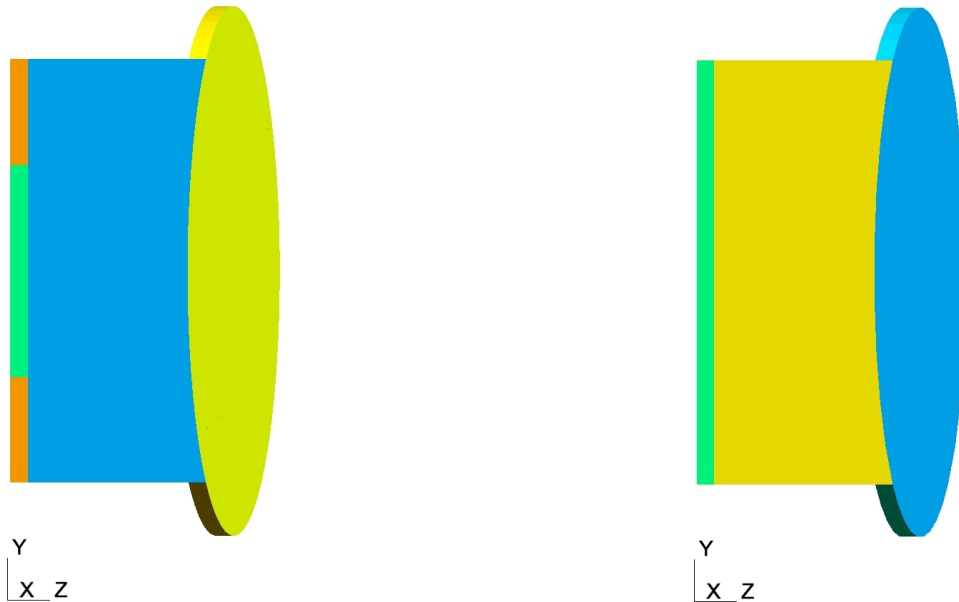
Table 3.3: Mesh details for Duct and Airway Models.

Model	Points	Surface Faces	Internal Faces	Hexahedral Elements
Duct	1,482	2,792	1,312	684
Airway	1,406	2,646	1,242	648

### Airway Model Variants

Two variants of the Airway Model were created. In the first variant, Airway-1 (Figure 3.3a), used with cases D-F, the inlet was divided into three sections covering approximately 25%, 50%, and 25% of the area, respectively. The middle half defined as the inlet and the top and bottom quarters defined as the outlet. Mass must be conserved in the system, so it is necessary to define explicit inlet and outlet patches to allow pressure to equilibrate. Otherwise, the OpenFOAM solver will crash. The second variant, Airway-2 (Figure 3.3b), used with case G, features one edge defined as both inlet and outlet.

The Airway-2 Model takes advantage of a standard OpenFOAM boundary condition



(a) Airway-1 Model. The left face is subdivided into three separate patches: the middle defined as the inlet and the top and bottom both defined as the outlet.

(b) Airway-2 Model has only one face. When used with the `pressureInletOutletVelocity` boundary condition, OpenFOAM calculates which patch face cells are inlet and outlet.

Figure 3.3: Airway Model Variants

`pressureInletOutletVelocity` where the boundary is treated as either inlet or outlet depending on the neighboring cells:

This velocity inlet/outlet boundary condition is applied to pressure boundaries where the pressure is specified. A zero-gradient condition is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the patch-face normal component of the internal-cell value [25].

The meshes built in Gmsh were imported to OpenFOAM using the `gmshToFoam` utility. The mesh was refined in the  $x - y$  direction using the `refineMesh` utility. By default, `refineMesh` is three directional. Bi-directional refinement requires definition in a special `refineMeshDict` dictionary, located in the `/system` subdirectory. The command `transformPoints -scale '(0.001 0.001 0.001)'` scales the mesh to millimeters.

A mesh check was performed on both the pipe and airway models using `checkMesh`, returning an OK check.

### **OpenFOAM-Specific Meshing Considerations**

Special consideration was taken when making the alveolated terminal airway mesh for use with the OpenFOAM dynamic mesh solver, `pimpleDyMFoam`. In this case, an OpenFOAM dictionary, `dynamicMeshDict`, specifies how the mesh is manipulated in two ways: manipulation where the mesh topology is changed and manipulation where it is not. The `dynamicFvMesh` solvers are used in those cases where the mesh topology is left unchanged; e.g. translation, rotation, stretching and compressing cells. The `topoChangerFvMesh` solvers are used in those cases where the mesh topology is changed; e.g. cases of sliding mesh interfaces, cell addition and subtraction. Here, the mesh topology was left unchanged and a `dynamicFvMesh` solver was employed. During mesh design of the Airway Models, the alveolar surface was subdivided into four subregions: three boundary arcs on the top, bottom, and right of the sac were provided parameter definitions for expansion and contraction; the remaining patch region was incorporated into the main duct upon meshing and patch definition. This allowed the `dynamicMeshDict` to be written in a way that permitted the outward expansion and contraction of the alveolar surface and it was done in order to prevent divergence in the `dynamicFvMesh` solver.

## **3.4 Post-Processing in ParaView**

Post-processing for this project was done using ParaView 5.3.0. ParaView is a general open-source visualization toolkit developed by Kitware, Inc., Los Alamos National Laboratory, and Sandia National Laboratory, with funding from the Department of Energy. ParaView supports dozens of file types and is notable for its flexibility across scientific disciplines. The `Glyph Vector` option was used to show the direction of flow.

# Chapter 4

## Results

### 4.1 Duct Model

#### 4.1.1 Case A: Steady-State Flow and Case B: Constant Transient Flow

For modeling the steady-state solution in the duct, the SIMPLE solution converged in 45 iterations. The parabolic flow velocity profile seen in Figure 4.1(a) verifies that OpenFOAM calculates the solution correctly; parabolic duct flow was observed in [16] and [22]. The one-directional transient laminar case demonstrates the same pattern: after 0.1 seconds, the flow profile is identical to the steady state solution. This solution was calculated in 30.93 seconds. The maximum Courant number was 0.0679313.

#### 4.1.2 Case C: Oscillating Transient Flow

For the case of oscillating transient flow through the duct, see Figures 4.1(c)-(f). The solution was computed for one second to simulate two complete breathing cycles with flow coming from only one opening, as it would in a bronchiole duct. The solution was computed in 317.38 seconds. The Courant number never grew larger than  $2.12587 \times 10^{-6}$ . At time 0.125s, the flow is fastest in the positive x-direction. The magnitude in that

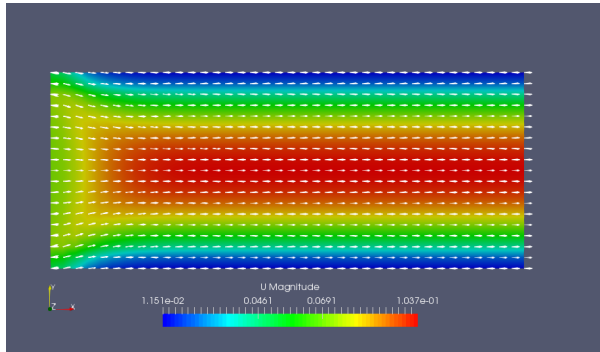


direction decreases and at 0.25s the direction changes. It can be seen at in 4.1(d) & (f) that the flow profile is not uniform across the geometry when the direction changes. Flow separates into distinct channels: one through the center and channels along the top and bottom duct walls that flow opposite of the center channel. Recirculation was observed in two zones near the inlet above and below the middle stream. At 0.375s, the magnitude of the velocity profile is the greatest in the negative  $x$ -direction.

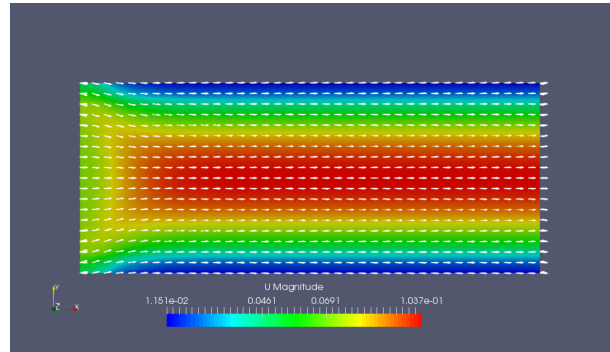
One might expect that if Figure 4.1(c) were flipped across the  $y$ -axis, then the result should look like the outflow when the magnitude is the greatest in the negative direction. Looking closely at Figure 4.1(e), the characteristic diffusion of the velocity seen in the heat map of right-left flow is in the same place as the left-right flows; Figure 4.1(e) is not mirror symmetric as expected. My proposed explanation: whereas the boundary condition on the inlet is given a specific value at each time step, the condition on the outlet is unassuming. Velocity at the outlet is defined as "zero gradient", from Appendix A.4 in [14]:

This boundary condition applies a zero-gradient condition from the patch internal field onto the patch faces.

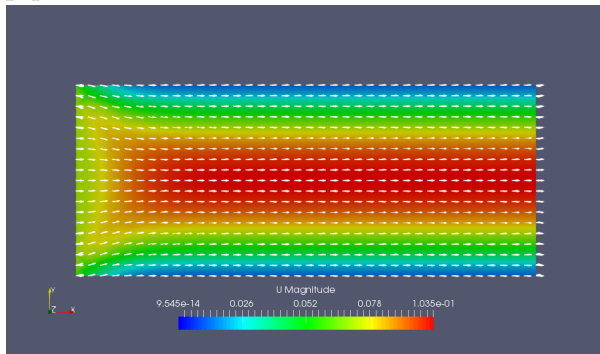
That is, at 0.375s the inward velocity into the right hand side is taken from internal patch faces, so the diffusion witnessed on the left hand side of Figure 4.1(c) is absent from the right hand side of Figure 4.1(e). The diffusion on the left hand side of Figure 4.1(e) comes from the flow moving to the external geometry where no boundary conditions are defined. Figure 4.1(f) shows the start of recirculation and also a divided profile similar to Figure 4.1(d) at 0.5 seconds as the sinusoidal velocity vector changes sign to point in the positive  $x$ -direction.



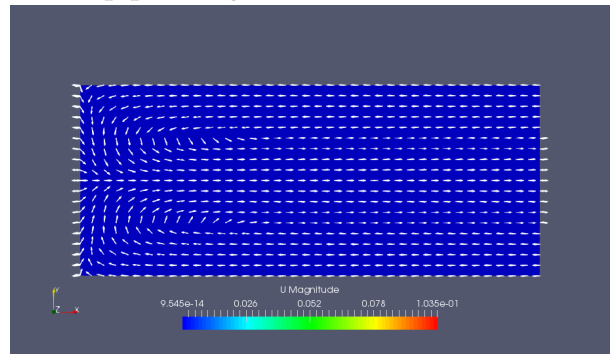
(a) Converged SIMPLE solution on the 2D pipe.



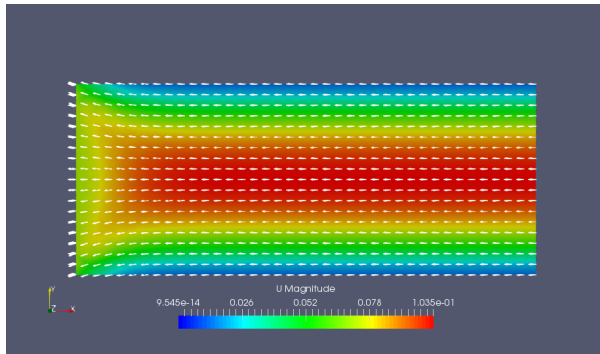
(b) Fully developed velocity profile at 0.1s in the 2D pipe using the icoFoam solver.



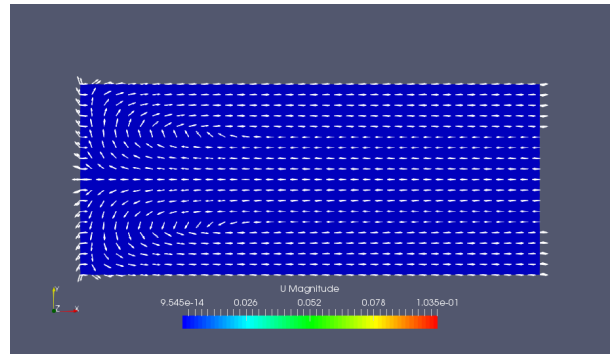
(c) 0.125s.



(d) 0.25s.



(e) 0.375s.



(f) 0.5s.

Figure 4.1: Velocity profiles for 2D pipe flow. (a) The steady state solution using simpleFoam converged in 45 iterations. (b) Fully developed flow after 0.1 seconds using icoFoam. (c) - (f) Directed flow profiles for the 2D duct with a sinusoidal periodic boundary conditions at the inlet using icoFoam; frequency  $f = 2$  Hz, amplitude  $A = 0.069542$  m/s.

## 4.2 Airway Model

To compute the steady-state solution of flow into the airway model, mesh Model 2-1, with separate inlet and outlet, was employed. Attempts at a steady-state solution using the same geometry with differently defined inlets and outlets caused OpenFOAM to diverge or crash.

### 4.2.1 Case D: Steady-State Flow and Case E: Constant Transient Flow

For steady-state flow, the `simpleFoam` solver was used. The SIMPLE algorithm converged in 72 iterations in 0.46 seconds. The greatest velocity magnitude was observed on the left hand side at the transitions between inlet and outlet. At the inlet, inward flow pressing against the internal field caused the fluid to turn back toward the outlets to conserve momentum and pressure. To the left of the middle of the duct, flow moved outward before turning inward from the top and bottom walls, forming two recirculation zones in the middle of the duct. Straight through the centerline of the geometry, toward the alveolar sac, flow moved uniformly in the  $x$ -direction. At the transition from duct to alveolus, some recirculation was observed. Toward the top and bottom edges of the sac, flow is pushed outward as it approaches the fixed boundary. The glyph vectors illustrate that low-magnitude flow pushed against the alveolar sac flows, practically demanding expansion of the walls.

For the case of transient flow into the airway model, the `icoFoam` solver was used. Immediately after the case-start, flow in the airway appeared similar to Figure 4.2, though no recirculation in the alveolar sac was observed. The flow entered at the inlet and the greatest velocity magnitude was observed at the transition to the outlet. For each of Figure 4.3(a)-(e), recirculation zones in the center of the duct were observed. Recirculation in the alveolar sac is best described as chaotic; no eddies or pattern-like recirculation was observed. The `icoFoam` solver required 968.62 seconds to compute the solution. The

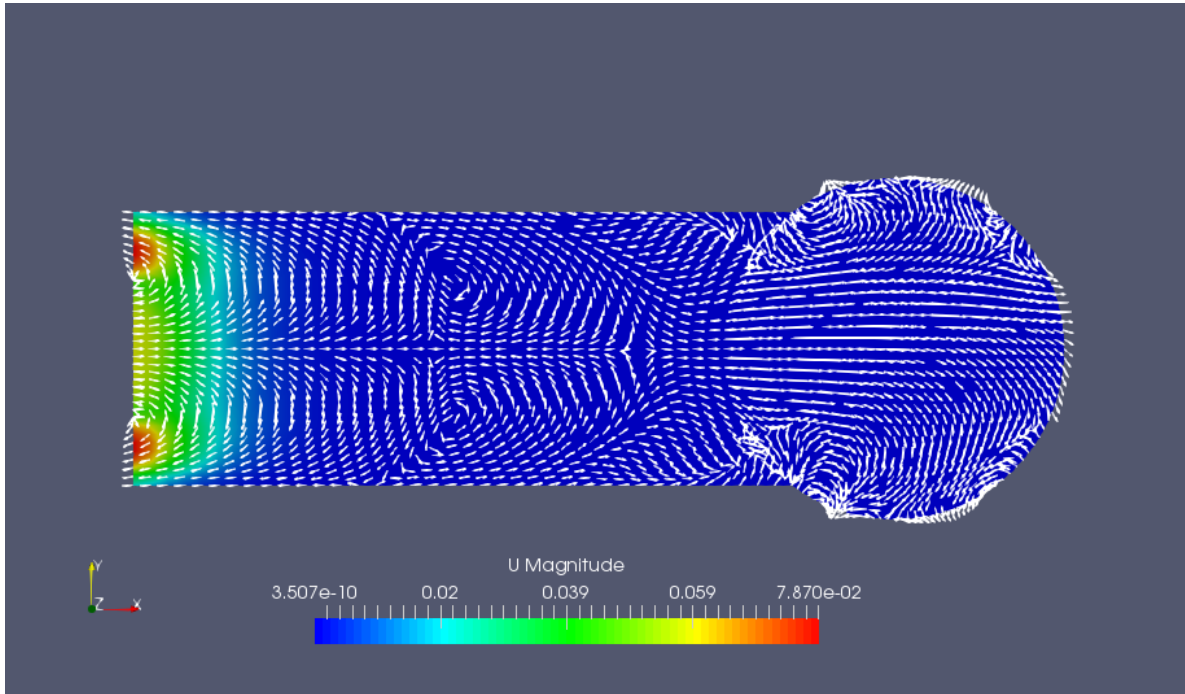


Figure 4.2: Steady state solution for the idealized terminal alveolated airway using simpleFoam. The steady state solution converged in 72 iterations.

Courant number never exceeded maximum value 0.000972216.

#### 4.2.2 Case F: Oscillating Transient Flow

The patterns in the airway model were more complex when the boundary conditions were periodic. The solution was computed using icoFoam. After only 0.0001 seconds, the beginnings of recirculation can be seen throughout the geometry. At 0.0025 seconds, the peak of inflow, the flow was similar to the simpleFoam and unidirectional icoFoam solutions. At time 0.005 seconds, when the direction of flow is zero, the velocity profile becomes more interesting. Recirculation was observed near the inlet/outlets, near the center of the airway duct, and in the alveolar sac. From left to right, The first recirculation zones were symmetric across the centerline. Recirculation in the alveolus appears more chaotic. At 0.0075 seconds, the reverse flow profile was fully developed: outflow from the "inlet" and inward through the "outlets". The normal recirculation zones found

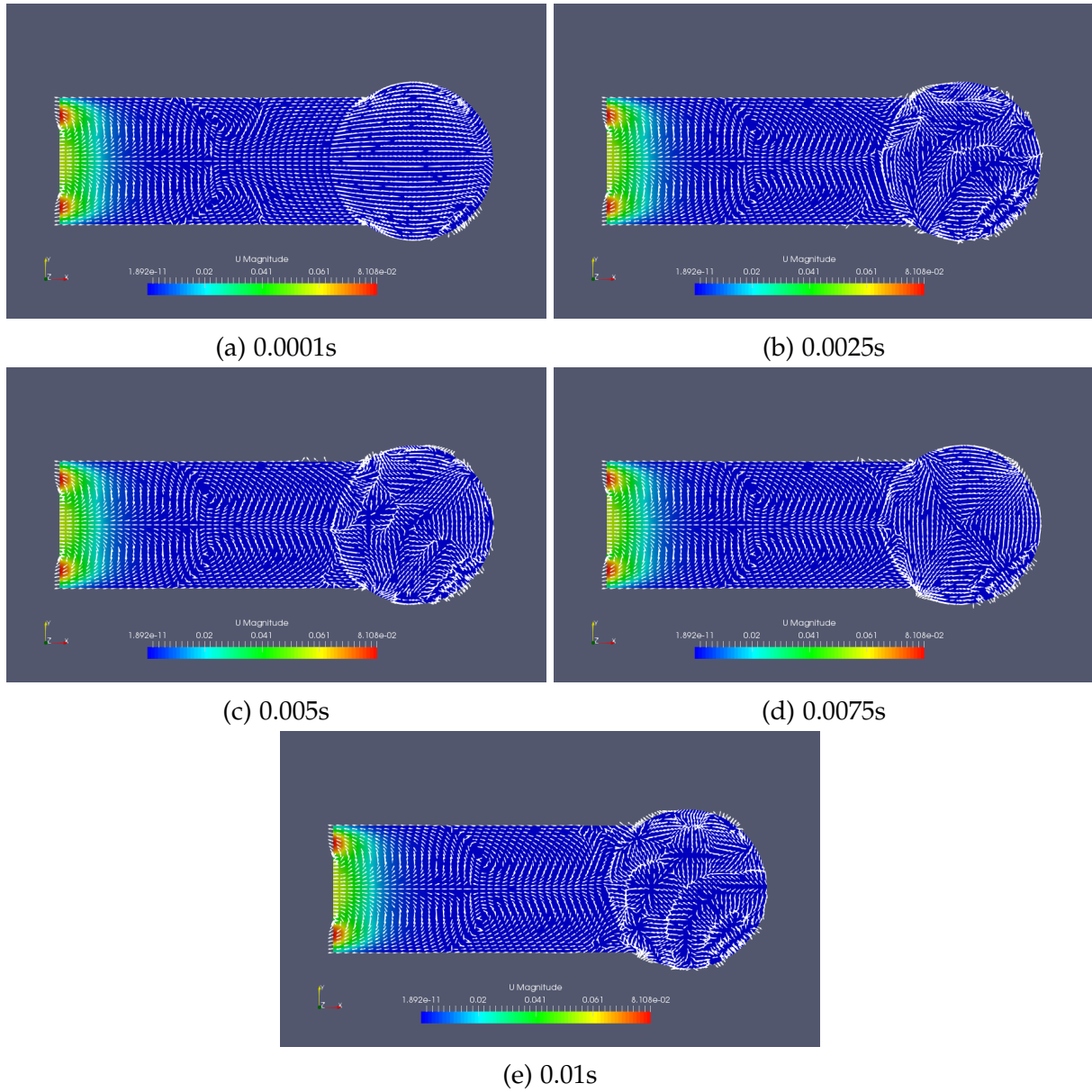


Figure 4.3: Transient laminar solutions on the idealized terminal alveolated Airway-1 Model using *icoFoam*. The inlet velocity is uniform 0.0575 m/s in the x-direction.

in the center of the airway duct, also seen in Figures 4.2 and 4.3(a)-(e) were observed here. At 0.01 seconds, multiple recirculation zones were observed, as in 4.4(c): near the inlet/outlets, to the left of the middle of the duct, and in the alveolar sac. These last two recirculation zones appear to be closer to axisymmetric than in 4.4(c). The solution was computed in 875.49 seconds. The Courant number never exceeded  $3.99728 \times 10^{-07}$ .

### 4.2.3 Case G: Transient Flow with Expanding Boundary

The case of flow into 2D airway with moving boundary was computed using the `pimpleDyMFoam` solver. The alveolar sac walls were expanded 15.6% using the displacement-Laplacian function. At 0.0001 seconds, when the rate of expansion of the alveolus was greatest, the largest magnitude of velocity was observed, as seen in Figure 4.5(a). As the sac expansion slows down, the fluid enters the airway at a decreasing rate, as seen in Figures 4.5(b)-(f). For each of Figures 4.5(a)-(e), flow into the airway was uniform in the  $x$ -direction throughout the duct. In the sac, the fluid moves outward toward the expanding walls. Due to the mesh construction, the expansion of the cells by the dynamic mesh motion solvers warped the mesh around the vertices used to construct the mesh. Since a circular/spherical model was one geometry option among several polyhedral-based models (see [21], [16], [4], [23], and [22]), this pinched circular geometry, which enjoyed the macroscopy of a general lung model but also the local flare of a biological alveolus at simulation end-time, was deemed acceptable for the case.

At the latest time in Figure 4.5(f), 0.025 seconds, the alveolus walls stopped expanding and fluid striking the alveolar sac walls began to recirculate in the sac. Fluid in the duct separated into two distinct flow directions: in the top and bottom of the duct, fluid pointed outward; in the middle of the duct, fluid pointed toward the alveolus sac. For this case, the `pimpleDyMFoam` solver required 6020.94 seconds. The maximum Courant number was  $3.02816 \times 10^{-8}$ .

Unidirectional wall expansion was the only case for which a solution could be found.

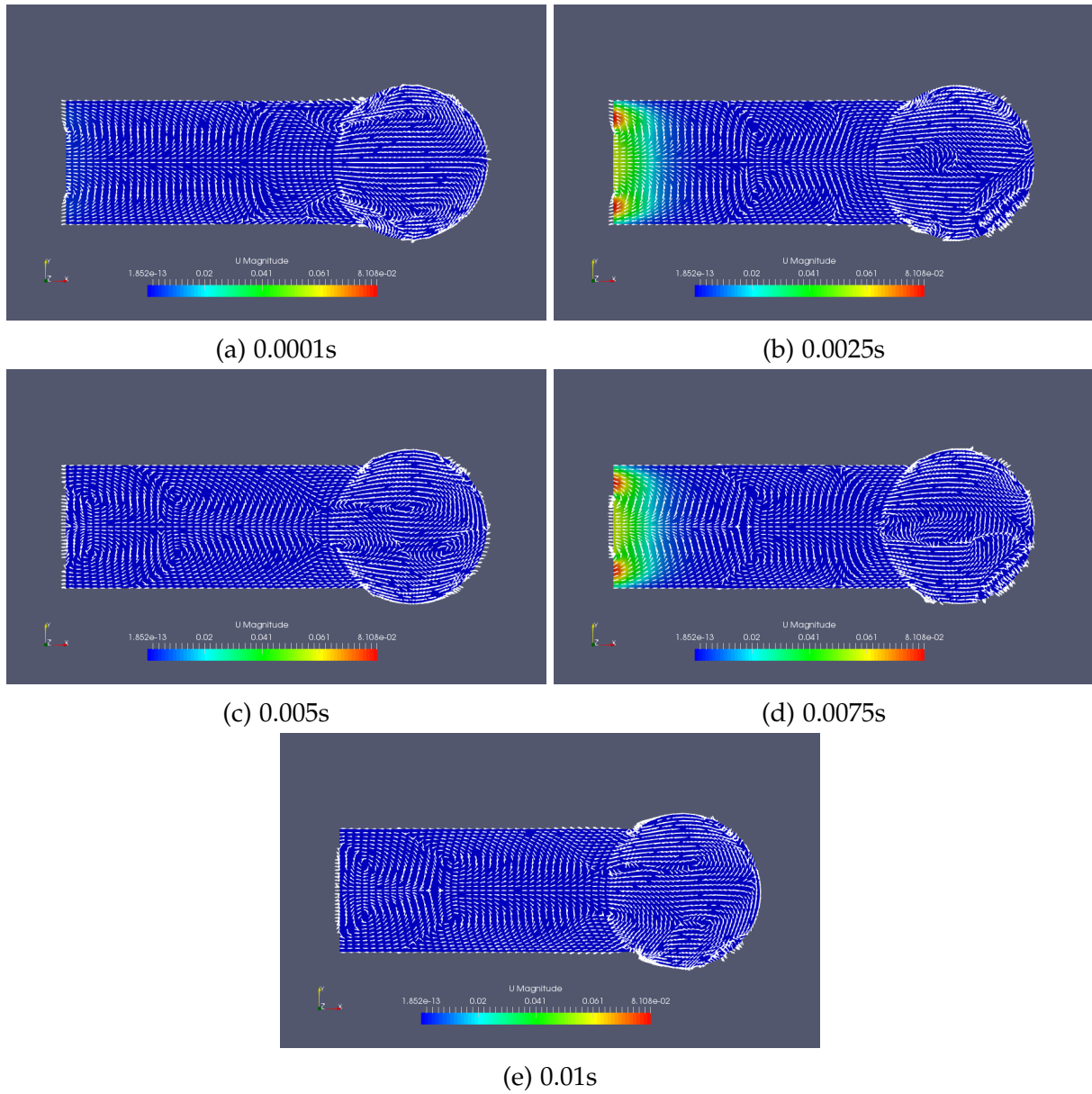
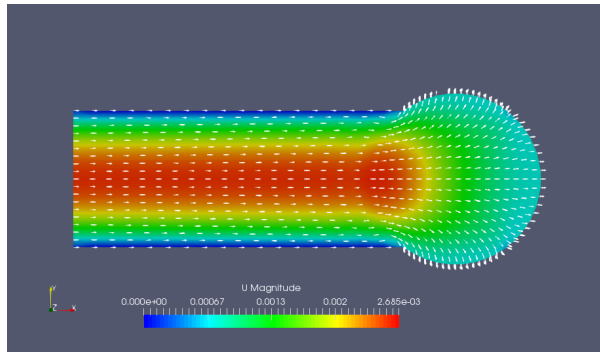


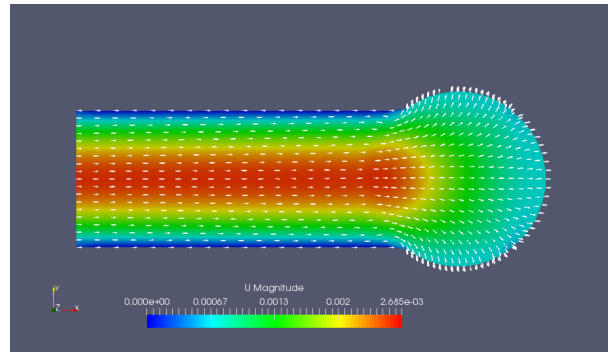
Figure 4.4: Transient laminar solutions with periodic boundary conditions defined on the inlet for the idealized terminal alveolated Airway-1 Model using *icoFoam*. The inlet velocity vector was sinusoidal:  $(0.0575 \ 0 \ 0)$  m/s with frequency  $f = 100$  Hz.

I attempted to define the inlet and outlets in such a way that would permit the fluid to flow out of the airway when the sac compressed and also attempted the simulation with the Airway-1 Model, but each trial resulted in divergence. Typically, the Courant number grew exponentially and `pimpleDyMFoam` crashed at 0.025004 seconds; the computed velocity diverged to infinity.

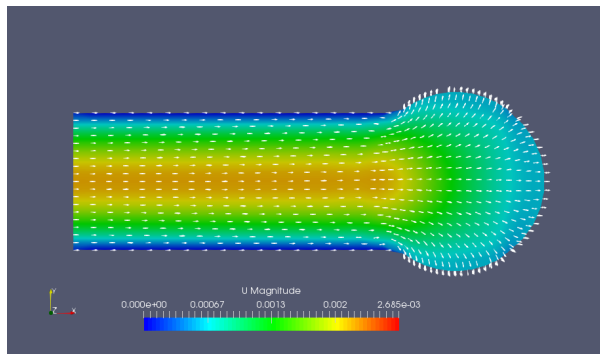




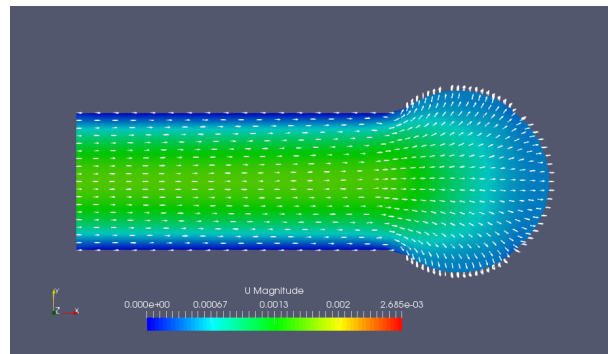
(a) 0.0001s



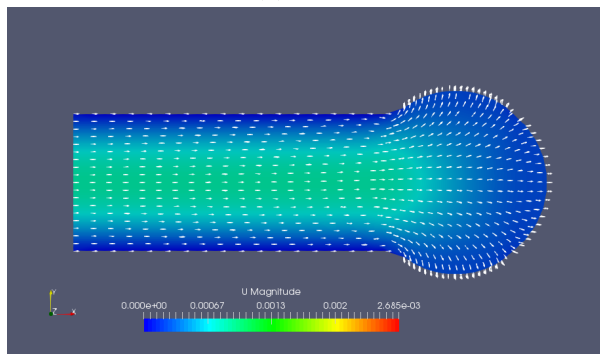
(b) 0.005s



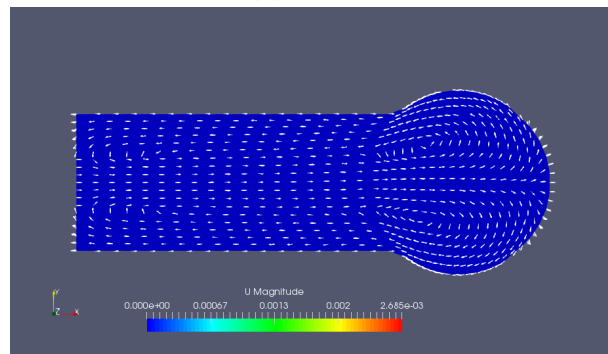
(c) 0.01s



(d) 0.015s



(e) 0.02s



(f) 0.025s

Figure 4.5: Transient laminar solutions on the idealized terminal alveolated airway with moving boundary using `pimpleDyMFoam`. The walls of the alveolar sac stretch outward with sinusoidal cell displacement equivalent to a 15.6%; frequency  $f = 10$  Hz, amplitude  $A = 2.34 \times 10^{-5}$  mm. The inlet and outlet were defined on the same edge. Neither inlet nor outlet was provided an initial condition for flow; flow was driven by wall expansion.

# Chapter 5

## Discussion

A growing body of research modeling airflow into the lungs greatly motivates the need for free and open-source software that is widely distributed and easy to implement. This would democratize research and accelerate the pace of discovery. Many researchers, scientists, and engineers in the developing world use OpenFOAM for their work.\* Many key advancements to OpenFOAM are coming from researchers in Europe, India, and China.† While many users are solving problems in mechanical or aerospace engineering, popularizing the use of OpenFOAM for problems in biomathematics and mathematical physiology is central to the development of new cases, solvers, and distributions for easy use.

In this chapter, I will compare results in the airway model, discuss some issues with OpenFOAM that must be addressed for widespread use to occur, and outline future avenues of development of and research using OpenFOAM.

---

\*This was determined by extensive searching through the online CFD community websites; e.g. the user forum of [cfd-online.com](http://www.cfd-online.com).

†The online comments/contributions of Bernhard Gschaider, Hrvoje Jasak, and Bruno Santos on [cfd-online.com](http://www.cfd-online.com) and the course site of *CFD with OpenSource Software* by Hakan Nilsson (Chalmers University of Technology, <http://www.tfd.chalmers.se/~hani/kurser/OS.CFD/>) were useful.

## 5.1 Airway Model Comparisons

### 5.1.1 Steady State vs. Transient Flow

Comparing the steady-state solution and the transient solutions for the airway model, there is strong agreement in the recirculation patterns in the main duct. Flow near the inlet and outlets agree for both the steady-state solution and all transient solutions.

At  $t = 0.0001s$ , the recirculation zones are axisymmetric across the centerline. Flow in the duct and in the alveolus are pointed back toward the recirculation zones except for an outward bulge in lower right quadrant of the alveolus. This is possibly due to numerical errors associated with the relatively low tolerances that were used to allow for the solution to be computed quickly.

After  $t = 0.0001s$ , the recirculation zones are fixed and symmetric across the centerline axis. Between the recirculation zones and the left boundary of the alveolar sac, fluid moves inward from the walls to broadly recirculate down the centerline. The interaction of fluid in the duct with the fluid at the left boundary of the alveolus—similar to a brackish confluence—determines the chaotic nature of flow in the alveolus.

The change in the curl of the vector field, where inflow in the duct meets outflow from the alveolus, was witnessed between time steps as a traveling wavefront. This wavefront can be seen along the lower wall of the duct at times  $t = 0.0025s$  and  $t = 0.005s$ , moving closer to the left boundary of the alveolus. No wavefront is observed along the upper wall of the duct. In these time-steps, the recirculation pattern in the alveolus is chaotic.

At  $t = 0.0075s$ , flow in the alveolus appears to be a saddle node with the unstable equilibrium located at the center of the alveolus. Inward flow originates from the upper left and lower right quadrants of the alveolar sac walls, the wavefront along the top wall of the duct now visible. In the final time-step, the wavefront is again visible on the bottom; both the top and bottom wavefronts appear to move away from the left sac boundary. Flow inside the sac is again chaotic, similar to times  $t = 0.0025s$  and

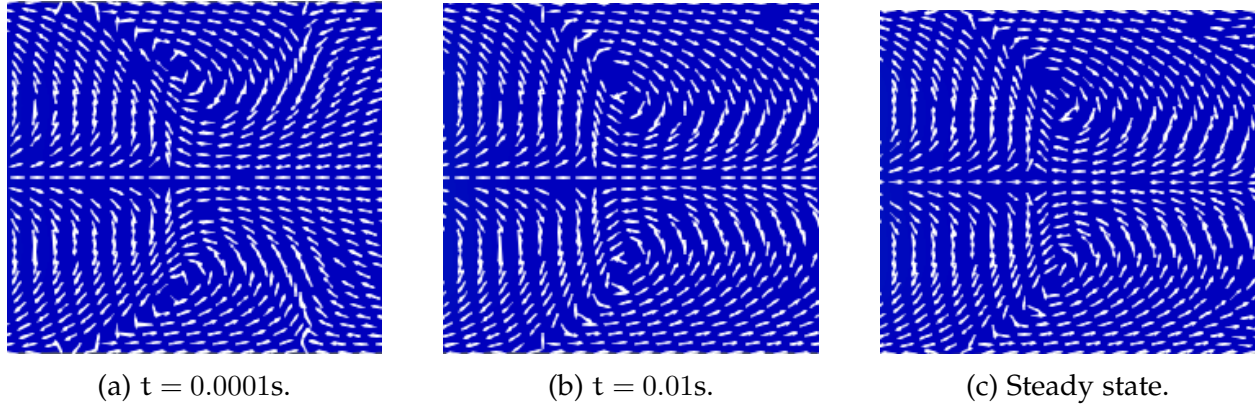


Figure 5.1: Recirculation zones for airway model in the duct center. The vector fields of the transient solution at  $t = 0.01s$  (b) and the steady state solution (c) are highly similar.

$t = 0.005s$ . The steady state solution shows wavefronts on along both top and bottom of the duct in roughly the same position as at time  $t = 0.01s$ .

### 5.1.2 Transient Flow With and Without Periodic Boundary Conditions

It is important to compare the behavior of the solver `icoFoam` on the airway model with uniform inlet flow (Figure 4.3, case E) and sinusoidal inlet flow (Figure 4.4, case F). The maximum velocity between both cases is equal while the minima of the two cases are not the same:  $1.892 \times 10^{-11}$  for oscillating flow versus  $1.852 \times 10^{-13}$  for constant flow. Making a side-by-side comparison of (a)-(e) in Figures 4.3 and 4.4, it is clear that the flow in the alveolus in case F is less chaotic.

At  $t = 0.0001s$ , flow inside the alveolus of case E (Figure 4.3(a)) is near zero, pointed outward from the sac. In case F (Figure 4.4, recirculation patterns are immediately apparent on the interior and the top and bottom of the left boundary of the alveolus. On the other hand, recirculation in the duct is only observed in case E. Flow in the duct of case F is laminar, symmetric across the  $x$ -axis. A likely explanation is that the uniform inlet flow hits a wall of zero velocity fluid, causing recirculation in the duct. Sinusoidal inlet flow permits smoother flow through the centerline with recirculation observed nearer to the boundaries, especially around the alveolus.

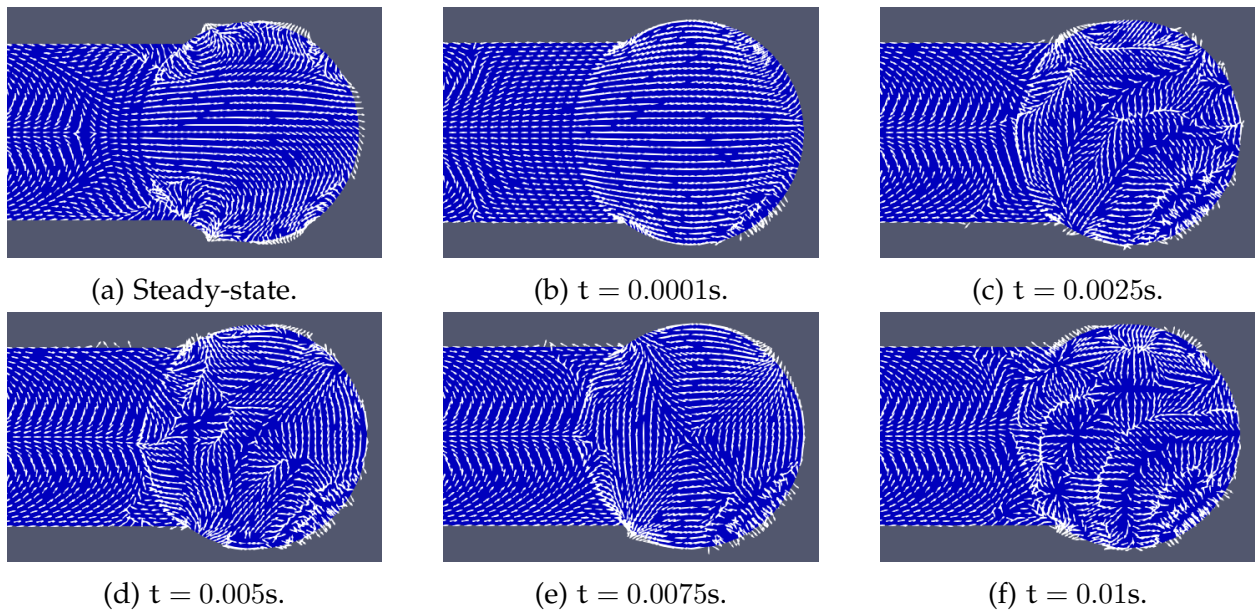


Figure 5.2: Flow patterns in the alveolus and traveling wavefronts for the steady-state (a) and transient (b)-(f) solutions. The steady-state wavefront of "brackish" fluid can be seen in 5.2(a) at the bottom and top left of the sac boundary. The wavefront is seen only in the bottom of the duct in (c) and (d). It approaches the alveolus boundary at each additional time step. The wavefront is then visible in the upper portion of (e). Chaotic recirculation is observed for the transient solutions (c)-(f). A saddle point is observed in the center of the sac in (e).

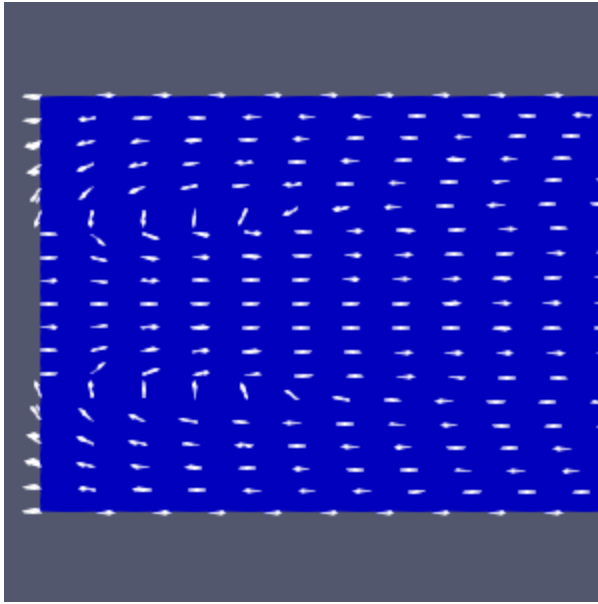
At  $t = 0.0025s$ , flow in the duct is identical for cases E and F. The confluence observed in the lower portion of the duct in Figure 4.3(b) is observed in both the upper and lower portions of the duct in case F. As mentioned above, recirculation in the alveolus in case E is chaotic. Case F demonstrates more obvious pattern behavior: flow in the alveolus is largely unaffected by changes in velocity. At  $t = 0.005s$ , recirculation in the alveolus of case E is chaotic. In case F, two recirculation zones are visible in the duct: one immediately after the inlet/outlet and a second closer to the middle of the duct. Flow out of the alveolus moves down the centerline. Flow enters the alveolus from the top and bottom of the duct. Recirculation occurs when the flow that follows the top and bottom outer boundaries of the sac meet at the middle of the right wall of the alveolus. For times  $t = 0.0075s$  and  $0.01s$ , the comparisons of cases E and F are similar to those at times  $t = 0.0025s$  and  $0.005s$ , though the direction of flow for case F is reversed: chaotic recirculation is observed in case E; smooth, laminar recirculation is observed in case F.

### 5.1.3 Pressure-Driven Flow with Moving Boundary

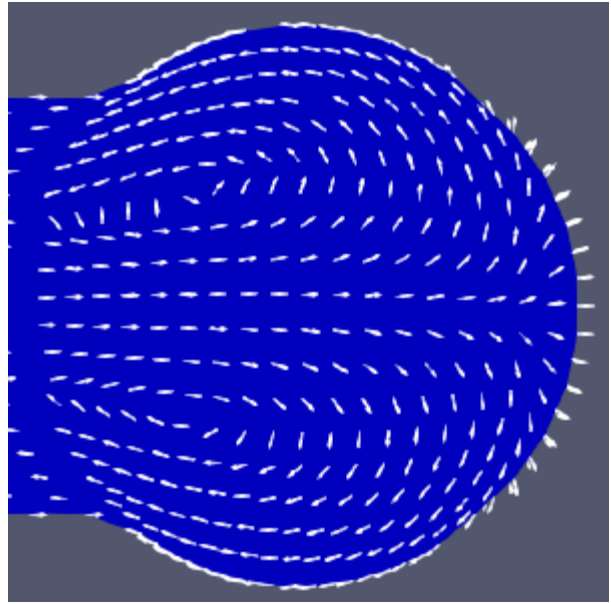
It is important to note that the case of flow in the airway due to expanding alveolar sac walls has the least maximum velocity of any case examined,  $2.685 \times 10^{-3}$  m/s. This directly impacts the Reynolds number. Since the characteristic diameter of the mesh is 0.30mm, the diameter of the sac, this momentum corresponds to Reynolds number  $Re = 0.04826$ . This value agrees with the Reynolds numbers mentioned in the literature in Section 3.2: the Reynolds number falls within the ranges of Muller et al. [23] and Li and Kleinstreuer [21].

For all time-steps during alveolus wall expansion, fluid enters and passes through the duct uniformly. Flow in the sac moves outward directly toward the expanding segments of wall. See Figures 4.5(a)-(d).

Through the course of the simulation, the magnitude of the velocity field decreases at each point. At  $t = 0.025s$ , the alveolus reached its maximum size. The magnitude



(a) OpenFOAM dynamic mesh solver automatically calculates the boundary type along the left patch faces as either inlet or outlet.



(b) Recirculation in the airway model terminal sac. Inflow comes down the centerline through the duct, spreads to the outer walls, and flows out through channels at the top and bottom of the duct-sac boundary.

Figure 5.3: A detailed view of the fluid dynamics at final time  $t = 0.025s$  in the expanding sac airway model.

of the velocity field at each point is near zero, though one can see the direction of the velocity vector at each point: fluid enters the mesh uniformly through the inlet down the centerline of the duct, continuing to the right boundary of the alveolus. At the right boundary, fluid begins to recirculate, diverging from the center to follow the boundary edge around the top and bottom of the alveolus. See Figure 5.3b. The velocity field then flows along the top and bottom of the duct outward through the outlet. Recall that the left edge of the mesh is defined as type `pressureInletOutletVelocity`: the solver calculates if a given cell face is an inlet or outlet; the type is not pre-determined.

In an actual human lung, airflow in a bronchiole duct diffuses along a concentration gradient caused by the uptake of oxygen ( $O_2$ ) and the expiration of carbon dioxide ( $CO_2$ ), resulting in bi-directional flow in the duct [17]. In evaluating the use of OpenFOAM for modeling lung physiology, it is of central importance that the software automatically computes bi-directional flow for the alveolus with moving wall, in addition to unidirectional flow when the wall is only expanding. Here, approximately 2/3 of the left patch faces are calculated as the outlet; the remaining 1/3 of patch faces are calculated as the inlet. See Figure 5.3a. Since OpenFOAM calculates bi-directional flow in addition to unidirectional flow under expanding wall, it has been shown that OpenFOAM can model physiologically realistic flow in simplified and idealized geometries.

## 5.2 Challenges

The potential of using OpenFOAM for mathematical physiology is bounded by its accessibility. The learning curve is very steep: documentation is poor and very disjointed. Error messages are frequently uninformative.

While the `simpleFoam` and `icoFoam` solvers are well-documented, documentation for the `pimpleDyMFoam` solver and necessary files was sparse at best and misleading at worst. To illustrate my point, consider `dynamicMotionSolverFvMesh`, the dynamic mesh engine



that calculates mesh motion and provides feedback to the solver for simulation whose only documentation comes from OpenFOAMWiki, a community development site run by Bernhard Gschaider of HFD Research:

`dynamicMotionSolverFvMesh`—This solver morphs the mesh around a specified set of boundaries. The meshing motion is calculated based on the pressures on those boundaries. In turn, the `dynamicMotionSolverFvMesh` provides feedback to the fluid simulation. It alters the velocity boundary conditions (U field) on the included boundaries to specify the local velocity of the defined body. This local velocity includes coupled translation and rotational motions, if permitted. This mesh control is almost exclusively used to solve problems involving rigid body motion. [26]

The above description initially suggests that the solver is well-suited for the purposes of pressure-driven flow in an expanding terminal alveolar sac. However, the last sentence may mislead a new user to find some other method if the problem is not a rigid body problem. To highlight this point, the tutorial associated with this focuses exclusively on a six degree-of-freedom rigid body problem and is narrowly written, as though the solver could only be used for any other purpose. The solver that is detailed is `sixDoFRigidBodyMotion` and the `sixDoFRigidBodyMotionCoeffs` field defines mesh morphing control, six degrees of freedom solver control, body, force, and motion definitions, and output control. Neither the documentation nor tutorial describe the general solvers included in the general mesh motion solver library, which currently contains 10 classes of mesh motion solvers that are not Six Degrees of Freedom solvers. Additionally, the documentation and tutorial do not direct the user to define a solver in the `fvSolution` dictionary, located in `/system`, a necessary step.

This example demonstrates a disadvantage that pervades OpenFOAM: much of the documentation and many of the tutorials are written narrowly. This may convince new users that their problem is out of the reach of the current distribution of OpenFOAM.

```

0 Foam::error::printStack(Foam::Ostream&) at ???
1 Foam::sigFpe::sigHandler(int) at ???
2 ? in "/lib/x86_64-linux-gnu/libc.so.6"
3 Foam::GAMGSolver::scale(Foam::Field<double>&,
Foam::Field<double>&, Foam::lduMatrix const&,
Foam::FieldField<Foam::Field, double> const&,
Foam::UPtrList<Foam::lduInterfaceField const> const&,
Foam::Field<double> const&, unsigned char) const at ???
4 Foam::GAMGSolver::Vcycle(Foam::PtrList<Foam::lduMatrix::smoother> const&,
Foam::Field<double>&, Foam::Field<double> const&,
Foam::Field<double>&, Foam::Field<double>&, Foam::Field<double>&,
Foam::Field<double>&, Foam::Field<double>&,
Foam::PtrList<Foam::Field<double> >&,
Foam::PtrList<Foam::Field<double> >&, unsigned char) const at ???
5 Foam::GAMGSolver::solve(Foam::Field<double>&,
Foam::Field<double> const&, unsigned char) const at ???
6 Foam::fvMatrix<double>::solveSegregated(Foam::dictionary const&) at ???
7 Foam::fvMatrix<double>::solve(Foam::dictionary const&) at ???
8 ? at ???
9 __libc_start_main in "/lib/x86_64-linux-gnu/libc.so.6"
10 ? at ???

```

Figure 5.4: Sample error messages, which are frequently uninformative.

While a determined user may find success getting results despite this deficiency, a more general approach is essential for new users.

Error messages in OpenFOAM can be notoriously cryptic, and even useless. This can make debugging very challenging. For example, consider a case mentioned in 4.2.3. There, I explained that no solution could be found for the case where the alveolar sac contracted after expansion. When the `pimpleDyMFoam` solver crashed, it was preceded by exponential growth in the Courant number. The error message:

Line 1 says that a floating point error occurred; i.e. the solver divided by 0 at that moment in the calculation. Lines 3-5 say that the geometric-algebraic multigrid (GAMG) solver crashed. Since this solver was used in the pressure-corrector step, it gives the hint that pressure is poorly defined. Lines 6-7 suggest the solver crashed during some matrix calculation. Lines 8-10 are unintelligible output. Error messages like this are typical.

While some information can be gleaned that tells the user there is an issue in a general case setup, all details must be inferred. One might conclude that the boundary conditions on pressure should be reworked. There are many standard boundary conditions that could be used for situations when pressure is specified at the inlet or outlet. But valuable time may be spent testing out dozens of options that may not be applicable even though a generic description—e.g. Appendix A.4 [25]—suggests it is. On the other hand, the model geometry may be poorly constructed or patch faces poorly defined and, as a result, do not work with the case solver. Finding the perfect balance between a properly-constructed mesh and a correctly-defined case file, often with little help from the User Guide [14] or the online OpenFOAM community, could sidetrack a new user indefinitely.

### 5.3 Future Work

This project serves as a proof-of-concept of the use of OpenFOAM for physiology modeling. Further research into computational fluid dynamics problems in mathematical biology/physiology will drive the need for inexpensive software solutions. Before that happens, several milestones must be reached.

Detailed tutorials are needed and in particular, an in-depth tutorial of the airway model with expanding boundary is necessary. A tutorial should include general parameter settings and solver definitions necessary to build a case with expanding boundary using `pimpleDyMFoam`. It is also essential to create a template for a moving boundary case where the topology is left unchanged. Modeling of the lung in OpenFOAM, as well as the heart, would greatly benefit from a general template.

Documentation must be brought to the level of other open-source mathematical language/software, e.g. Octave, R, Sage, and others. The lack of advanced documentation is a great disadvantage when considering OpenFOAM versus proprietary alternatives.

Finally, OpenFOAM must move from the command line to a graphical user interface. More rapid development will likely come on the heels of a system that attracts users. This may sound like too much emphasis is placed on aesthetic but is a valid reason new users may reach for FLUENT or MATLAB when developing CFD cases, especially those with moving boundary or fluid-surface interaction.

## 5.4 Conclusions

In this work, I used three open-source software packages to:

1. Create two-dimensional meshes of a duct and idealized terminal alveolar airway.
2. Construct fluid flow cases for testing the OpenFOAM software.
3. Successfully tested the capability of OpenFOAM to model an idealized lung model where fluid flow is induced by a pressure-drop, similar to the actual biological process.
4. Visualized the results with a post-processor.

The geometries and meshes were created in Gmsh; I utilized both the built-in scripting language and the graphical user interface to construct structured meshes similar in dimension to actual lung models found in the literature. The fluid flow cases—steady-state flow, constant transient flow, oscillating transient flow, and pressure-driven flow with moving boundary—were written as text files and run with three flow solvers in OpenFOAM. Results were visualized in ParaView; Glyph Vectors were used to demonstrate velocity field profiles detailing the flow structure through the different meshes.

The primary goal of this project was proof-of-concept. The final result was achieved: an idealized terminal alveolar sac airway model was developed and an OpenFOAM case file was written wherein the expanding alveolar sac boundary caused fluid to be drawn into the alveolus with no inlet or outlet velocity specified. Though the model only works

for expansion and not contraction, the freeze-frame solution of the model at the moment of change from expansion to contraction indicates that the OpenFOAM solver calculated which cells to consider as inlet and which to consider as outlet and that separate flow channels arise naturally. In conclusion, OpenFOAM can effectively run two-dimensional computational models of the lung on the scale of 800  $\mu\text{m}$ , where an expanding boundary induces bi-directional flow that is similar to flow patterns in the human lung.

# Bibliography

- [1] Ahrens, J., Geveci, B., Law, C. (2005). ParaView: Paraview: An end-user tool for large data visualization. *The Visualization, Handbook*, 717.
- [2] Arcadian. (Artist). (2006). Bronchi, Bronchial Tree, and Lungs [Digital Image]. Retrieved on 6 August 2017 from [https://commons.wikimedia.org/wiki/File:Illu\\_bronchi\\_lungs.jpg](https://commons.wikimedia.org/wiki/File:Illu_bronchi_lungs.jpg)
- [3] Bradie, B. (2006). *A Friendly Introduction to Numerical Analysis*. New York City: *Pearson*.
- [4] Chhabra, S. & Prasad, A. (2010). Flow and particle dispersion in a pulmonary alveolus – Part I: Velocity measurements and convective particle transport. *J. Biomechanical Eng.* 132(5), 051009-1–051009-12.
- [5] Chorin, A. J. & J. E. Marsden, J.E. (1990). *A Mathematical Introduction to Fluid Mechanics*. Berlin: *Springer-Verlag*.
- [6] Cotes, J. E. (2009). *Lung Function: Physiology, Measurement, and Application in Medicine*. Hoboken: *Wiley*.
- [7] Delaunay, B. (1934). Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles*, 6, 793–800.

- [8] Fefferman, C. L. (2017). Existence and Smoothness of the Navier-Stokes Equation. *Clay Mathematical Institute*. Retrieved on 10 July 2017 from <http://www.claymath.org/sites/default/files/navierstokes.pdf>
- [9] Freed, A. D., Einstein, D. R., Carson, J. P., & Jacob, R. E. US Department Of Energy. (2012). Viscoelastic Model for Lung Parenchyma for Multi-Scale Modeling of Respiratory System Phase II: Dodecahedral Micro-Model. *Pacific Northwest National Laboratory, Report No. PNNL-21287*. Retrieved on 10 June 2017 from [http://www.pnnl.gov/main/publications/external/technical\\_reports/pnnl-21287.pdf](http://www.pnnl.gov/main/publications/external/technical_reports/pnnl-21287.pdf).
- [10] Geuzaine, C. & Remacle, J.-F. (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11), 1309-1331.
- [11] Gockenback, M. S. (2006). Understanding and Implementing the Finite Element Method. Philadelphia: *SIAM*.
- [12] Gordon, W. & Hall, C. (1973). Construction of curvilinear coordinate systems and application to mesh generation. *International Journal for Numerical Methods in Engineering*, 7, 461-477. doi:10.1002/nme.1620070405.
- [13] Greenshields, C. J. (2015). OpenFOAM Programmer's Guide version 3.0.1. *The OpenFOAM Foundation, CFD Direct Ltd.*. Retrieved on 24 April 2017 from <https://cfd.direct/openfoam/user-guide/>
- [14] Greenshields, C. J. (2016). OpenFOAM User Guide version 4.0. *The OpenFOAM Foundation, CFD Direct Ltd.*. Retrieved on 24 April 2017 from <https://cfd.direct/openfoam/user-guide/>

- [15] Hall, N. (Ed.). (2015). Gas properties definitions. *Glenn Research Center, National Aeronautics and Space Administration*. Retrieved 26 April 2017 from <https://www.grc.nasa.gov/WWW/k-12/airplane/gasprop.html>
- [16] Harding, E. M. & Robinson, R. J. (2010). Flow in a terminal alveolar sac model with expanding walls using computational fluid dynamics. *Inhalation Toxicology*, 22(8), 669–678.
- [17] Hogg, J. C., Paré, P. D., Hackett, T.-L. (2017). The contribution of small airway obstruction to the pathogenesis of chronic obstructive pulmonary disease. *Physiol Rev*, 97, 529–552. DOI: 10.1152/physrev.00025.2015
- [18] Hofemeier, P. & Sznitman, J. (2015). Revisiting pulmonary acinar particle transport: convection, sedimentation, diffusion and their interplay. *Journal of Applied Physiology*. DOI: 10.1152/jappphysiol.01117.2014
- [19] Issa, R. I. (1986). Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62(1), 40–65.
- [20] Kelley, C. T. (1995). Iterative methods for linear and nonlinear equations. Philadelphia: SIAM.
- [21] Kleinstreuer, C. & Li, Z. (2011). Airflow analysis in the alveolar region using the lattice-Boltzmann method. *Med. Biol. Eng. Comput.*, 49, 441–451.
- [22] Ma, B. et al. (2009). CFD simulation and experimental validation of fluid flow and particle transport in a model of alveolated airways. *J. Aerosol Science*, 40, 403–414.
- [23] Muller, P. et al. (2014). Maximal efficiency of convective mixing occurs in mid acinus: A 3D-numerical analysis by an Eulerian approach. *J. Aerosol Science*, 76, 163–174.



- [24] OpenFOAM (®) Extended Code Guide. (2017). *OpenCFD Ltd., ESI Group*. Retrieved 27 May 2017 from <http://www.openfoam.com/documentation/cpp-guide/html/openfoam-guide.html/>
- [25] OpenFOAM (®) Documentation: User Guide. (2016). *OpenCFD Ltd., ESI Group*. Retrieved 27 May 2017 from <http://www.openfoam.com/documentation/user-guide/>
- [26] Parameter Definitions - dynamicMotionSolverFvMesh (21 October 2016). In *OpenFOAMWiki*. Retrieved 5 June 2017 from [https://openfoamwiki.net/index.php/Parameter\\_Definitions.\\_dynamicMotionSolverFvMesh/](https://openfoamwiki.net/index.php/Parameter_Definitions._dynamicMotionSolverFvMesh/)
- [27] Patankar, S. V. (1980). Numerical Heat Transfer and Fluid Flow. USA: *Hemisphere*.
- [28] Spalart, P. R. & Allmaras, S. R. (1994). A One-Equation Turbulence Model for Aerodynamic Flows. *Recherche Aerospaciale*, 1, 1994, 5–21.
- [29] Tsuda, A., Butler, J. P. , Fredberg, J. J. (1994). Effects of alveolated duct structure on aerosol kinetics I. Diffusional deposition in the absence of gravity. *J. Applied Physiology*, 76, 2497–2509.
- [30] Weibel, E. R. (1963). Morphometry of the Human Lung. Heidelberg: *Springer Berlin Heidelberg*.

# Appendix A

## OpenFOAM Code

This chapter includes the case files written for cases A-G. The code is organized by the three main subdirectories in each case file: /0, constant, & /system.

This section only includes user-defined files. Files generated by OpenFOAM utilities like gmshToFoam are not included. Files in the following subdirectories of each case were excluded: /0/polyMesh and /constant/polyMesh.

### A.1 Case A

#### A.1.1 /0

/nut

```
1  /*-----*-- C++ *-----*\
2  / ===== / /
3  /  \ \ /  F i e l d      / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ /  O p e r a t i o n / Version: 4.1 /
5  /  \ \ /  A n d      / Web: www.OpenFOAM.org /
6  /  \ \ /  M a n i p u l a t i o n / /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        nut;
```

```

14 }
15 // ***** //
16
17 dimensions      [0 2 -1 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type      fixedValue;
26         value     uniform 0;
27     }
28
29     outlet
30     {
31         type      fixedValue;
32         value     uniform 0;
33     }
34
35     walls
36     {
37         type      fixedValue;
38         value     uniform 0;
39     }
40
41     frontAndBack
42     {
43         type      empty;
44     }
45 }
46
47 // ***** //

```

/nuTilda

```

1  /*-----* C++ *-----*\
2  | ===== |
3  | \\ / Field | OpenFOAM: The Open Source CFD Toolbox |
4  | \\ / Operation | Version: 4.1 |
5  | \\ / And | Web: www.OpenFOAM.org |
6  | \\ / Manipulation |
7  \*-----*\
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        volScalarField;

```



```

12     class      volScalarField;
13     object     p;
14 }
15 // * * * * *
16
17 dimensions    [0 2 -2 0 0 0 0];
18
19 internalField uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type     zeroGradient;
26     }
27     outlet
28     {
29         type     fixedValue;
30         value    uniform 0;
31     }
32     walls
33     {
34         type     zeroGradient;
35     }
36     frontAndBack
37     {
38         type     empty;
39     }
40 }
41
42 // * * * * *

```

/U

```

1 /*-----*- C++ -*-----*/
2 / ===== /
3 / \ / \ / F i e l d / OpenFOAM: The Open Source CFD Toolbox /
4 / \ / \ / O p e r a t i o n / Version: 4.1 /
5 / \ / \ / A n d / Web: www.OpenFOAM.org /
6 / \ / \ / M a n i p u l a t i o n / /
7 /*-----*-*/
8 FoamFile
9 {
10     version    2.0;
11     format     ascii;
12     class      volVectorField;
13     object     U;
14 }
15 // * * * * *

```

```

16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField    uniform (0 0 0);
20
21 boundaryField
22 {
23     inlet
24     {
25         type          fixedValue;
26         value          uniform (0.069542 0 0); // DD = 0.24mm
27     }
28
29     outlet
30     {
31         type          zeroGradient;
32     }
33
34     walls
35     {
36         type          noSlip;
37     }
38     frontAndBack
39     {
40         type          empty;
41     }
42 }
43
44 // ***** //

```

## A.1.2 /constant

/transportProperties

```

1  /*-----*- C++ -*-----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1  /
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g  /
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  /*-----*-
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;

```

```
15 }
16 // *****
17
18 transportModel Newtonian;
19
20 rho [1 -3 0 0 0 0] 1;
21
22 nu [0 2 -1 0 0 0] 16.69e-6;
23
24 // *****

                                /turbulenceProperties
1 /*-----*-- C++ -----*\
2 /===== / /
3 / \\ / F i e l d / OpenFOAM: The Open Source CFD Toolbox /
4 / \\ / O p e r a t i o n / Version: 4.1 /
5 / \\ / A n d / Web: www.OpenFOAM.org /
6 / \\ / M a n i p u l a t i o n / /
7 \*-----*/
8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object         turbulenceProperties;
15 }
16 // *****
17
18 simulationType laminar;
19
20 RAS
21 {
22     RASModel      SpalartAllmaras;
23
24     turbulence     off;
25
26     printCoeffs   off;
27 }
28
29 // *****
```



### A.1.3 /system

/controlDict

```
1 /*----- C++ -----*/
2 / ===== /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / Operation / Version: 4.1 /
5 / \ \ / And / Web: www.OpenFOAM.org /
6 / \ \ / Manipulation /
7 /*-----*/
8 FoamFile
9 {
10     version 2.0;
11     format ascii;
12     class dictionary;
13     location "system";
14     object controlDict;
15 }
16 // ***** //
17
18 application simpleFoam;
19
20 startFrom startTime;
21
22 startTime 0;
23
24 stopAt endTime;
25
26 endTime 1000;
27
28 deltaT 1;
29
30 writeControl timeStep;
31
32 writeInterval 100;
33
34 purgeWrite 0;
35
36 writeFormat ascii;
37
38 writePrecision 6;
39
40 writeCompression off;
41
42 timeFormat general;
43
44 timePrecision 6;
45
```



```

46  runTimeModifiable true;
47
48
49  // ***** //

                                /fvSchemes

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / A nd / Web: www.OpenFOAM.org /
6  /  \ \ / M anipulation / /
7  \*-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSchemes;
15 }
16 // * * * * * //
17
18 ddtSchemes
19 {
20     default        steadyState;
21 }
22
23 gradSchemes
24 {
25     default        Gauss linear;
26 }
27
28 divSchemes
29 {
30     default        none;
31     div(phi,U)      bounded Gauss linearUpwind grad(U);
32     div(phi,nuTilda) bounded Gauss linearUpwind grad(nuTilda);
33     div((nuEff*dev2(T(grad(U)))) Gauss linear;
34 }
35
36 laplacianSchemes
37 {
38     default        Gauss linear corrected;
39 }
40
41 interpolationSchemes
42 {

```

```

43     default          linear;
44 }
45
46 snGradSchemes
47 {
48     default          corrected;
49 }
50
51 wallDist
52 {
53     method meshWave;
54 }
55
56
57 // ***** //

                                     /fvSolution

1  /*-----*-- C++ -----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSolution;
15 }
16 // ***** //

17
18 solvers
19 {
20     p
21     {
22         solver      GAMG;
23         tolerance   1e-03;
24         relTol      0.1;
25         smoother    GaussSeidel;
26     }
27
28     U
29     {
30         solver      smoothSolver;
31         smoother    GaussSeidel;

```

```

32     nSweeps      2;
33     tolerance    1e-03;
34     relTol       0.1;
35 }
36
37 nuTilda
38 {
39     solver        smoothSolver;
40     smoother      GaussSeidel;
41     nSweeps       2;
42     tolerance     1e-03;
43     relTol        0.1;
44 }
45 }
46
47 SIMPLE
48 {
49     nNonOrthogonalCorrectors 0;
50     pRefCell      0;
51     pRefValue     0;
52
53     residualControl
54     {
55         p          1e-03;
56         U          1e-3;
57         nuTilda    1e-3;
58     }
59 }
60
61 relaxationFactors
62 {
63     fields
64     {
65         p          0.3;
66     }
67     equations
68     {
69         U          0.7;
70         nuTilda    0.7;
71     }
72 }
73
74
75 // ***** //

```

/refineMeshDict

```

1 /*-----*-----*-----*-----*-----*-----*-----*-----*-----*
2 /===== /

```

```

3  /  \  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x      /
4  /  \  /  O p e r a t i o n      /  V e r s i o n :  4 . 1      /
5  /  \  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g      /
6  /  \  /  M a n i p u l a t i o n      /
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // * * * * *
16
17 set                fluid;
18
19 coordinateSystem   global;
20
21 globalCoeffs
22 {
23     tan1            (1 0 0);
24     tan2            (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology     true;
34
35 geometricCut       false;
36
37 writeMesh          false;

```

## A.2 Case B

### A.2.1 /0

/p

```

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x      /
4  /  \  /  O p e r a t i o n      /  V e r s i o n :  4 . 1      /

```

```

5  /  \ \ /  A nd           / Web:    www.OpenFOAM.org           /
6  /  \ \ /  M anipulation /                                     /
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14 }
15 // *****
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField    uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type        zeroGradient;
26     }
27     outlet
28     {
29         type        fixedValue;
30         value        uniform 0;
31     }
32     walls
33     {
34         type        zeroGradient;
35     }
36
37     frontAndBack
38     {
39         type        empty;
40     }
41 }
42
43 // *****

```

/U

```

1  /*-----*-- C++ --*-----*/
2  / ===== /
3  /  \ \ /  F ield           / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ /  O peration       / Version:  4.1                       /
5  /  \ \ /  A nd             / Web:      www.OpenFOAM.org           /
6  /  \ \ /  M anipulation    /                                     /
7  /*-----*/

```

```

8 FoamFile
9 {
10     version      2.0;
11     format       ascii;
12     class        volVectorField;
13     object       U;
14 }
15 // * * * * *
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23
24     inlet
25     {
26         type      fixedValue;
27         value     uniform (0.069542 0 0);
28     }
29     outlet
30     {
31         type      zeroGradient;
32     }
33     walls
34     {
35         type      noSlip;
36     }
37     frontAndBack
38     {
39         type      empty;
40     }
41 }
42
43 // *****

```

## A.2.2 /constant

/transportProperties

```

1 /*-----*-- C++ --*-----*\
2 / ===== /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / Operation / Version: 4.1 /
5 / \ \ / And / Web: www.OpenFOAM.org /
6 / \ \ / Manipulation /
7 /*-----*--

```



```

30 writeControl    timeStep;
31
32 writeInterval   1000;
33
34 purgeWrite      0;
35
36 writeFormat     ascii;
37
38 writePrecision  6;
39
40 writeCompression off;
41
42 timeFormat      general;
43
44 timePrecision   3;
45
46 runTimeModifiable true;
47
48
49 // ***** //

                                /fvSchemes

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /
7  \*-----*-----*\
8  FoamFile
9  {
10     version    2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      fvSchemes;
15 }
16 // ***** //

17
18 ddtSchemes
19 {
20     default     Euler;
21 }
22
23 gradSchemes
24 {
25     default     Gauss linear;
26 }

```



```

27
28 divSchemes
29 {
30     default          none;
31     div(phi,U)      Gauss limitedLinearV 1;
32 }
33
34 laplacianSchemes
35 {
36     default          Gauss linear corrected;
37 }
38
39 interpolationSchemes
40 {
41     default          linear;
42 }
43
44 snGradSchemes
45 {
46     default          corrected;
47 }
48
49
50 // ***** //

                                     /fvSolution

1  /*-----*-- C++ -----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1  /
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g  /
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*--\

8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     location     "system";
14     object       fvSolution;
15 }
16 // ***** //

17
18 solvers
19 {
20     p
21     {
22         solver      PCG;

```

```

23     preconditioner  DIC;
24     tolerance      1e-3;
25     relTol         0.05;
26 }
27
28 pFinal
29 {
30     $p;
31     relTol         0;
32 }
33
34 U
35 {
36     solver          smoothSolver;
37     smoother        symGaussSeidel;
38     tolerance       1e-03;
39     relTol          0;
40 }
41 }
42
43 PISO
44 {
45     nCorrectors     2;
46     nNonOrthogonalCorrectors 2;
47 }
48
49
50 // ***** //

                                /refineMeshDict

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ /  /  O p e r a t i o n / Version: 4.1 /
5  /  \ \ /  /  A n d            / Web: www.OpenFOAM.org /
6  /  \ \ /  /  M a n i p u l a t i o n / /
7  \*-----*--*-----*\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // ***** //
16
17 set                fluid;
18

```

```

19 coordinateSystem      global;
20
21 globalCoeffs
22 {
23     tan1                (1 0 0);
24     tan2                (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology         true;
34
35 geometricCut           false;
36
37 writeMesh              false;

```

## A.3 Case C

### A.3.1 /0

/p

```

1  /*-----*-- C++ --*-----*/
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M : T h e O p e n S o u r c e C F D T o o l b o x /
4  /  \ \ /  /  O p e r a t i o n /  V e r s i o n : 4 . 1 /
5  /  \ \ /  /  A n d           /  W e b :      w w w . O p e n F O A M . o r g /
6  /  \ \ /  /  M a n i p u l a t i o n /
7  /*-----*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14 }
15 // * * * * *
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField    uniform 0;
20

```

```

21 boundaryField
22 {
23     inlet
24     {
25         type          zeroGradient;
26     }
27     outlet
28     {
29         type          fixedValue;
30         value         uniform 0;
31     }
32     walls
33     {
34         type          zeroGradient;
35     }
36
37     frontAndBack
38     {
39         type          empty;
40     }
41 }
42
43 // ***** //

```

/U

```

1  /*-----*-- C++ -----*\
2  / ===== / /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation / /
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        volVectorField;
13     object       U;
14 }
15 // ***** //
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23

```

```

24 inlet
25 {
26     type                uniformFixedValue;
27     uniformValue        sine;
28     uniformValueCoeffs
29     {
30         frequency       2;
31         amplitude        0.069542;
32         scale            (1 0 0);
33         level            (0 0 0);
34     }
35 }
36 outlet
37 {
38     type                zeroGradient;
39 }
40 walls
41 {
42     type                noSlip;
43 }
44 frontAndBack
45 {
46     type                empty;
47 }
48 }
49
50 // ***** //

```

### A.3.2 /constant

/transportProperties

```

1  /*-----*- C++ -*-----*\
2  / ===== / /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation / /
7  /*-----*- /
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;
15 }
16 // ***** //

```

```

17
18 nu                [0 2 -1 0 0 0 0] 16.69e-6;
19
20
21 // *****

```

### A.3.3 /system

```

                                /controlDict

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        controlDict;
15 }
16 // *****
17
18 application      icoFoam;
19
20 startFrom        latestTime;
21
22 startTime        0;
23
24 stopAt           endTime;
25
26 endTime          1;
27
28 deltaT           1e-5;
29
30 writeControl      timeStep;
31
32 writeInterval     1000;
33
34 purgeWrite        0;
35
36 writeFormat       ascii;
37
38 writePrecision    6;

```

```

39 writeCompression off;
40
41 timeFormat      general;
42
43 timePrecision   3;
44
45 runTimeModifiable true;
46
47
48
49 // ***** //

                                /fvSchemes

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*/

8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSchemes;
15 }
16 // ***** //
17
18 ddtSchemes
19 {
20     default      Euler;
21 }
22
23 gradSchemes
24 {
25     default      Gauss linear;
26 }
27
28 divSchemes
29 {
30     default      none;
31     div(phi,U)   Gauss limitedLinearV 1;
32 }
33
34 laplacianSchemes
35 {

```

```

36     default      Gauss linear corrected;
37 }
38
39 interpolationSchemes
40 {
41     default      linear;
42 }
43
44 snGradSchemes
45 {
46     default      corrected;
47 }
48
49
50 // ***** //

                                /fvSolution

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*-----*\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSolution;
15 }
16 // ***** //
17
18 solvers
19 {
20     p
21     {
22         solver      PCG;
23         preconditioner  DIC;
24         tolerance    1e-3;
25         relTol       0.05;
26     }
27
28     pFinal
29     {
30         $p;
31         relTol      0;

```



```

32     }
33
34     U
35     {
36         solver          smoothSolver;
37         smoother        symGaussSeidel;
38         tolerance       1e-03;
39         relTol          0;
40     }
41 }
42
43 PISO
44 {
45     nCorrectors        2;
46     nNonOrthogonalCorrectors 2;
47 }
48
49
50 // ***** //

                                /refineMeshDict

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // ***** //
16
17 set                fluid;
18
19 coordinateSystem   global;
20
21 globalCoeffs
22 {
23     tan1            (1 0 0);
24     tan2            (0 1 0);
25 }
26
27 directions

```

```

28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology      true;
34
35 geometricCut       false;
36
37 writeMesh           false;

```

## A.4 Case D

### A.4.1 /0

/nut

```

1 /*-----*-- C++ --*-----*\
2 / ===== /
3 / \ \ / / F i e l d / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / / O p e r a t i o n / Version: 4.1 /
5 / \ \ / / A n d / Web: www.OpenFOAM.org /
6 / \ \ / / M a n i p u l a t i o n / /
7 /*-----*-----*\
8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        nut;
14 }
15 // ***** //
16
17 dimensions      [0 2 -1 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type      fixedValue;
26         value      uniform 0;
27     }
28     outlet
29     {

```

```

30     type          fixedValue;
31     value         uniform 0;
32 }
33 alv1
34 {
35     type          nutUSpaldingWallFunction;
36     value         uniform 0;
37 }
38 alv2
39 {
40     type          nutUSpaldingWallFunction;
41     value         uniform 0;
42 }
43 alv3
44 {
45     type          nutUSpaldingWallFunction;
46     value         uniform 0;
47 }
48 walls
49 {
50     type          nutUSpaldingWallFunction;
51     value         uniform 0;
52 }
53 frontAndBack
54 {
55     type          empty;
56 }
57 defaultFaces
58 {
59     type          zeroGradient;
60 }
61 }
62
63 // ***** //

```

/nuTilda

```

1  /*-----*-- C++ -----*/
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*--*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;

```



```

62
63 // ***** //

                                /p

1 /*-----*-- C++ -*-----*\
2 / ===== /
3 / \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ / Operation / Version: 4.1 /
5 / \ / And / Web: www.OpenFOAM.org /
6 / \ / Manipulation /
7 /*-----*--*/
8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14 }
15 // ***** //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField    uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type      zeroGradient;
26     }
27     outlet
28     {
29         type      fixedValue;
30         value      uniform 0;
31     }
32     walls
33     {
34         type      zeroGradient;
35     }
36     alv1
37     {
38         type      zeroGradient;
39     }
40     alv2
41     {
42         type      zeroGradient;
43     }
44     alv3

```

```

45     {
46         type          zeroGradient;
47     }
48     frontAndBack
49     {
50         type          empty;
51     }
52     defaultFaces
53     {
54         type          zeroGradient;
55     }
56 }
57
58 // ***** //

```

/U

```

1  /*-----*-- C++ *-----*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format       ascii;
12     class        volVectorField;
13     object       U;
14 }
15 // * * * * * //
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23     inlet
24     {
25         type      fixedValue;
26         value     uniform (0.0575 0 0);
27     }
28     outlet
29     {
30         type      zeroGradient;
31     }
32     walls

```



```

33     {
34         type            noSlip;
35     }
36     alv1
37     {
38         type            noSlip;
39     }
40     alv2
41     {
42         type            noSlip;
43     }
44     alv3
45     {
46         type            noSlip;
47     }
48     frontAndBack
49     {
50         type            empty;
51     }
52     defaultFaces
53     {
54         type            zeroGradient;
55     }
56 }
57
58 // ***** //

```

## A.4.2 /constant

/transportProperties

```

1  /*-----*-- C++ *-----*\
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1  /
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g  /
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;
15 }
16 // ***** //
17

```

```

18 transportModel Newtonian;
19
20 rho [1 -3 0 0 0 0 0] 1;
21
22 nu [0 2 -1 0 0 0 0] 16.69e-6;
23
24 // ***** //

                                /turbulenceProperties

1 /*-----*-- C++ *-----*\
2 / ===== /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / Operation / Version: 4.1 /
5 / \ \ / And / Web: www.OpenFOAM.org /
6 / \ \ / Manipulation /
7 \*-----*--\
8 FoamFile
9 {
10     version 2.0;
11     format ascii;
12     class dictionary;
13     location "constant";
14     object turbulenceProperties;
15 }
16 // ***** //
17
18 simulationType laminar;
19
20 RAS
21 {
22     RASModel SpalartAllmaras;
23
24     turbulence off;
25
26     printCoeffs off;
27 }
28
29 // ***** //

```

### A.4.3 /system

/controlDict

```

1 /*-----*-- C++ *-----*\
2 / ===== /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /

```



```

 4 /  \ \ /  Operation / Version: 4.1 /
 5 /  \ \ /  And / Web: www.OpenFOAM.org /
 6 /  \ \ /  Manipulation / /
 7 \*-----*/
 8 FoamFile
 9 {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        controlDict;
15 }
16 // ***** //
17
18 application      simpleFoam;
19
20 startFrom        latestTime;
21
22 startTime        0;
23
24 stopAt           endTime;
25
26 endTime          10000;
27
28 deltaT           1;
29
30 writeControl     timeStep;
31
32 writeInterval    5000;
33
34 purgeWrite       0;
35
36 writeFormat      ascii;
37
38 writePrecision   6;
39
40 writeCompression off;
41
42 timeFormat       general;
43
44 timePrecision    6;
45
46 runTimeModifiable true;
47
48
49 // ***** //

```

## /fvSchemes

```
1  /*-----*- C++ -*-----*\
2  / ===== /
3  /  \ \ /   F i e l d       / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ /   O p e r a t i o n / Version: 4.1 /
5  /  \ \ /   A n d / Web: www.OpenFOAM.org /
6  /  \ \ /   M a n i p u l a t i o n /
7  \*-----*-
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSchemes;
15 }
16 // * * * * *
17
18 ddtSchemes
19 {
20     default       steadyState;
21 }
22
23 gradSchemes
24 {
25     default       Gauss linear;
26 }
27
28 divSchemes
29 {
30     default       none;
31     div(phi,U)    bounded Gauss linearUpwind grad(U);
32     div(phi,nuTilda) bounded Gauss linearUpwind grad(nuTilda);
33     div((nuEff*dev2(T(grad(U)))) Gauss linear;
34 }
35
36 laplacianSchemes
37 {
38     default       Gauss linear corrected;
39 }
40
41 interpolationSchemes
42 {
43     default       linear;
44 }
45
46 snGradSchemes
47 {
48     default       corrected;
```

```

49 }
50
51 wallDist
52 {
53     method meshWave;
54 }
55
56
57 // ***** //

                                /fvSolution

1  /*-----*-- C++ -----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*--\
8  FoamFile
9  {
10     version 2.0;
11     format ascii;
12     class dictionary;
13     location "system";
14     object fvSolution;
15 }
16 // ***** //
17
18 solvers
19 {
20     p
21     {
22         solver GAMG;
23         tolerance 1e-04;
24         relTol 0.1;
25         smoother GaussSeidel;
26     }
27
28     U
29     {
30         solver smoothSolver;
31         smoother GaussSeidel;
32         nSweeps 2;
33         tolerance 1e-04;
34         relTol 0.1;
35     }
36
37 nuTilda

```

```

38     {
39         solver          smoothSolver;
40         smoother        GaussSeidel;
41         nSweeps         2;
42         tolerance       1e-04;
43         relTol          0.1;
44     }
45 }
46
47 SIMPLE
48 {
49     nNonOrthogonalCorrectors 0;
50     pRefCell                 0;
51     pRefValue                0;
52
53     residualControl
54     {
55         p                    1e-4;
56         U                    1e-4;
57         nuTilda              1e-4;
58     }
59 }
60
61 relaxationFactors
62 {
63     fields
64     {
65         p                    0.3;
66     }
67     equations
68     {
69         U                    0.7;
70         nuTilda              0.7;
71     }
72 }
73
74
75 // ***** //

```

/refineMeshDict

```

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  /  \ \ /  /  F i e l d           /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  /  O p e r a t i o n       /  V e r s i o n :   4 . 1  /
5  /  \ \ /  /  A n d                   /  W e b :           w w w . O p e n F O A M . o r g  /
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*--
8  FoamFile

```

```

9  {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     object       refineMeshDict;
14 }
15 // * * * * *
16
17 set                fluid;
18
19 coordinateSystem   global;
20
21 globalCoeffs
22 {
23     tan1            (1 0 0);
24     tan2            (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology     true;
34
35 geometricCut       false;
36
37 writeMesh          false;

```

## A.5 Case E

### A.5.1 /0

/p

```

1  /*-----*-- C++ --*-----*/
2  / ===== /
3  / \ \ / \ Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / \ Operation / Version: 4.1 /
5  / \ \ / \ And / Web: www.OpenFOAM.org /
6  / \ \ / \ Manipulation /
7  /*-----*-----*/
8  FoamFile
9  {
10     version      2.0;

```





```

49     {
50         type          empty;
51     }
52     defaultFaces
53     {
54         type          zeroGradient;
55     }
56 }
57
58 // *****

```

## A.5.2 /constant

```

                                     /transportProperties

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*-
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;
15 }
16 // *****
17
18 nu              [0 2 -1 0 0 0 0] 16.69e-6;
19
20
21 // *****

```

## A.5.3 /system

```

                                     /controlDict

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /

```





```

3  /  \  /  Field      / OpenFOAM: The Open Source CFD Toolbox /
4  /  \  /  Operation / Version: 4.1 /
5  /  \  /  And       / Web: www.OpenFOAM.org /
6  /  \  /  Manipulation /
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSchemes;
15 }
16 // ***** //
17
18 ddtSchemes
19 {
20     default      Euler;
21 }
22
23 gradSchemes
24 {
25     default      Gauss linear;
26 }
27
28 divSchemes
29 {
30     default      none;
31     div(phi,U)   Gauss limitedLinearV 1;
32 }
33
34 laplacianSchemes
35 {
36     default      Gauss linear corrected;
37 }
38
39 interpolationSchemes
40 {
41     default      linear;
42 }
43
44 snGradSchemes
45 {
46     default      corrected;
47 }
48
49
50 // ***** //

```

```

                                /fvSolution
1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*- C++ -*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSolution;
15 }
16 // *****
17
18 solvers
19 {
20     p
21     {
22         solver          PCG;
23         preconditioner  DIC;
24         tolerance       1e-03;
25         relTol          0.05;
26     }
27
28     pFinal
29     {
30         $p;
31         relTol          0;
32     }
33
34     U
35     {
36         solver          smoothSolver;
37         smoother        symGaussSeidel;
38         tolerance       1e-03;
39         relTol          0;
40     }
41 }
42
43 PISO
44 {
45     pRefCell 0;
46     pRefValue 0;
47     nCorrectors      2;
48     nNonOrthogonalCorrectors 2;

```

```

49 }
50
51
52 // ***** //

                                /refineMeshDict

1  /*-----*-- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // ***** //
16
17 set                fluid;
18
19 coordinateSystem   global;
20
21 globalCoeffs
22 {
23     tan1            (1 0 0);
24     tan2            (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology     true;
34
35 geometricCut       false;
36
37 writeMesh          false;

```

## A.6 Case F

### A.6.1 /0

/p

```
1  /*----- C++ -----*/
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14 }
15 // * * * * *
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type      zeroGradient;
26     }
27     outlet
28     {
29         type      fixedValue;
30         value     uniform 0;
31     }
32     walls
33     {
34         type      zeroGradient;
35     }
36     alv1
37     {
38         type      zeroGradient;
39     }
40     alv2
41     {
42         type      zeroGradient;
```

```

43     }
44     alv3
45     {
46         type        zeroGradient;
47     }
48     defaultFaces
49     {
50         type        zeroGradient;
51     }
52     frontAndBack
53     {
54         type        empty;
55     }
56 }
57
58 // ***** //

```

/U

```

1  /*-----*-- C++ --*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*--*/
8  FoamFile
9  {
10     version    2.0;
11     format     ascii;
12     class      volVectorField;
13     object     U;
14 }
15 // ***** //
16
17 dimensions    [0 1 -1 0 0 0 0];
18
19 internalField uniform (0 0 0);
20
21 boundaryField
22 {
23     inlet
24     {
25         type            uniformFixedValue;
26         uniformValue    sine;
27         uniformValueCoeffs
28         {
29             frequency    100;
30             amplitude     0.0575;

```

```

31     scale          (1 0 0);
32     level          (0 0 0);
33   }
34 }
35 outlet
36 {
37     type          zeroGradient;
38 }
39 walls
40 {
41     type          noSlip;
42 }
43 alv1
44 {
45     type          noSlip;
46 }
47 alv2
48 {
49     type          noSlip;
50 }
51 alv3
52 {
53     type          noSlip;
54 }
55 frontAndBack
56 {
57     type          empty;
58 }
59 defaultFaces
60 {
61     type          zeroGradient;
62 }
63 }
64
65 // ***** //

```

## A.6.2 /constant

/transportProperties

```

1  /*-----*- C++ -*------*\
2  / ===== /
3  /  \ \ /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  O p e r a t i o n  /  V e r s i o n :  4 . 1  /
5  /  \ \ /  A n d      /  W e b :      w w w . O p e n F O A M . o r g  /
6  /  \ \ /  M a n i p u l a t i o n  /
7  \*-----*\
8  FoamFile

```





```

31
32 writeInterval 1000;
33
34 purgeWrite 0;
35
36 writeFormat ascii;
37
38 writePrecision 6;
39
40 writeCompression off;
41
42 timeFormat general;
43
44 timePrecision 6;
45
46 runTimeModifiable true;
47
48
49 // ***** //

```

#### /fvSchemes

```

1  /*-----*- C++ -*-----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  \*-----*\
8  FoamFile
9  {
10     version 2.0;
11     format ascii;
12     class dictionary;
13     location "system";
14     object fvSchemes;
15 }
16 // ***** //
17
18 ddtSchemes
19 {
20     default Euler;
21 }
22
23 gradSchemes
24 {
25     default Gauss linear;
26 }
27

```

```

28 divSchemes
29 {
30     default          none;
31     div(phi,U)      Gauss limitedLinearV 1;
32 }
33
34 laplacianSchemes
35 {
36     default          Gauss linear corrected;
37 }
38
39 interpolationSchemes
40 {
41     default          linear;
42 }
43
44 snGradSchemes
45 {
46     default          corrected;
47 }
48
49
50 // ***** //

                                /fvSolution

1  /*----- C++ -----*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /
7  \*-----*/
8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSolution;
15 }
16 // ***** //
17
18 solvers
19 {
20     p
21     {
22         solver      PCG;
23         preconditioner DIC;

```

```

24     tolerance      1e-03;
25     relTol         0.05;
26 }
27
28 pFinal
29 {
30     $p;
31     relTol          0;
32 }
33
34 U
35 {
36     solver           smoothSolver;
37     smoother         symGaussSeidel;
38     tolerance        1e-03;
39     relTol           0;
40 }
41 }
42
43 PISO
44 {
45     pRefCell 0;
46     pRefValue 0;
47     nCorrectors 2;
48     nNonOrthogonalCorrectors 2;
49 }
50
51
52 // ***** //

```

/refineMeshDict

```

1  /*-----*- C++ -*-----*\
2  / ===== / /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation / /
7  \*-----*- //
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // ***** //
16
17 set          fluid;

```

```

18
19 coordinateSystem      global;
20
21 globalCoeffs
22 {
23     tan1                (1 0 0);
24     tan2                (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology         true;
34
35 geometricCut           false;
36
37 writeMesh              false;

```

## A.7 Case G

### A.7.1 /0

/p

```

1  /*-----*- C++ -*-----*/
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /
7  /*-----*-
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volScalarField;
13     object        p;
14 }
15 // * * * * * //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField    uniform 0;

```

```

20
21 boundaryField
22 {
23     inletOut
24     {
25         type            inletOutlet;
26         //type          outletInlet;
27         //inletValue    uniform 0.01;
28         //value         0;
29         inletValue     $internalField;
30         value          $internalField;
31     }
32     walls
33     {
34         type            zeroGradient;
35     }
36     alv1
37     {
38         type            zeroGradient;
39     }
40     alv2
41     {
42         type            zeroGradient;
43     }
44     alv3
45     {
46         type            zeroGradient;
47     }
48     frontAndBack
49     {
50         type            empty;
51     }
52     defaultFaces
53     {
54         type            zeroGradient;
55     }
56 }
57
58 // ***** //

```

/U

```

1  /*-----*-- C++ *-----*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /

```

```

7  /*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         volVectorField;
13     object        U;
14 }
15 // * * * * *
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23     inOut
24     {
25         type      pressureInletOutletVelocity;
26         value     uniform (0 0 0);
27     }
28     walls
29     {
30         type      fixedValue;
31         value     uniform (0 0 0);
32     }
33     alv1
34     {
35         type      movingWallVelocity;
36         value     uniform (0 0 0);
37     }
38     alv2
39     {
40         type      movingWallVelocity;
41         value     uniform (0 0 0);
42     }
43     alv3
44     {
45         type      movingWallVelocity;
46         value     uniform (0 0 0);
47     }
48     frontAndBack
49     {
50         type      empty;
51     }
52     defaultFaces
53     {
54         type      zeroGradient;
55     }

```

```

56 }
57
58 // ***** //

                                /pointDisplacement

1  /*-----*-- C++ -----*\
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / A nd / Web: www.OpenFOAM.org /
6  / \ \ / M anipulation /
7  \*-----*--\
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         pointVectorField;
13     object        pointDisplacement;
14 }
15 // ***** //
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23     alv1
24     {
25         type      oscillatingDisplacement;
26         value     uniform (0 0 0);
27         amplitude (0 -0.0000113 0);
28         omega     62.8318;
29     }
30     alv2
31     {
32         type      oscillatingDisplacement;
33         value     uniform (0 0 0);
34         amplitude (0.0000113 0 0);
35         omega     62.8318;
36     }
37     alv3
38     {
39         type      oscillatingDisplacement;
40         value     uniform (0 0 0);
41         amplitude (0 0.0000113 0);
42         omega     62.8318;
43     }

```

```

44     wall
45     {
46         type        fixedValue;
47         value        uniform (0 0 0);
48     }
49     inOut
50     {
51         type        fixedValue;
52         value        uniform (0 0 0);
53     }
54     frontAndBack
55     {
56         type        fixedValue;
57         value        uniform (0 0 0);
58     }
59     defaultFaces
60     {
61         type        zeroGradient;
62     }
63 }
64
65 // ***** //

```

## A.7.2 /constant

```

                                /dynamicMeshDict

1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / Operation / Version: 4.1 /
5  / \ \ / And / Web: www.OpenFOAM.org /
6  / \ \ / Manipulation /
7  /*-----*-
8  FoamFile
9  {
10     version    2.0;
11     format      ascii;
12     class       dictionary;
13     location    "constant";
14     object      dynamicMeshDict;
15 }
16 // ***** //
17
18 dynamicFvMesh    dynamicMotionSolverFvMesh;
19
20 motionSolverLibs ( "libfvMotionSolvers.so" );
21 //solver         velocityLaplacian;

```



```

22 //velocityLaplacianCoeffs
23
24 solver          displacementLaplacian;
25 displacementLaplacianCoeffs
26 {
27     diffusivity quadratic inverseDistance (alv1);
28     diffusivity quadratic inverseDistance (alv2);
29     diffusivity quadratic inverseDistance (alv3);
30 }

                                /transportProperties

1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ \ / \ Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / \ Operation / Version: 4.1 /
5  / \ \ / \ And / Web: www.OpenFOAM.org /
6  / \ \ / \ Manipulation /
7  /*-----*- C++ -*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;
15 }
16 // ***** //
17
18 transportModel Newtonian;
19
20 nu              [0 2 -1 0 0 0 0] 16.69e-6;
21
22 // ***** //

```

```

                                /turbulenceProperties

1  /*-----*- C++ -*-----*/
2  / ===== /
3  / \ \ / \ Field / OpenFOAM: The Open Source CFD Toolbox /
4  / \ \ / \ Operation / Version: 4.1 /
5  / \ \ / \ And / Web: www.OpenFOAM.org /
6  / \ \ / \ Manipulation /
7  /*-----*- C++ -*-----*/
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;

```

```

12     class      dictionary;
13     location   "constant";
14     object     turbulenceProperties;
15 }
16 // * * * * *
17
18 simulationType laminar;
19
20 // * * * * *

```

### A.7.3 /system

/controlDict

```

1 /*----- C++ -----*\
2 / ===== / /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / Operation / Version: 4.1 /
5 / \ \ / And / Web: www.OpenFOAM.org /
6 / \ \ / Manipulation / /
7 /*-----*\
8 FoamFile
9 {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     location     "system";
14     object       controlDict;
15 }
16 // * * * * *
17
18 application     pimpleDyMFoam;
19
20 startFrom       latestTime;
21
22 startTime       0;
23
24 stopAt          endTime;
25
26 endTime         0.025;
27
28 deltaT          1e-7;
29
30 writeControl    timeStep;
31
32 writeInterval   1000;
33
34 purgeWrite      0;

```

```

35
36 writeFormat      ascii;
37
38 writePrecision   6;
39
40 writeCompression off;
41
42 timeFormat       general;
43
44 timePrecision    6;
45
46 runTimeModifiable true;
47
48 adjustTimeStep  no;
49
50 maxCo           0.2;
51
52 // ***** //

```

/fvSchemes

```

1  /*-----*- C++ -*------*\
2  / ===== /
3  /  \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4  /  \ \ / Operation / Version: 4.1 /
5  /  \ \ / And / Web: www.OpenFOAM.org /
6  /  \ \ / Manipulation /
7  \*-----*-
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "system";
14     object        fvSchemes;
15 }
16 // ***** //
17
18 ddtSchemes
19 {
20     default      Euler;
21 }
22
23 gradSchemes
24 {
25     default      Gauss linear;
26 }
27
28 divSchemes

```

```

29 {
30     default          none;
31
32     div(phi,U)       Gauss linear;
33     div((nuEff*dev2(T(grad(U)))) Gauss linear;
34 }
35
36 laplacianSchemes
37 {
38     default          Gauss linear corrected;
39
40     laplacian(diffusivity,cellDisplacement/*cellMotionU*/) Gauss linear uncorrected;
41 }
42
43 interpolationSchemes
44 {
45     default          linear;
46 }
47
48 snGradSchemes
49 {
50     default          corrected;
51 }
52
53
54 // ***** //

```

#### /fvSolution

```

1 /*----- C++ -----*\
2 / ===== /
3 / \ \ / Field / OpenFOAM: The Open Source CFD Toolbox /
4 / \ \ / Operation / Version: 4.1 /
5 / \ \ / And / Web: www.OpenFOAM.org /
6 / \ \ / Manipulation /
7 \*-----*/
8 FoamFile
9 {
10     version      2.0;
11     format       ascii;
12     class        dictionary;
13     location     "system";
14     object       fvSolution;
15 }
16 // ***** //
17
18 solvers
19 {
20

```

```

21 p
22 {
23     solver          GAMG;
24     tolerance       0;
25     relTol          0.01;
26     smoother        GaussSeidel;
27     cacheAgglomeration no;
28 }
29
30 pFinal
31 {
32     $p;
33     tolerance       1e-03;
34     relTol          0;
35 }
36
37 pcorr
38 {
39     $p
40     tolerance       1e-3;
41     relTol          0;
42 }
43
44 U
45 {
46     solver          smoothSolver;
47     smoother        symGaussSeidel;
48     tolerance       1e-03;
49     relTol          0.1;
50 }
51
52 UFinal
53 {
54     $U;
55     tolerance       1e-03;
56     relTol          0;
57 }
58
59 cellDisplacement
60 {
61     solver          PCG;
62     preconditioner  DIC;
63     tolerance       1e-03;
64     relTol          0;
65 }
66 /*
67 cellMotionU
68 {
69     solver          PCG;

```

```

70     preconditioner    DIC;
71     tolerance         1e-03;
72     relTol            0;
73 }
74 */
75 }
76
77 PIMPLE
78 {
79     correctPhi         yes;
80     nOuterCorrectors  2;
81     nCorrectors        2;
82     nNonOrthogonalCorrectors 2;
83     pRefCell           0;
84     pRefValue          0;
85 }
86
87 relaxationFactors
88 {
89     equations
90     {
91         "U.*"          1;
92     }
93 }
94
95
96 // ***** //

```

/refineMeshDict

```

1  /*-----*-- C++ --*-----*/
2  / ===== /
3  /  \ \ /  /  F i e l d      /  O p e n F O A M :  T h e  O p e n  S o u r c e  C F D  T o o l b o x  /
4  /  \ \ /  /  O p e r a t i o n  /  V e r s i o n :  4 . 1  /
5  /  \ \ /  /  A n d      /  W e b :      w w w . O p e n F O A M . o r g  /
6  /  \ \ /  /  M a n i p u l a t i o n  /
7  \*-----*--
8  FoamFile
9  {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     object        refineMeshDict;
14 }
15 // ***** //
16
17 set                      fluid;
18
19 coordinateSystem         global;

```

```
20
21 globalCoeffs
22 {
23     tan1           (1 0 0);
24     tan2           (0 1 0);
25 }
26
27 directions
28 (
29     tan1
30     tan2
31 );
32
33 useHexTopology     true;
34
35 geometricCut       false;
36
37 writeMesh          false;
```

# Appendix B

## Gmsh Code

This chapter includes the .geo files written for the Duct, Airway-1, and Airway-2 Models.

### B.1 Duct

```
1  p = 0.58;
2  DD = 2*0.12;
3  sac = 0.82;
4  d1 = DD*0.25;
5  d2 = DD*0.75;
6
7  Point(1) = {0, 0, 0, 0};
8  Point(2) = {p, 0, 0, 0};
9  Point(3) = {p, DD, 0, 0};
10 Point(4) = {0, DD, 0, 0};
11
12 Line(1) = {1, 2};
13 Line(2) = {2, 3};
14 Line(3) = {3, 4};
15 Line(4) = {4, 1};
16
17 Transfinite Line {1, 3} = 20 Using Progression 1;
18 Transfinite Line {2, 4} = 10 Using Progression 1;
19 Line Loop(10) = {1, 2, 3, 4};
20
21 Plane Surface(11) = {10};
22
23 Transfinite Surface {11} = {1, 2, 3, 4};
24
```



```

25 Recombine Surface{11};
26
27 Extrude {0, 0, .01} {
28     Surface{11};
29     Layers{1};
30     Recombine;
31 }
32
33 Transfinite Volume{1} = {1,2,3,4,5,6,10,14};
34
35 Physical Surface("inlet") = {32};
36 Physical Surface("outlet") = {24};
37 Physical Surface("walls") = {20,28};
38 Physical Surface("frontAndBack") = {11,33};
39 Physical Volume("fluid") = {1};

```

## B.2 Airway-1

```

1  p = 0.58;
2  DD = 2*0.12;
3  sac = 0.82;
4  d1 = DD*0.25;
5  d2 = DD*0.75;
6
7  Point(1) = {0, 0, 0, 0};
8  Point(2) = {p, 0, 0, 0};
9  Point(3) = {p, DD, 0, 0};
10 Point(4) = {0, DD, 0, 0};
11 Point(6) = {0.67, 0.12, 0, 0};
12 Point(7) = {0.76,0,0,0};
13 Point(8) = {0.76,DD,0,0};
14 Point(9) = {0,d2,0,0};
15 Point(10) = {0, d1, 0, 0};
16
17 Line(1) = {1, 2};
18
19 Circle(2) = {2, 6, 7};
20 Circle(3) = {7, 6, 8};
21 Circle(4) = {8,6,3};
22 Circle(5) = {3, 6, 2};
23
24 Line(6) = {3, 4};
25 Line(7) = {4, 9};
26 Line(8) = {9,10};
27 Line(9) = {10,1};
28
29 Transfinite Line {1, 6} = 20 Using Progression 1;

```

```

30 Transfinite Line {5, 2, 3, 4} = 18 Using Progression 1;
31 Transfinite Line {8} = 10 Using Progression 1;
32 Transfinite Line {7, 9} = 5 Using Progression 1;
33
34 Line Loop(10) = {1, -5, 6, 7, 8, 9};
35 Plane Surface(11) = {10};
36
37 Line Loop(12) = {5, 2, 3, 4};
38 Plane Surface(13) = {12};
39
40 Transfinite Surface {11} = {1, 2, 3, 4};
41 Transfinite Surface {13} = {2, 7, 8, 3};
42
43 Recombine Surface{11, 13};
44
45 Extrude {0, 0, .01} {
46     Surface{11, 13};
47     Layers{1};
48     Recombine;
49 }
50
51 Transfinite Volume{1} = {1, 2, 3, 4, 11, 12, 17, 21};
52 Transfinite Volume{2} = {2, 7, 8, 3, 12, 37, 42, 17};
53
54 Physical Surface("inlet") = {40};
55 Physical Surface("outlet") = {36,44};
56 Physical Surface("walls") = {24,32};
57 Physical Surface("alv1") = {58};
58 Physical Surface("alv2") = {62};
59 Physical Surface("alv3") = {66};
60 Physical Surface("frontAndBack") = {11,45};
61 Physical Volume("fluid") = {1,2};

```

## B.3 Airway-2

```

1 p = 0.58;
2 DD = 2*0.12;
3 sac = 0.82;
4
5 Point(1) = {0, 0, 0, 0};
6 Point(2) = {p, 0, 0, 0};
7 Point(3) = {p, DD, 0, 0};
8 Point(4) = {0, DD, 0, 0};
9 Point(6) = {0.67, 0.12, 0, 0};
10 Point(7) = {0.76,0,0,0};
11 Point(8) = {0.76,DD,0,0};
12

```

```

13 Line(1) = {1, 2};
14
15 Circle(2) = {2, 6, 7};
16 Circle(3) = {7, 6, 8};
17 Circle(4) = {8,6,3};
18 Circle(5) = {3, 6, 2};
19
20 Line(6) = {3, 4};
21 Line(7) = {4, 1};
22
23 Transfinite Line {1, 6} = 10 Using Progression 1;
24 Transfinite Line {7} = 10 Using Progression 1;
25 Transfinite Line {5, 2, 3, 4} = 10 Using Progression 1;
26
27 Line Loop(8) = {7, 1, -5, 6};
28 Plane Surface(9) = {8};
29
30 Line Loop(10) = {5, 2, 3, 4};
31 Ruled Surface(11) = {10};
32
33 Transfinite Surface {9} = {1, 2, 3, 4};
34 Transfinite Surface {11} = {2, 7, 8, 3};
35
36 Recombine Surface {9,11};
37
38 Extrude {0, 0, 0.01} {
39     Surface{9, 11};
40     Layers{1};
41     Recombine;
42 }
43
44 Transfinite Volume{1} = {1, 2, 3, 4, 10, 14, 19, 9};
45 Transfinite Volume{2} = {2, 7, 8, 3, 14, 27, 32, 19};
46
47 Recombine Volume {1,2};
48
49 Physical Surface("inOut") = {20};
50 Physical Surface("walls") = {24,32};
51 Physical Surface("alv1") = {46};
52 Physical Surface("alv2") = {50};
53 Physical Surface("alv3") = {54};
54 Physical Surface("frontAndBack") = {9,33};
55 Physical Volume("fluid") = {1,2};

```

## Vita

Jeremy Myers was born in 1986 at the Portsmouth Naval Hospital in Portsmouth, Virginia. He was very interested in science as a youngster, fascinated by space exploration and the cosmos, attending various science camps throughout his early teens.

In high school, he showed an aptitude for foreign languages and pursued this path as an undergraduate at James Madison University, where he graduated in 2009 with a B.A. in International Affairs and minors in Russian Studies and Economics.

He returned to his childhood science and enrolled at Virginia Commonwealth University in 2011. He became a full-time student in 2013 in the Mathematics Department. During a study abroad opportunity with the "Math in Moscow" program, Jeremy decided to pursue his Master's degree in mathematics at VCU and to focus in computational mathematics.

In the fall of 2017, Jeremy began Ph.D studies in Computer Science at the College of William and Mary.