

Claremont Colleges

Scholarship @ Claremont

KGI Theses and Dissertations

KGI Student Scholarship

Winter 12-18-2020

Modeling Residence Time Distribution of Chromatographic Perfusion Resin for Large Biopharmaceutical Molecules: A Computational Fluid Dynamic Study

Kevin Vehar

Follow this and additional works at: https://scholarship.claremont.edu/kgi__theses



Part of the [Biotechnology Commons](#), [Complex Fluids Commons](#), [Fluid Dynamics Commons](#), [Geometry and Topology Commons](#), and the [Other Biomedical Engineering and Bioengineering Commons](#)

Recommended Citation

Vehar, Kevin. (2020). *Modeling Residence Time Distribution of Chromatographic Perfusion Resin for Large Biopharmaceutical Molecules: A Computational Fluid Dynamic Study*. KGI Theses and Dissertations, 18. https://scholarship.claremont.edu/kgi__theses/18.

This Restricted to Claremont Colleges Dissertation is brought to you for free and open access by the KGI Student Scholarship at Scholarship @ Claremont. It has been accepted for inclusion in KGI Theses and Dissertations by an authorized administrator of Scholarship @ Claremont. For more information, please contact scholarship@cuc.claremont.edu.

Modeling Residence Time Distribution of Chromatographic Perfusion Resin For Large Biopharmaceutical Molecules: A Computational Fluid Dynamic Study

Author:

Kevin VE HAR

Supervisor:

Dr. Cameron BARDLIVING

*A Dissertation submitted to the Faculty of Keck Graduate Institute in
partial fulfillment of the requirements for the degree of Doctor of Philosophy
in Applied Life Sciences*

Claremont, California

2020





OFFICE OF THE REGISTRAR

PhD Dissertation Completion Form

We, the undersigned, certify that we have read this dissertation of Kevin Vehar, 97202357
Name and ID#
and approve it as adequate in scope and quality for the degree of Doctor of Philosophy.

Dissertation Committee:

<u>Cameron Bardliving, PhD</u> (Typed name of Chair), Chair	DocuSigned by: <u>Cameron Bardliving, PhD</u> E38667F2EF0E403...
<u>Parviz Shamlou, PhD</u> (Typed name), Member	DocuSigned by: <u>Parviz Shamlou, PhD</u> 79769B036B4D44C...
<u>Hu Zhang, PhD</u> (Typed name), Member	DocuSigned by: <u>Hu Zhang, PhD</u> 568A164F5A834C2...
<u>Erno Pungor, PhD</u> (Typed name), Visiting Examiner	DocuSigned by: <u>Erno Pungor, PhD</u> C07600B3EB5F487...
<u>Jim Sterling, PhD</u> (Typed name), PhD Program Director	DocuSigned by: <u>Jim Sterling, PhD</u> 2809379DF9994D0...

Abstract

*Modeling Residence Time Distribution of
Chromatographic Perfusion Resin For Large
Biopharmaceutical Molecules: A Computational Fluid
Dynamic Study*

by Kevin VE HAR

The need for production processes of large biotherapeutic particles, such as virus-based particles and extracellular vesicles, has risen due to increased demand in the development of vaccinations, gene therapies, and cancer treatments. Liquid chromatography plays a significant role in the purification process and is routinely used with therapeutic protein production. However, performance with larger macromolecules is often inconsistent, and parameter estimation for process development can be extremely time- and resource-intensive. This thesis aimed to utilize advances in computational fluid dynamic (CFD) modeling to generate a first-principle model of the chromatographic process while minimizing model parameter estimation's physical resource demand. Specifically, I utilized explicit geometric rendering to develop a CFD steady-state model to simulate fluid flow through and around a perfusive porous resin in a pseudo packed bed flow-cell to predicted fluid velocities and shear stress. I generated different explicit geometries, and

compared the velocity profiles of steady-state simulations against reported literature values of commercially available resin's intraparticle convective flow. I then developed a two-part transient CFD discrete phase model to model a tracer protein's capture and release from a resin. Particle age distribution functions were calculated to characterize the macromixing in the model and compared them with existing single parameter models. These models exhibited similar distribution profiles and provided additional information about the shear forces acting on the particles. These preliminary studies revealed that shear is relatively low shear at process operating conditions, and the low yield of large biotherapeutic particles in chromatography is likely not due to shear forces.

Acknowledgments

I want to thank my PI, Dr. Cameron Bardliving, for guiding and motivating me throughout this journey. As both a mentor and a friend, our talks got me through many challenging times during my PhD. I would also like to thank Dr. Parviz Shamlou, Dr. Hu Zhang, and Dr. Erno Pungor for their constant support and guidance as my doctoral committee members.

I would also like to thank all the incredibly supportive people I've met over the years at KGI. Lynn Svay played a key role during my time at KGI and was indeed one of the best lab instructors, mentors, and friends anyone could have asked for, and I wouldn't be where I am today if it wasn't for her. I would also like to thank Dr. Stephanie Parker for her guidance and good humor. Additionally, I would like to thank Dr. Sue Behrens for her continued support during my final year. I am also very grateful for my peers and friends that I have made throughout my time at KGI: Dr. Andrew Burns, Dr. Corinna Doris, Dr. Flaka Radoniqi, Aster Escalante, Payam Amiri, Christine Urrea, and David Kent.

Finally, I would like to thank my family for their love and support in all aspects of my life. My parents, Gordon and Janet, have been incredibly supportive forces in my life. Despite being dyslexic, they always believed in me and never let my dyslexia be a crutch. The fact that I can read, let alone wrote this thesis, is a testament to them. I'd

also like to thank my younger sister, Julia, for her love and support. And finally, I would like to thank Jack for his constant love, support, and patience despite the vague nature of scientific research. I would not have been able to complete this without him.

Contents

Abstract	ii
Acknowledgments	iv
Contents	vi
List of Figures	x
List of Tables	xvi
1 Introduction	1
1.1 Biotherapeutic Particles	1
1.2 Manufacturing Process	3
1.3 This Work	6
2 Background/Literature Review	9
2.1 Gene Therapy	9
2.1.1 Types of Viral Vectors	11
2.1.2 Manufacturing Process Overview	13

2.2	Physics of Flow and Traditional Definitions	15
2.2.1	Packed Beds and Packing Regimes	15
2.2.2	Fluid Flow Definitions	16
2.2.3	Aspect Ratio	17
2.2.4	Wall Regions	19
2.2.5	Empirical Equations and Models	21
2.2.5.1	Distributions of Residence Times	22
2.3	Computational Fluid Dynamics (CFD)	25
2.3.1	Computational methods	25
2.3.2	Meshing and Contact Point Modifications	26
2.3.3	Porous Media Rendering	28
2.4	Discrete Phase Modeling Equations	29
2.4.1	Saffman Lift Force	31
2.4.2	Stokes-Cunningham Drag Law and Brownian Motion	31
2.4.3	Brownian Random Force	32
2.4.4	Particle Tracking with Eulerian-Lagrangian Method	33
2.4.5	Coupling between continuous and discrete phases	33
2.4.6	Shear Stress Integral	34
3	Methodology	37
3.1	Packed Bed Generation	37
3.1.1	Geometry Software: Blender	38
3.1.2	Physics Engine and Rigid Body Dynamics	38
3.1.3	Blender	43

3.1.4	Python Scripts	44
3.1.4.1	Packed Bed Generator Script	44
3.1.4.2	Contact Point Modifier Script	46
3.1.5	CFD Modeling of Packed Bed	48
3.2	Porosity and Explicit Geometry Rendering	52
3.2.1	Case Setup for Different Porosity Settings (Geometry)	57
3.2.2	Meshing	59
3.2.3	CFD Setup	60
3.3	DPM Flow Cell	61
3.3.1	Geometry	61
3.3.2	Mesh	62
3.3.3	CFD Setup	63
4	Results and Discussion	69
4.1	Steady State Models	69
4.1.1	Pore Lattice Exploration	83
4.2	DPM Models	88
4.2.1	Capture Simulation	89
4.2.2	Release Simulation	106
5	Conclusions	116
A	References for Figure 1.2	118
B	Contact Modification Python Script	128

C UDF Codes	135
C.1 Interpolate	135
C.2 Shear Integral UDF	136
C.3 Monitor Points in Fluent UDF	141
Bibliography	149

List of Figures

1.1	Historical data on vectors used in gene therapy clinical trials adapted from (Ginn et al., 2018)	3
1.2	Chromatography yields for viral vectors as reported in literature	5
2.1	Viral vector structure, (Taylor, 2010)	12
2.2	Virus purification for large-scale operations, adapted from Merten et al. (2014)	14
2.3	Two different aspect ratios	18
2.4	Radial porosity variance, (De Klerk, 2003)	20
2.5	Computational fluid dynamics process work flow.	26
2.6	Diagram of the types of contact point modification methods: (a) Gaps; (b) Overlaps; (c) Caps and (d) Bridges; adapted from Dixon, Nijemeisland, and Stitt (2013)	28
2.7	Integration using the Trapezoidal Rule	35
3.1	Overview of Bullet Physics engine simulation loop for Rigid Body Dynamics (RBD) adapted from Coumans (2015).	40

3.2	Contact modification based on proximity. The top picture shows the source object is colored by proximity, red indicates close proximity to the target, blue indicates low face proximity to the target object. The bottom shows the applied simple deformation on the source object with the previous proximity coloring.	49
3.3	Results of bed packings for different beads shapes with different ratios of bed to particle diameters.	51
3.4	Depiction of the thickness and length lattice parameters for the Shell tool at different possible slices in the lattice. The blue shaded area represents the solid porous structure, while the white space represents the fluid void space.	56
3.5	Flow cell geometry	57
4.1	Shear stress distribution of mesh. (a) shows location of two volume probes, red corresponds to the BOI resin volume and blue corresponds to the packed bed region bounded to the , colors correspond to chart line colors in (b-h).	71
4.2	Velocity Vectors in YZ-plane at $X = 0$	73
4.3	Close up view of velocity vectors form Figure 4.2 YZ-plane at $X = 0$	74
4.4	Closer view of velocity vectors between beads form Figure 4.2 YZ-plane at $X = 0$	75
4.5	Velocity vectors with contours of the velocity magnitude form Figure 4.2 YZ-plane at $X = 0$	76
4.6	Closer view of velocity vectors between beads along with the contours of the velocity magnitude on YZ-plane at $X = 0$	77

4.7	Volume rendering of shear stress in the fluid domain for steady-state simulations with inlet velocities of 50, 100, 150, 200, 250, 300 and 1000 cm h ⁻¹ .	78
4.8	Contours of shear stress on YZ-plane at X = 0 for steady-state simulations with inlet velocities of 50, 100, 150, 200, 250, 300 and 1000 cm h ⁻¹ .	79
4.9	Contours of wall shear on the surrounding beads as well as on the explicit geometry of the Bead Of Interest (BOI).	80
4.10	Closeup of contours of wall shear stress on explicit geometry of the BOI with velocity vectors in black.	81
4.11	Sliced view of contours of wall shear stress on explicit geometry of the BOI with velocity vectors in black.	82
4.12	The relationship of the cell Péclet number distribution for IgG vs the radial coordinate of the BOI resin for pore lattice structures of 3.5 μm, 3.0 μm, 2.0 μm and 1.5 μm. The diffusion coefficient used was for IgG protein in dilute solution at 25 °C where $D = 40 \mu\text{m}^2 \text{s}^{-1}$.	85
4.13	The relationship of the AAV cell Péclet number distribution vs the radial coordinate of the BOI resin for pore lattice structures of 3.5 μm, 3.0 μm, 2.0 μm and 1.5 μm with an inlet velocity of 1000 cm h ⁻¹ . (a) The diffusion coefficient used was based on the Stokes Einstein equation $D = 11 \mu\text{m}^2 \text{s}^{-1}$ for AAV assuming a spherical particle with radius of 13 nm. (b) The diffusion coefficient used was based on the Seisenberger et al. (2001) observed diffusion coefficient $D = 7.5 \mu\text{m}^2 \text{s}^{-1}$.	87
4.14	Distribution of particle residence time for the capture simulation trapped on BOI rendered as (a) a histogram and (b) a boxplot.	91

4.15	Histograms of Residence Time Distribution of Discrete Phase Model (DPM) simulations separated by particle fate location (i.e., particles captured on resin or particles that flowed through to outlet).	92
4.16	Age distribution functions of DPM particles captured on the BOI resin during the the capture simulation at 5 inlet velocities. (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.(b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.	94
4.17	Age distribution functions of DPM particles that escaped through the outlet during the capture simulation at 5 inlet velocities. (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.(b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.	95
4.18	Relationship of the mean residence time (t_m) of outlet particles with volumetric flow rate (Q). The black error bars represent the variance of the residence time distribution.	96
4.19	The results of the tanks in series RTD model (solid orange curve) fit to the particles trapped at the resin during the capture DPM simulations (solid blue curve) at various inlet velocities. The blue vertical dash line indicates the mean residence time of the DPM simulation; the orange vertical dash line indicates mean residence time of the tanks in series RTD model. . . .	99

4.20	The results of the tanks in series RTD model (solid orange curve) fit to the particles that escape through the outlet during the capture DPM simulations (solid blue curve) at various inlet velocities. The blue vertical dash line indicates the mean residence time of the DPM simulation; the orange vertical dash line indicates mean residence time of the tanks in series RTD model.	100
4.21	Kernel Density Estimate of Residence Time Distribution of DPM simulations separated by particle fate location. (i.e., particles captured on resin or particles that flowed through to outlet).	101
4.22	Percent of trapped particles captured on BOI categorized by inner and outer regions (dark blue and light blue respectively).	102
4.23	Boxplot showing the Residence Time Distribution of particles separated out by surface.	103
4.24	Residence Time of particles vs the particle capture location normalized to the particle's radius. Points are colored by the capture surface of the BOI.	104
4.25	(a) Particle Shear Stress Distribution in Terms of Percent Bound to Resin. (b) Particle Shear Stress integral Distribution	105
4.26	Individual chromatograms from Figure 4.27	107
4.27	Stacked release chromatograms, see Table 4.3 for particle statistics	108

4.28	Age distribution functions of DPM particles that escaped through the outlet during the release simulation for 5 inlet velocities; (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.(b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.	109
4.29	Contours of the Z-velocity of in an XY-plane cross section through the center of the BOI. Negative velocity values point towards the outlet. . . .	112
4.30	Distribution of the cell Péclet number within a 50 micron radius of the center of the BOI.	113
4.31	Contours of the Péclet number in an XY-plane cross section through the center of the BOI.	114
4.32	Levenspiel and Smith's (1957) axial dispersion model with open-open boundary condition Residence Time Distribution (RTD) fit to the DPM release simulation RTDs using the open-source rtdpy python package (Flamm, 2019).	115

List of Tables

3.1	Different particle shapes for bed packing	50
3.2	Explicit pattern geometries considered for porous rendering. Geometries were generated using ANSYS SpaceClaim shell infill tool except for the Sphere/Ring Clustering, which was generated using Blender.	53
3.3	Different BOI lattice geometries with their corresponding 2D slice and 3D renderings considered for explicit porous geometry rendering.	59
3.4	Temporal parameters used for the transient DPM particle loading simulations for the 5 different inlet velocities.	66
4.1	Mesh Independence	70
4.2	Different BOI lattice geometries with their corresponding 2D slice and 3D renderings considered for explicit porous geometry rendering.	84
4.3	DPM Particle exit statistics for release simulations	106
4.4	Curve fitting results for the release simulation	111

Acronyms

AAV adeno-associated virus. 2, 3, 15, 85, 86

BOI Bead Of Interest. xii, xiv–xvi, 57–64, 67, 69, 70, 72, 80–82, 84, 88–91, 102, 104, 106, 110, 112–114

CFD Computational Fluid Dynamics. 6–8, 25–28, 48, 52, 57, 60, 62, 63, 69, 83, 86, 88, 116, 117

DEM Discrete Element Method. 39

DPM Discrete Phase Model. xiii–xv, 8, 29, 34, 64, 65, 67, 68, 88, 89, 92, 97, 98, 101, 106, 108, 115, 117

KDE Kernel Density Estimation. 90

PBG Packed Bed Generator. 44

RBD Rigid Body Dynamics. x, 39, 40, 43

RTD Residence Time Distribution. xv, 22–24, 90, 93, 97, 98, 108, 115

UDF User Defined Function. 34, 65–67

VLPs Virus-Like Particles. 2, 3

VWP Vertex Weight Proximity. 47, 48

Dedicated to Jack

Chapter 1

Introduction

1.1 Biotherapeutic Particles

Demand for improved production processes of biotherapeutic particles has risen due to increased interest in the development of gene therapies, vaccines, and cancer treatments. These fields have successfully utilized large and complex molecules, such as virus-based particles and extracellular vesicles (EVs), in various medical applications (Ginn et al., 2018; Effio and Hubbuch, 2015). Even though these macromolecules are inherently different particles, they possess similarities in addition to their size range.

EVs are nanometer-sized particles secreted from most cells that act as important mediators of intercellular communications. They are enclosed in a bilipid membrane and often contain lipids, proteins, and various nucleic acids from the source cell that can be transferred and regulate the biological functions of the target cell. These particles are very similar to an enveloped virus in structure, using similar mechanism routes to enter cells by binding to a cell's plasma membrane and entering via fusion or endocytosis (van Dongen et al., 2016; Nolte-'t Hoen et al., 2016).

Vaccination remains one of the most effective ways to prevent viral diseases. It traditionally used either dead, inactivated, or attenuated samples of a virus to train the immune system to recognize and combat the harmful pathogen without being exposed to the disease (Moleirinho et al., 2020; Roldão et al., 2010). Initially, while this method of using “whole” viral particles was successful for a variety of viruses, some of these early vaccines actually caused actual virus outbreaks, most notably with the foot-and-mouth disease virus (FMDV). An alternative vaccination method was developed using Virus-Like Particles (VLPs), a multiprotein structure that conformationally mimics the native virus but lacks the replication genome. While they no longer possess the genetic information required to replicate, VLPs can be safer and cheaper vaccine candidates. However, they still possess the manufacturing challenges faced by these larger biotherapeutic particles (Roldão et al., 2010).

While viruses are used to protect against various infectious diseases, researchers have also developed ways to reprogram viral particles to deliver a therapeutic gene instead of the viral genome into target cells. This type of viral gene therapy treatment has been applied to a variety of clinical applications, from oncolytic treatments to hemophilia (Merten et al., 2014; Ginn et al., 2018). To date, many of the gene therapy clinical trials rely on a variety of viral vectors to deliver the therapeutic gene, which includes adenovirus, retrovirus, lentivirus, adeno-associated virus (AAV), vaccinia virus, herpes simplex virus, and pox virus. Figure 1.1 shows the breakdown of vectors used to date in clinical trials, as reported by Ginn et al. (2018). While there have been several gene therapy clinical trials to date, the majority of products are still in phase I and phase I/II (BioCentury Inc., 2019). This bottleneck of gene therapy products in early phases of clinical trials highlights a need for improvements in high volume clinical-grade vector production.

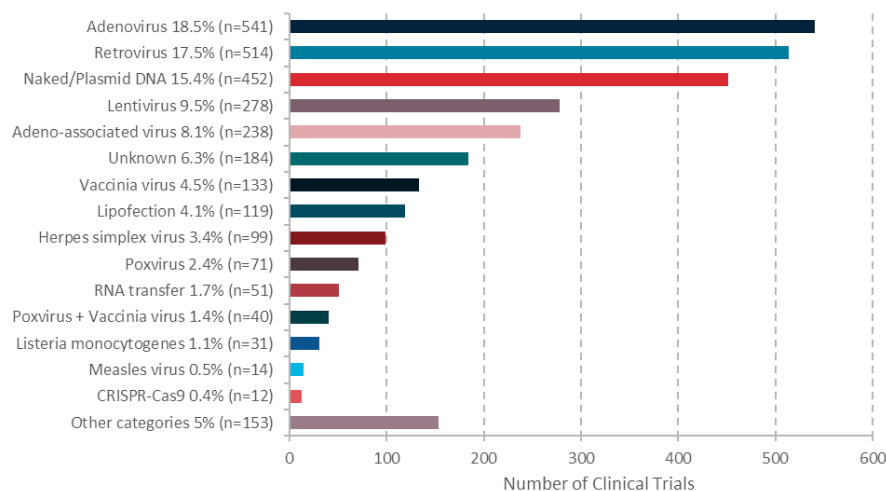


FIGURE 1.1: Historical data on vectors used in gene therapy clinical trials adapted from (Ginn et al., 2018)

1.2 Manufacturing Process

As with most biologic manufacturing processes, the complexity of the molecules' physical and chemical properties dictates the complexity of the purification strategies. These larger biotherapeutic particles can range in size from 20 nm for AAV to 1000 nm for the measles virus (Moleirinho et al., 2020). Additionally, the particles' shape can compound the complexity. Some particles, like influenza, VLPs, and the measles virus, have a pleomorphic shape, while other particles have a ridged icosahedral geometry adenovirus and AAVs. Such complexities among particles prevent the generation of a generic purification process like those used for protein therapeutics. The entire production and purification process required additional efforts to ensure gentle conditions and aseptic processing in the before-mentioned viral cases. Obviously, each step and method used in the production process must be tailored to the specific biotherapeutic particle to ensure

that the particle's structure and integrity are maintained throughout the process. Otherwise, these particles could lose their ability to generate the desired immune response for vaccines or their infectivity for gene therapy viral vectors.

Generally speaking, the purification strategies for viral vectors can be broken down generally into common steps. After generating particles with either mammalian or insect cell culture, the downstream purification process begins with harvest and clarification steps. This is followed by intermediate purification steps using concentration techniques and chromatography methods. Finally, particles undergo a polishing and formulation step. The process can include a sterile filtration step. These downstream purification steps account for most of the overall manufacturing cost for viral vector production and are critical for a successful product (Fuerstenau-Sharp et al., 2017).

One of the unit operations that take up a significant portion of the downstream process is chromatography. This unit operation is routinely used with therapeutic protein production due to its scalability; however, this is not the case with larger molecules such as DNA and viruses. Figure 1.2 shows the variability of purification yields for viral vectors as reported in literature (adapted from Walker (2011)). Moreover, due to a packed bed's inherent complexity, characterizing a chromatography unit operation and accurately modeling the process can be extremely time and resource-intensive.

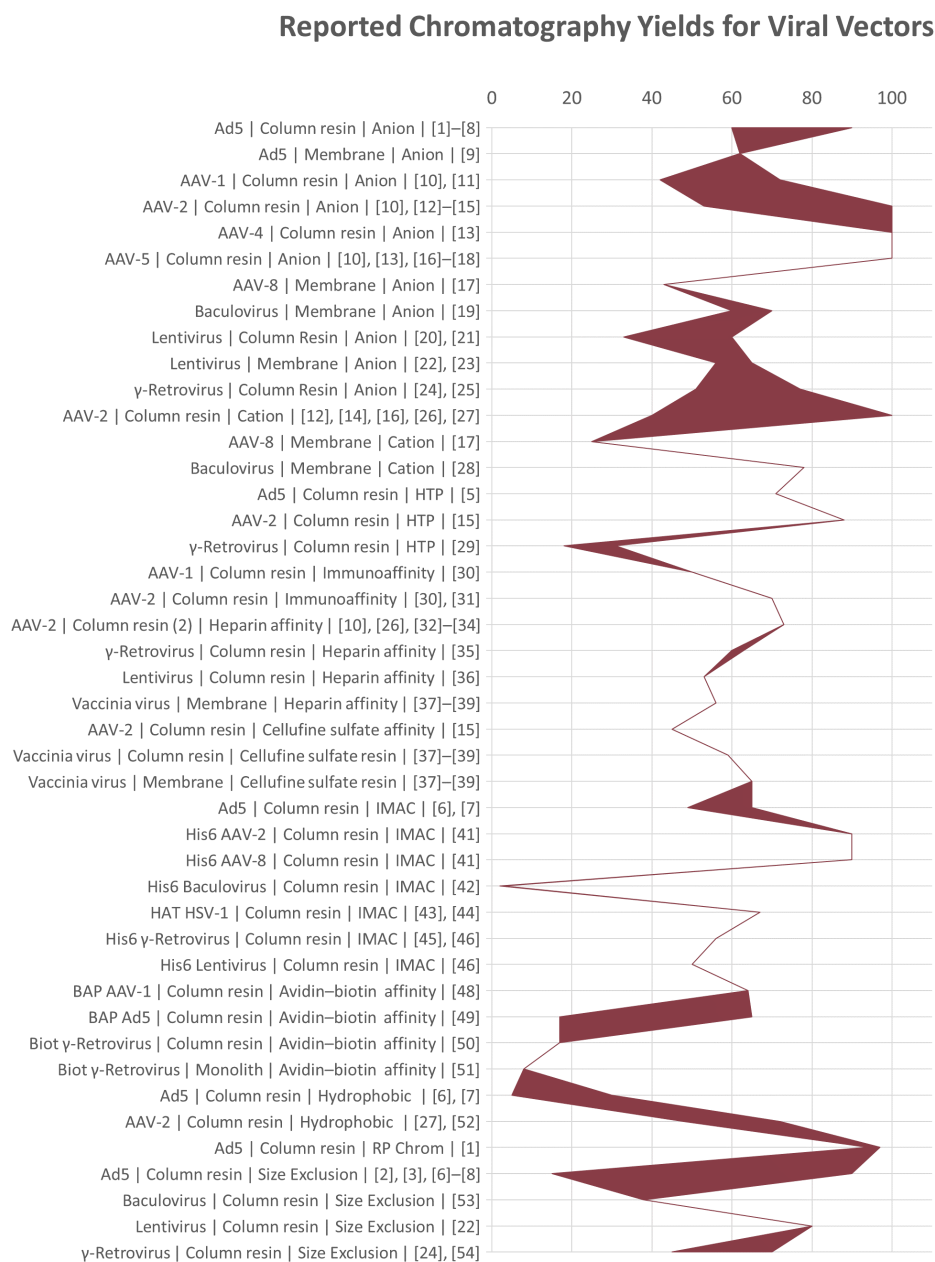


FIGURE 1.2: Chromatography yields for viral vectors as reported in literature. Adapted from (Walker, 2011), references can be found in Appendix A.

1.3 This Work

This research aims to utilize advances in Computational Fluid Dynamics (CFD) modeling to generate a first-principles model of the chromatographic process while minimizing the demand for physical resources. Such a model would help further process understanding in therapeutic protein production and be utilized to select appropriate gene therapy purification strategies while minimizing material operational costs.

Aim 1 was to identify the best geometries required for capturing the fluid flow during the chromatographic process at the bead level using CFD. Bioprocess liquid chromatography columns consist of a bed of randomly packed beads with two levels of porosity to improve mass transfer issues associated with larger molecules. These porous beads have throughpores that enable fractions of the convective fluid flow to assist with the diffusive flow by pushing large molecules into the bead's inner region, made up of smaller diffusive pores. This indicated four levels of porous geometric resolutions to consider when modeling a chromatography column, that of the packed bed, bead, throughpore, and diffusive pore level. Current commercial CFD software packages offer porous media zones; however, they are designed to model pressure gradient using momentum sinks and not the chaotic nature of fluid flow that could result in shear and eddy formation that can arise with the flow around closely packed spheres. Different particle arrangements were investigated to capture the porosity structure of a packed bed geometry for CFD modeling. Different explicit porous renderings of bead geometries were also examined to capture the convective nature of the throughpores with perfusion resin. Because I was mainly focused on looking at shear within the chromatography columns, these two levels (packed bed and throughpore) were deemed to have a sufficient level of resolution for CFD modeling. Additionally, to minimize the influence of the fluid flow's boundary conditions, the inlet

and outlet were extruded out from the interest region by four bead diameters.

Aim 2 was devoted to mesh these complex, explicit geometries of packed beds and porous particles effectively and efficiently. Accurate grid generation is critical to CFD modeling; thus, meshing methods need to consider geometric fidelity and grid density to capture all relevant flow fields while balancing demand on computational resources. Increasing geometric complexity can increase the mesh complexity required for an accurate CFD simulation. File type compatibility with meshing software was also factored in due to the constraints brought on by the packed bed generating software platform's output files. Issues can arise with the meshing of packed beds for CFD modeling at contact point regions, so contact point modification methods and their implementation of geometry modification were also investigated. For explicit renderings of porosity for chromatography beads, features considered include the number of subdomains, sharp corners/edges, continuity of the fluid region, and geometry pattern density. Additionally, typical CFD mesh quality metrics were considered for generating the computational grid before conduction mesh independence studies. These factors led to a pseudo-flow-cell model with 14 spheres surrounding a bead with explicit porous geometry to mimic fluid flow around a perfusion resin in a packed bed.

Aim 3 was to develop a CFD steady-state model that simulates fluid flow through and around a perfusion resin in a pseudo packed bed flow cell by modifying the shape and density of the porous geometry patterns. Mesh independence studies for the different explicit geometries were performed as steady-state simulations. The fluid flow ratio through and around particles was compared with commercial resin's literature values to identify the appropriate explicit porous geometry rendering before assessing shear and eddy formation within the packed bed. These preliminary studies show that

shear is relatively low at process operating conditions and that the low yield of large biotherapeutic particles in chromatography is likely, not due to shear forces.

Aim 4 was to develop a transient model of protein binding and eluting from the explicit geometric resin. The model was split into a two-part transient CFD DPM where tracer particles with IgG properties injected into the fluid domain using the steady-state model's velocity profile to initialize the simulation. DPM particle fates were tracked, and age distribution functions were calculated for particles captured on the resin with explicit geometry (also referred to as the bead of interest) and the outlet to characterize the macromixing in the model. I then compared this model with existing single parameter models. My models show similar distribution profiles as previous models and provide additional information about the shear forces acting on the particles.

Chapter 2

Background/Literature Review

2.1 Gene Therapy

Gene therapy is a novel therapeutic technique, which utilizes genetic methods to treat various human diseases. Gene therapy can be used to silence a mutated gene, replace the non-functioning gene with a functional one, or to introduce a new gene into the body to help fight a disease. For these genes to be effective in fighting the disease, they need to be inserted into the correct cell types/tissues, in addition to being expressed within those cells. A wide variety of vectors and gene delivery techniques have been developed to transfer the genetic material to targeted cells. While nonviral approaches have been increasingly trendy in recent years, viral vectors remain by far the most popular approach, making up two-thirds of the trials performed to date (Ginn et al., 2018) , see Figure 1.1 for a breakdown of vectors used in clinical trials.

Broadly, gene therapy can be categorized into two types based on the cells it targets: germline gene therapy and somatic gene therapy. In somatic gene therapy, the genetic material is inserted into some targeted cells, but the change is not passed down

to the next generation. In contrast, germline gene therapy, the therapeutic gene will be passed along to the next generation. This difference is significant because current legislation only allows gene therapy to target somatic cells (Wirth, Parker, and Ylä-Herttuala, 2013). The first authorized gene transfer study, which used a retroviral vector, took place in 1989 at the National Institutes of Health (NIH) (Rosenberg et al., 1990). Since then, a wide variety of vectors and gene delivery techniques have been employed in clinical trials, with the majority being viral vectors (Rosenberg et al., 1990; Ginn et al., 2018).

Viral vectors were initially chosen because they insert genetic material into the host cells as part of their replication cycle. Usually, this genetic material contains the building blocks to create more viruses by hijacking the cells' production machinery, creating more viral particles, and infecting more cells. Some of these viruses can physically incorporate their genes into the host's genome and can be expressed throughout the life span of that cell. Scientists have commandeered this process by replacing the genes coding for replication with those of genes that, when expressed, encode for a therapeutic effect. Harnessing this ability would allow viral vectors to genetically modify cells and provide a therapeutic benefit to people struggling with lifelong diseases.

There are, however, multiple ways to use these vectors for therapeutic benefit. The three general methods are gene silencing, gene replacement, and gene augmentation. In gene silencing, RNA interference inactivates or "knocks out" a mutated gene that is not functioning correctly. Another way to use gene therapy is by gene addition/replacement/correction, where one adds or replaces the mutated (nonfunctioning) gene that caused disease with a healthy copy of the gene. For example, BioMarin was developing a gene therapy that inserts a working copy of factor VIII for people with

Hemophilia A who are missing or have low levels of clotting factor VIII (BioMarin Pharmaceutical Inc., 2019). Finally, in gene augmentation, a new gene into the body to help fight a disease such as with CAR T cell therapy Decision Resources Group (2019). Utilizing these viral vectors as genetic tools could provide a permanent solutions to many lifelong diseases.

To date, many of the gene therapy clinical trials rely on a variety of viral vectors to deliver the therapeutic gene, which includes adenovirus, retrovirus, lentivirus, adeno-associated virus, vaccinia virus, herpes simplex virus, and pox virus. Figure 1.1 shows the breakdown of vectors used to date in clinical trials, as reported by Ginn et al. (2018). While there have been several clinical trials to date, the majority of products are still in phase I and phase I/II (BioCentury Inc., 2019). This bottleneck of gene therapy products in early phases of clinical trials highlights a need for improvements in high volume clinical-grade vector production.

2.1.1 Types of Viral Vectors

Each viral vector has its advantages and disadvantages, often stemming from their anatomy, see Figure 2.1. The relative merits of different viral vectors stem from how they store their genome, the makeup of their capsid proteins, and whether or not they have an envelope.

Viral vectors use either DNA or RNA to store genes and regulatory sequences, often referred to as an expression cassette, that are then used to direct the host cell's machinery into making viral RNA and proteins. These expression cassettes are comprised of a promoter sequence, open reading frame and a 3' untranslated region

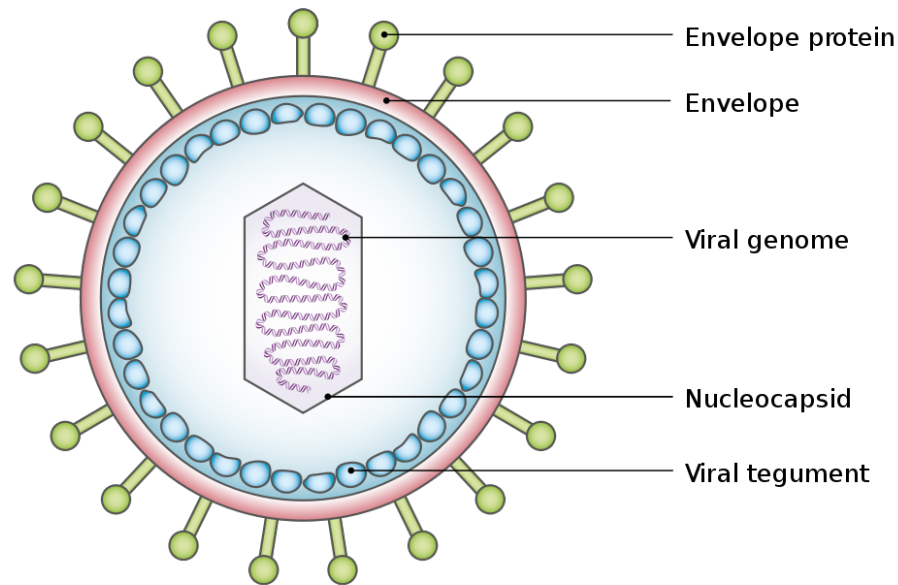


FIGURE 2.1: Viral vector structure, (Taylor, 2010)

Viruses that keep their genetic material in the form of RNA require expression of additional enzymes, such as reverse transcriptase and integrase, to reverse transcribe the RNA molecule into a DNA molecule and then integrated it into a semi-random location in the host cell genome (Poletti and Mavilio, 2018). These additional enzymes are often packed into the viral particle with the nucleic acids. Additionally, depending on the virus selected, transfection of the genetic material can be either stable (i.e. DNA successfully integrated into the cellular genome) or episomal (i.e. left free in the nucleus and not integrated into the cellular genome).

The capsid, or the protein coat enclosing the nucleic acids of the virion, also plays an essential role in shaping the properties of the viral vector. This shell is made up of numerous copies of one or a few protein subunits that self-assemble to form a symmetric shell that protects the genome and constrains the genome length. Additionally, these

capsid proteins are usually positively charged for counteracting the negatively charged nucleic acids of the genome. Some viral vectors also have a phospholipid envelope derived from the host cell membrane, covering their protective protein capsid. This viral envelope has glycoproteins from the host as well as the viral genome, which allows the virus to hide from the immune system, as well as identify and bind to host receptor sites, fuse with the host's membrane, and allow the capsid and genome to enter and further proliferate in a new host cell. These surface proteins, whether on an envelope or as part of a capsid, can vary even among different types of vectors and dictate the surface characteristics and the specificity of the viral vector.

2.1.2 Manufacturing Process Overview

For the clinical implementation of gene therapy, large-scale production processes need to be in place to generate highly pure and biologically active vectors. Such processes need to fulfill regulatory chemistry, manufacturing, and controls (CMC) requirements, in addition to being cost-effective, robust, and scalable. Moreover, these processes would ideally apply to a large variety of viral vectors (Morenweiser, 2005). Figure 2.2 shows a generic multi-step vector production process with upstream and downstream components that can vary depending on the properties of the viral vector.

The upstream component of viral vector production for gene therapy involves the growth and harvesting of viruses, while downstream focuses on vector purification. It should be noted that downstream purification accounts for a bulk of the overall manufacturing cost and is often the processing bottleneck (Lyddiatt and O'Sullivan, 1998). The harvesting step, or primary recovery, can vary depending on whether vector production is

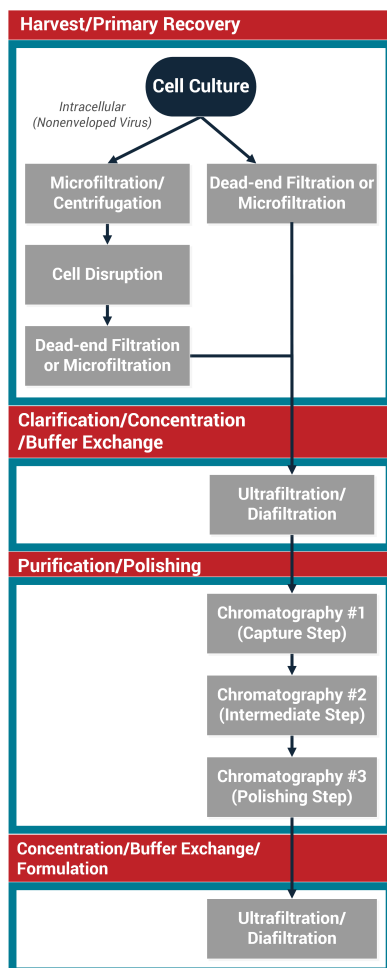


FIGURE 2.2: Virus purification for large-scale operations, adapted from Merten et al. (2014)

intracellular or extracellular for nonenveloped and enveloped vectors, respectively. For intracellular vector expression (e.g., adenoviral and AAV vectors), cells need first to be separated from the cell culture before undergoing a cell disruption step. The downstream steps generally include clarification, capture, purification, polishing, and formulation (Merten et al., 2014). The clarification and formulation steps concentrate and exchange buffers using ultrafiltration and diafiltration unit operations. Capture, purification, and polishing are often executed using different types of chromatography. Each of these processes must be customized to the specific biochemical and physical properties of the gene-therapy vector to preserve the viral infectivity and maximize product recovery (Fuerstenau-Sharp et al., 2017). Considering characteristics, such as virus particle size, stability, and charge at neutral pH, is critical for selecting purification methods and identifying possible steps that affect the final-product quality (de las Mercedes Segura, Kamen, and Garnier, 2006; Fuerstenau-Sharp et al., 2017).

2.2 Physics of Flow and Traditional Definitions

2.2.1 Packed Beds and Packing Regimes

Despite extensive use of packed beds process engineering, fluid flow in packed beds is complex and exceedingly difficult to study because of the media's inherently random and disordered characteristics. Process engineers use packed beds in a multitude of unit operations, such as filtration, distillation, and chromatography, to name a few. These packed beds can generally be thought of as a matrix-like structure formed by particles deposited into a container. These packed particles form pores or voids which fluid

is free to percolate through. When the packed bed is confined inside a cylindrical tube, it is referred to as a packed column, and it can either be fixed or fluidized. As the name implies, a fixed bed is comprised of particles that are static and fixed in place, and unable to move. In contrast, a fluidized bed is a physical phenomenon where particles are carried randomly in the container by fluid flow. This work is mainly concerned with the study of fixed beds and their influence on the flow.

2.2.2 Fluid Flow Definitions

The transport of flow through porous media follows the same relationships as those in basic fluid mechanics. The fluid flux through the packed bed is expressed by the volumetric flow rate, Q ($\text{m}^3 \text{s}^{-1}$). The superficial (or empty tube) velocity, U , is related to the volumetric flow rate by the following expression:

$$U = \frac{Q}{A} \quad (2.1)$$

Where A is the cross-sectional area (m^2) of the tube. Superficial velocity is referred to as “superficial” because it is what the fluid’s velocity would be if there were no porous media, i.e., an empty tube. But because the presence of solid particles within the bed reduces the available fluid flow area, the fluid squeezed through the pores at a velocity greater than the superficial velocity to preserve the fluid’s continuity. This velocity is referred to as the interstitial velocity (or the velocity within the bed, U_o), and is related to the superficial velocity by the following expression:

$$U_o = \frac{U}{\varepsilon} \quad (2.2)$$

Where ε is the global property of average porosity. Resistance to fluid flow through porous media is determined the void volume in the bed (V_v) and governed by the area available for the flow to pass. Volume concentration (C) is the ratio of volume solids in bed to the total bed volume and the remaining volume fraction of the bed is a dimensionless term called the void fraction or porosity (ε). Equation 2.2 forms the average pore velocity since there is no guarantee that pores are homogeneous in a disordered bed (Baker, 2011; Holdich, 2002).

$$\varepsilon = \frac{V_v}{V} \quad (2.3)$$

$$\varepsilon + C = 1 \quad (2.4)$$

It follows that at one extreme, when porosity is zero, the bed is full of solids, and there is nowhere for the fluid to flow; thus, the resistance is infinite. At the other extreme, when porosity is unity, and there are no solids present, the interstitial velocity of the fluid is the same as the superficial velocity.

2.2.3 Aspect Ratio

Packed beds are characterized as particles packed into a pipe where they interact with a fluid. Still, the dimensions of the pipe and the particles influence the flow and can vary dramatically. To characterize and compare different packed bed scenarios, it is often desirable to use a dimensionless property called the aspect ratio (A_{ratio}). This dimensionless property is the packed bed's ratio between the equivalent diameter (d_p) of

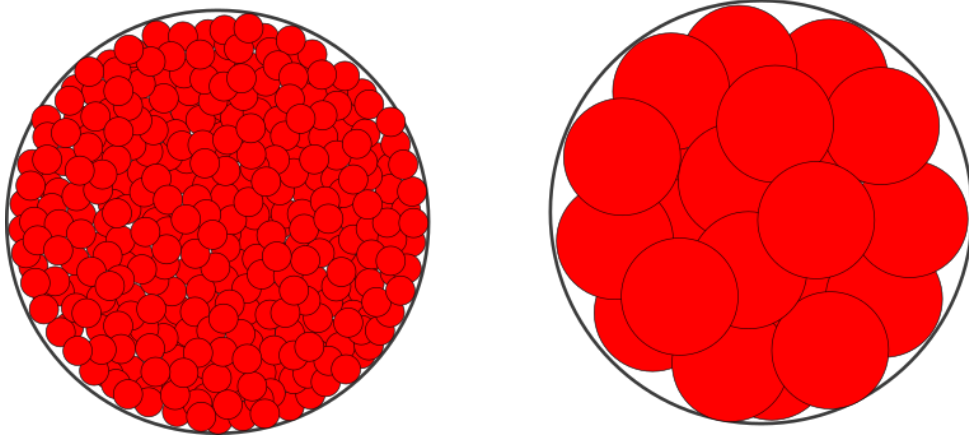


FIGURE 2.3: Left: high aspect ratio. Right: low aspect ratio

the particle and the container's diameter (D), given as:

$$A_{ratio} = \frac{D}{d_p} \quad (2.5)$$

Packed beds are described as having either low or high aspect ratios, which influences the packing structure (ordered vs. disordered) and the velocity profile. For packed beds with high aspect ratios (such as a tube filled with sand), the velocity profile would be relatively uniform throughout the column due to the pseudo-homogeneous network of pores formed by the small solid particles. On the other hand, low aspect ratio packed beds are usually highly disordered with inhomogeneous packing, leading to variation in local porosity and thus velocity profiles. Refer to Figure 2.3 for an example of different aspect ratios. The exact value that dictates whether a bed has a high or low aspect ratio is not well defined, but $A_{ratio} = 50$ is a reasonable distinguishing value (Baker, 2011). (De Wilde et al., 2009)

2.2.4 Wall Regions

When particles are packed into a container, they usually orient themselves in a random disordered fashion. Those that are in close contact with the walls are not as efficiently packed as those closer to the center of the bed because of the container's flat surface. Such radial variations in packing result in an increased porosity next to the wall in comparison to the bed's core. This higher porosity region is often referred to as the wall region, and the region unaffected by the confining wall is called the core region.

Many researchers have conducted experiments to determine packing structures and voidage variations of packed beds (Roblee, Baird, and Tierney, 1958; Benenati and Brosilow, 1962; Goodling et al., 1983; Di Felice and Gibilaro, 2004). The first investigators to do this were Roblee, Baird, and Tierney (1958) who measured radial variation by packing cardboard cylinders with cork spheres, filling the void space with molten wax, and slicing it into sections once the wax had solidified. The authors found that the bed porosity showed attenuating oscillations in the near-wall region until it reached a constant value of around 4 to 5 particle diameters from the wall. Benenati and Brosilow (1962) found similar results when they filled a container with lead spheres (shot) of uniform size and epoxy resin. After the authors cured the container, they machined it into sections and used the average density of each annular ring to determine the average voidage of that part. They found that the porosity behaved similarly to the findings of Roblee, Baird, and Tierney (1958).

Another paper published by Goodling et al. (1983) reported similar findings of bed porosity behavior. They filled a cylinder with polystyrene spheres (for the packing) and an epoxy mixed with finely ground iron (for the void space). Once cured, the authors cut the column into thin annular rings and determined the radial porosity for each ring

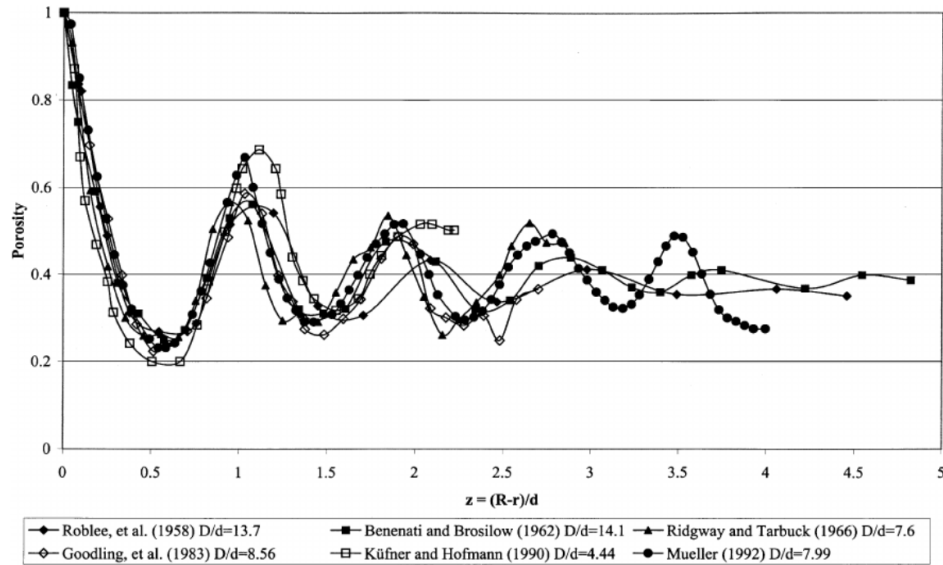


FIGURE 2.4: Radial porosity variance, (De Klerk, 2003)

by calculating the change in mass and volume between cuts. Goodling et al. (1983) found that at the cylinder wall, the radial porosity reached unity but oscillated with a damped magnitude towards the mean build porosity near the bed core. Moreover, the experiments showed that the wall effects could be detected up to a distance of 5 sphere diameters, similar to Roblee, Baird, and Tierney (1958), and Benenati and Brosilow (1962). De Klerk (2003) compiled the variation in radial porosity determined by these and various other authors, refer to Figure 2.4.

For most bioprocessing manufacturing applications, such as liquid chromatography, packed beds have a high aspect ratio. Thus, the wall region has little impact on the fluid flow at large scale, but may become more important at smaller scales, such as in high throughput experiments.

2.2.5 Empirical Equations and Models

Liquid chromatography remains one of the main pillars of the purification process for manufacturing therapeutic proteins, despite being the process bottleneck and most expensive unit operation (Kelley, 2007). Process development and optimization strategies in the biotech industry often utilize modeling to understand the process and product effectively, efficiently, and economically. These models help identify the critical quality attributes (CQAs) of the product and the critical process parameters (CPPs) of the process (Rathore, 2014)

There are two types of modeling approaches used in bioprocessing, empirical modeling, and mechanistic modeling. Empirical modeling treats the system as a black box by relying heavily on a statistical analysis of experimental data from a design of experiments (DOE) to find relationships between output responses and input variables. While this approach does provide sufficient process knowledge for creating a design space, it is limited in its accuracy and robustness (Rathore and Kumar, 2017). In contrast, mechanistic modeling employs functional relationships derived from natural laws governing the physical and biochemical effects. While the empirical approach has dominated the industry's process development and optimization methods, properly calibrated mechanistic models can better predict process performance inside the calibration conditions and extrapolate performance to outside these conditions.

Many mechanistic models have been developed over the years for liquid chromatography, each with their assumptions, but they are primarily based on mass conservation equations. According to Shekhawat and Rathore (2019), liquid chromatography has three levels where a mass transfer of solute molecules occur in a packed bed column: “(i) interstitial bulk volume to the external stagnant film around the adsorbent particles

through convection and axial dispersion, (ii) external film to interior mobile phase of the adsorbent particles through film diffusion, and (iii) interior mobile phase to the stationary phase of the adsorbent particles through pore diffusion” (Shekhawat and Rathore, 2019). Additionally, surface diffusion of the adsorbed solute molecules on the stationary phase can also occur; however, for low-affinity solutes, this is usually neglected because it is one to two orders of magnitude lower than pore diffusion (Suzuki, 1990).

These mass transport models available in the literature can be broken into three broad categories: (i) equilibrium theory, (ii) plate theory, and (iii) rate models (Glueckauf, 1955; Ruthven, 1984). Refer to (Shekhawat and Rathore, 2019) for an excellent review of the different models.

2.2.5.1 Distributions of Residence Times

First proposed by MacMullin and Weber Jr (1935), the distribution of residence times for analysis of chemical reactor performance has been widely used in chemical engineering since Prof. Danckwerts (1953) characterized most of the distributions of interest. Residence time is defined as the time the atoms have spent in the reactor, and engineers often look at the distribution since some molecules leave reactors immediately while others linger. This RTD of a reactor is a characteristic of mixing in the reactor and can give distinctive clues about the type of mixing and information about the reactor’s features (Fogler, 2006).

RTDs are measured experimentally by injecting a tracer (i.e., inert chemical, molecule, or atom) at some time and then measuring the tracer’s concentration at the outlet stream as a function of time. There are two types of injection methods, pulse input, and step input. For pulse input experiments, a known amount of tracer, N_0 , is

quickly injected into the inlet, or feed stream, of the reactor for as short of a time as possible. The outlet-concentration is then measured as a function of time and this curve is referred to as the C-curve, or $C(t)$, in RTD analysis. The amount of tracer, ΔN , leaving the reactor between t and $t + \Delta t$ is defined as:

$$\Delta N = C(t) v \Delta t \quad (2.6)$$

where v is the the outlet volumetric flow rate. If we divide by the total amount of tracer material injected, N_0 into the reactor we obtain the following equation which represents the fraction of material that has a residence time in the reactor between time t and $t + \Delta t$:

$$\frac{\Delta N}{N_0} = \frac{v C(t)}{N_0} \quad (2.7)$$

Where for pulse injection we define

$$E(t) = \frac{v C(t)}{N_0} \quad (2.8)$$

so that

$$\frac{\Delta N}{N_0} = E(t) \Delta t \quad (2.9)$$

This quantity, $E(t)$, is essentially the exit-age distribution function often referred to as the E-curve in chemical engineering (Fogler, 2006). This function describes quantitatively how much time fluid elements have spent in the reactor. When the volumetric flow rate

v is constant, $E(t)$ is defined as:

$$E(t) = \frac{C(t)}{\int_0^\infty C(t) dt} \quad (2.10)$$

Since the fraction of material that has been in the reactor from $t = 0$ to $t = \infty$ is one, it follows that:

$$\int_0^\infty E(t) dt = 1 \quad (2.11)$$

Using these E -curves engineers often compare RTDs by using their moments instead of comparing the entire distribution (Wen, Fan, and others, 1975). The mean residence time (t_m) is calculated using the first moment of the RTD function, $E(t)$, and for constant volumetric flow, space time ($\tau = V/v$) is equal to the mean residence time (see Fogler (2006) for proof):

$$\tau = t_m = \frac{\int_0^\infty tE(t) dt}{\int_0^\infty E(t) dt} = \int_0^\infty tE(t) dt \quad (2.12)$$

The second moment of $E(t)$ that engineers often use is the variance, which measures the “spread” of the distribution, and is the square of the standard deviation:

$$\sigma^2 = \int_0^\infty (t - t_m)^2 E(t) dt \quad (2.13)$$

Additionally, RTD curves are often normalized using the parameter Θ which is defined as:

$$\Theta \equiv \frac{t}{\tau} \quad (2.14)$$

so that the dimensionless function $E(\Theta)$ is defined as:

$$E(\Theta) = \tau E(t) \quad (2.15)$$

The normalized functions allow engineers to compare RTD functions of identical reactors across different flow rates.

2.3 Computational Fluid Dynamics (CFD)

CFD combines the fields of fluid mechanics and computer science in a way that allows us to predict information about the ways fluid flows for a given situation. CFD allows engineers to approximately predict the fluid flow fields using numerical analysis and algorithms to solve the Navier-Stokes equations for mass and momentum. These models provide insight into product/process performance that would otherwise be difficult to attain experimentally, in addition to minimizing resource investment.

2.3.1 Computational methods

The CFD modeling process can be divided into three stages: pre-processing, solution, and post-processing (see Figure 2.5). During pre-processing, the geometry of the fluid region is first digitally rendered using computer-aided design (CAD) software. This fluid geometry is then divided into smaller discrete control volumes, also called elements or cells. The domain containing all of these elements is referred to as the computational grid or mesh. After generating the mesh, pre-processing software allows users to set the governing equations, material properties, and boundary conditions appropriate for the

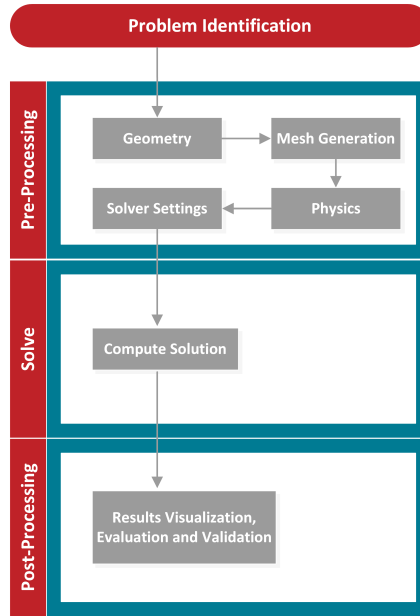


FIGURE 2.5: Computational fluid dynamics process work flow.

simulation. Then, during the solution phase, the solver incorporates the model information, discretizes the governing equations, and iteratively solves the flow variables for each cell in the mesh. Once the error between successive iterations of the solution variables has reached a specified level, referred to as residuals, the solution is considered converged. The results are then extracted and analyzed during the post-processing stage. The post-processing software gives engineers tools to generate high-end graphical visualization of quantitative measures (Horner, Joshi, and Waghmare, 2017).

2.3.2 Meshing and Contact Point Modifications

For CFD simulations using the finite-volume method, the fluid domain is subdivided into small control volumes, or computational cells, during mesh generation. For

CFD models of fixed bed reactors, automatic meshing algorithms have difficulty generating quality mesh near the particle-particle and particle-wall contact points. These regions around the contact points can be very narrow, resulting in computational cells with extremely poor quality (i.e., low aspect ratio, large skewness, etc.), which often results in convergence problems. Such cell quality problems can be overcome by highly refining the mesh in this region. However, this increase in the number of cells increases the computational cost and the calculation time to solution.

The drawback to these methods dealing with low mesh quality at contact points led to four alternative methods presented in the literature: gaps, overlaps, bridges, and caps (Dixon, Nijemeisland, and Stitt, 2013)(see Figure 2.6). These methods can be separated into two classes of methods based on how they manipulate the geometry.

Global modification methods (gaps and overlaps) affect the entire bed structure by either shrinking or enlarging particles, respectively, by a specified value. In contrast, local methods (bridges and caps) modify only the contact point and its immediate neighborhood. The “bridges” method, presented by Ookawara et al. (2007), involves inserting a cylinder between the two objects in contact or within a specified tolerance of each other. In their study, Ookawara et al. oriented these cylinders so that the cylinder axis was aligned with the vector connecting the particle centers. The cylinders encapsulated the contact point and narrow region surrounding it, which avoids drastically changing the bed void fraction. The counterpart to the bridges method is the caps method, proposed by Eppinger, Seidler, and Kraume (2011), where particles are locally flattened at contact points, so the vertices of the surface elements maintain a specified minimum distance. This method is the equivalent to removing the spherical caps at the contact points, and the derivation of the method’s name. This type of local mesh modification leads to a

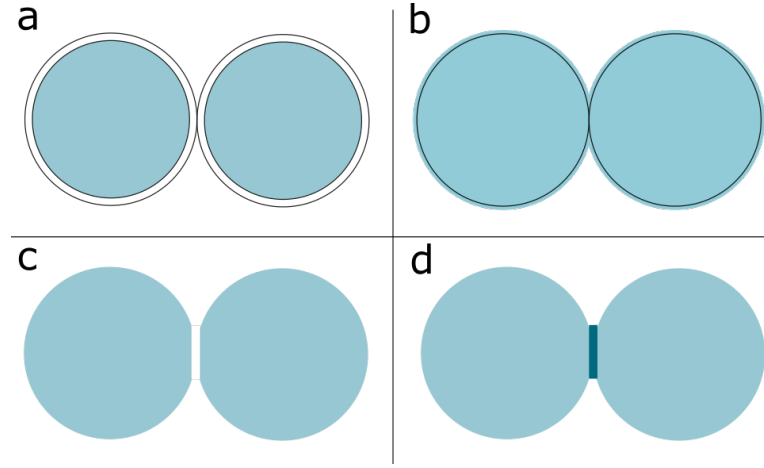


FIGURE 2.6: Diagram of the types of contact point modification methods: (a) Gaps; (b) Overlaps; (c) Caps and (d) Bridges; adapted from Dixon, Nijemeisland, and Stitt (2013)

small gap between the particles, which can be filled with cells of good quality by meshing algorithms. These local mesh modification methods for contact points have a smaller impact on the bed void fraction than the global methods, which can affect the accuracy of CFD drag coefficient and pressure drop calculations (Dixon, Nijemeisland, and Stitt, 2013).

2.3.3 Porous Media Rendering

Traditionally, CFD used continuum modeling of porous media, where porous structures are represented as a volume average continuum without resolving the microscale features. While this method of modeling is mathematically rigorous, it still has practical shortcomings when it comes to porous material modeling. Such challenges arise with multiphase modeling because discrete pore-scale phenomena and events are lost, and only the volume-averaged amount of fluid phase for the computational node is known.

Moreover, these models utilize macroscopic transport properties that rely on constitutive relationships derived from experiments, such as permeability coefficient or effective diffusivity. Often these continuum models rely on simple extensions of Darcy's law, such as the Blake-Kozeny equation, to characterize the heat and fluid flow inside porous media. Such macroscopic properties can lose their predictive utility when the porous media is not perfectly uniform since the actual distribution is lost in the formulation (Gostick et al., 2016).

Some of these issues of macroscopic representation can be overcome through more of a microscopic approach to modeling porosity. Instead of using single-domain-based model, porosity can be directly modeled with explicit geometry, such as a cluster of spheres (Wittig, Richter, and Nikrityuk, 2012; Smits, Nakanishi, and Desmet, 2016) or as a network of pipes (Gostick et al., 2016). Such explicit porous renderings also allow better capture of flow around complex particle shapes. Additionally, combining both the microscopic and macroscopic approach allows accurate capture of the fluid flow for resins with a bimodal pore size distribution, i.e., the flow-through particles of perfusion resins (Smits, Nakanishi, and Desmet, 2016).

2.4 Discrete Phase Modeling Equations

The DPM in ANSYS Fluent uses the Lagrangian reference frame to calculate particle trajectories through the integration of the force balance on the particle (ANSYS Inc., 2019b). Such a force balance equates the particle inertia with forces acting on the

particle and is written in the following form:

$$m_p \frac{d\vec{u}_p}{dt} = \vec{F}_D + m_p \frac{\vec{g}(\rho_p - \rho)}{\rho_p} + \vec{F} \quad (2.16)$$

Where m_p is the particle mass, \vec{u}_p is the particle velocity, t is time, ρ is the fluid density, ρ_p is the density of the particle, \vec{F} is any additional forces, and \vec{F}_D is the drag force on the particle calculated by:

$$\vec{F}_D = m_p \frac{\vec{u} - \vec{u}_p}{\tau_r} \quad (2.17)$$

Where \vec{u} is the fluid phase velocity, \vec{u}_p is the particle velocity, and τ_r is the particle relaxation time calculated using the following formula (Gosman and Loannides, 1983):

$$\tau_p = \frac{\rho_p d_p^2}{18\mu} \frac{24}{C_d \text{Re}} \quad (2.18)$$

Where μ is molecular viscosity of the fluid, d_p is the particle diameter and Re is the relative Reynolds number, defined as (ANSYS Inc., 2019b; ANSYS Inc., 2019a)

$$\text{Re} \equiv \frac{\rho d_p |\vec{u}_p - \vec{u}|}{\mu} \quad (2.19)$$

In Equation 2.16, the term \vec{F} allows additional forces to be incorporated into the force balance, which can be significant for specific conditions. For this application the Saffman lift force and Brownian motion used and the following sections will go into further detail of their calculations.

2.4.1 Saffman Lift Force

Particles in the presence of a shear stress field often experience a lift force. This usually occurs when particles are near a wall, which pushes the particles away from the wall. ANSYS Fluent uses the lift force from Li and Ahmadi (1992), which is the generalized form of Saffman's expression (Saffman, 1965):

$$\vec{F}_{lift} = m_p \frac{2K v^{0.5} \rho d_{ij}}{\rho_p d_p (d_{lk} d_{kl})^{0.25}} (\vec{u} - \vec{u}_p) \quad (2.20)$$

Where $K = 2.594$ and d_{ij} is the deformation tensor (ANSYS Inc., 2019c). This equation is mainly intended for low Reynolds numbers and sub-micron particles.

2.4.2 Stokes-Cunningham Drag Law and Brownian Motion

Normally, the drag function F_D is expressed as:

$$F_D(u - u_p) = \frac{\mu}{\rho_p d_p^2} \frac{18 C_D \text{Re}}{24} (u - u_p) \quad (2.21)$$

However, the drag function F_D takes the following form for sub-micron particles in laminar flow:

$$F_D = \frac{18\mu}{d_p^2 \rho_p C_c} \quad (2.22)$$

Where

$$C_c = 1 + \frac{2\lambda}{d_p} (1.257 + 0.4^{-(1.1d_p/2\lambda)}) \quad (2.23)$$

Often referred to as the Stokes-Cunningham Drag Law, it is a modified form of the Stokes' drag law and uses the Cunningham correction factor (C_c) defined in Equation 2.23 (Ounis, Ahmadi, and McLaughlin, 1991). Additionally, μ is the viscosity, ρ_p is the particle density, d_p is the particle diameter and λ is the molecular mean free path. The Cunningham correction factor also plays a role in calculating the effect of Brownian motion on the particle trajectories; see Section 3.3.7 below for further details.

2.4.3 Brownian Random Force

The random motion that sub-micron particles move in when suspended in a fluid is referred to as Brownian motion. This motion can be included in the model as an additional force term using a Gaussian white noise process with spectral intensity ($S_{n,i,j}$) (Li and Ahmadi, 1992).

$$S_{n,i,j} = S_0 \delta_{ij} \quad (2.24)$$

$$S_0 = \frac{216vk_BT}{\pi^2 \rho d_p^5 \left(\frac{\rho_p}{\rho}\right)^2 C_c} \quad (2.25)$$

Where δ_{ij} is the Kronecker delta function, T is the absolute temperature of the fluid, v is the kinematic viscosity, k_B is the Boltzmann constant and C_c is the Cunningham correction from Equation 2.23. The amplitude of the Brownian force is defined as:

$$F_{b_i} = m_p \zeta_i \sqrt{\frac{\pi S_o}{\Delta t}} \quad (2.26)$$

The force amplitude is evaluated at each time step, where ζ_i are the zero-mean,

unit-variance-independent Gaussian random numbers. Brownian force is used when simulations are laminar and requires the energy equation to be solved.

2.4.4 Particle Tracking with Eulerian-Lagrangian Method

In the Eulerian-Lagrangian approach to flow modeling, the primary fluid is treated as a continuum using the Eulerian reference frame. This main fluid phase uses numerical methods to solve the Navier-Stokes equations while the discrete phase tracks the dispersed particles, droplets, or bubbles using the Lagrangian description. These particles can exchange momentum, mass, and energy with the continuum of the main fluid phase as long as the discrete phase has a low volume fraction. Particle-particle interactions can be included or neglected in the simulation. These interactions can increase the complexity of the model and lengthen the computational time.

2.4.5 Coupling between continuous and discrete phases

In ANSYS Fluent, “coupling” refers to calculating the solution of the Eulerian and Lagrangian fields simultaneously. In the discrete phase model, ANSYS Fluent keeps track of the heat, mass, and momentum gained or lost by the particle trajectory, which can be incorporated into the subsequent continuous phase calculations. While the continuous phase always impacts the discrete phase (one-way coupling), the discrete phase may or may not be set up so that it also affects the continuum (two-way coupling).

Two-way coupling is achieved first by solving the continuous phase flow field calculations in the Eulerian reference frame, then computing the trajectories of the particles/droplets in the Lagrangian reference frame, and finally, updating the continuous

phase source terms (mass, momentum, and energy) for the next continuous flow field calculation. This whole process is repeated until the solution in both phases stops changing. The interphase exchange for the momentum and energy terms are as follows:

$$F_s \sum \left(\frac{18\mu C_D \text{Re}}{\rho_p d_p^2 24} (u_p - u) + F_{other} \right) \dot{m} \Delta t \quad (2.27)$$

$$Q_s = \dot{m}_p c_p (T_{pin-cell} - T_{pout-cell}) \quad (2.28)$$

These source terms embody the effects of the particles on the fluid by appearing on the right-hand side of the main fluid's momentum and energy equations. The $T_{pin-cell}$ and $T_{pout-cell}$ stand for the particle temperature going entering and leaving a cell, respectively, and are calculated in relation to the time of tracking the particles (Mahdavi, Sharifpur, and Meyer, 2018; ANSYS Inc., 2019c)

2.4.6 Shear Stress Integral

Because shear stress can have a negative impact on product quality, a User Defined Function (UDF) was written to compare the amount of shear stress experienced by the particle as it passes through the cells in the fluid domain. This custom UDF uses two DPM User Defined Scalars (UDS) to obtain the time-integrated value of the shear rate along the particle path using the trapezoidal rule. Equation 2.29 depicts a generic formula where S can be any of the fluid flow variables, and a diagram of the integral

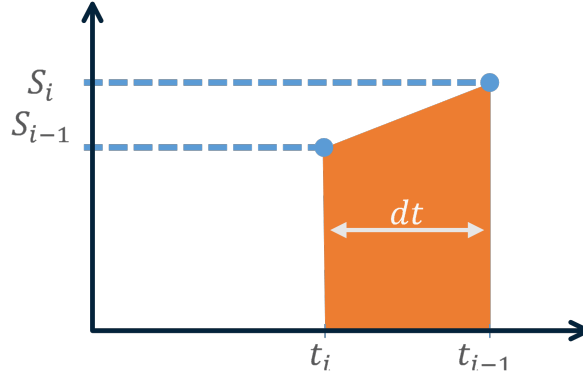


FIGURE 2.7: Integration using the Trapezoidal Rule

calculation is presented in Figure 2.7.

$$dt \cdot (S_i + S_{i-1})/2 \quad (2.29)$$

Where S_i is the fluid variable value from the cell the particle is currently in, S_{i-1} is the fluid variable value from the previous time step, dt is the time step size, t is the time with subscripts denoting the time step number. Using assignment addition, the integrated value at each step is added to `TP_USER_REAL(tp,0)`, and then `TP_USER_REAL(tp,1)` is then defined as the new fluid variable value for the current cell that will be used as the past value for the next time step. Because shear stress is not a variable directly calculated in fluent, the product of the strain rate magnitude and molecular viscosity is used instead, and is defined in Equation 2.30 as:

$$S = \mu \cdot \dot{\gamma} \quad (2.30)$$

Where μ is the molecular viscosity (kg/m·s) and $\dot{\gamma}$ is the strain rate (1/s), both variable derivations can be found in the Ansys Fluent Theory Guide (ANSYS Inc., 2019c).

Chapter 3

Methodology

3.1 Packed Bed Generation

In order to model a liquid chromatography column, a randomly packed bed was generated using rigid body dynamics available in the open-source code program Blender (Blender Foundation, 2019). I first investigated packing a column with spheres with methods adapted from (Partopour and Dixon, 2017). Because of updates to Blender's API, scripts from Partopour and Dixon had to be updated from 2.77 to 2.80 to handle the new features, and adapt to changes that make the API more consistent and reliable (Blender Developer Wiki, 2019). Besides spheres, other shapes such as cylinders, rashig rings, cubes, and hollow cubes were also considered for this investigation. Additional Python scripts were written to handle particle-particle contact point modification for any desired particle shape.

3.1.1 Geometry Software: Blender

Geometry setup and packing are done in the free, open-source software environment Blender 2.80, a free professional 3D modeling software that started initially as a proprietary program developed by the Dutch animation studio NeoGeo in 1995 until 1998, where it was released online as SGI freeware. Finally, in 2002, the non-profit Blender Foundation raised enough funds to release the Blender source code online (Roosendaal, 2019). Since then, Blender has primarily been developed voluntarily by its community of users/programmers. It now has a wide range of functionalities that include video editing, animation tools, sophisticated texture mapping, game logic, sculpting, path tracing rendering, real-time physics simulation etc. Moreover, it also has a scripting language program interface (python, (Python, 2018)) that allows users to create customized scripts and extensions to automate any part of the 3D modeling process.

Blender defines objects using surface meshes made up of nodes, edges, and faces represented by either quads, triangles, or other polygon meshes. These shapes can be defined using Bezier or other predefined 2D-curves that can be extruded into 3D objects, in addition to merely importing existing mesh files. These surface mesh properties can be easily modified, moved, or imported through Blender’s GUI or with Python scripting. Such features allow precise and rapid construction complex geometries.

3.1.2 Physics Engine and Rigid Body Dynamics

Real-time physics simulations in Blender are generated using a “physics engines,” which is a computer program that provide fast approximations of simulations for specific physical systems, such as rigid bodies, soft bodies, and fluids, using an adapted version

of the Discrete Element Method (DEM). Many of these engines (e.g., Open Dynamics Engine (*Open Dynamics Engine*), PhysX (*PhysX*), True Axis (*True Axis*), and Bullet Physics (Coumans, 2005), to name a few) are designed to provide set of standard features to support generation approximate physics-based animations that look as realistic as possible for a wide range of application areas. Computer games, animation software for digital production, including special effects in film and animation movies, robotics validation, virtual prototyping, and training simulators, are some of the industries which utilize physics engines for real-time playback (Hummel et al., 2012; Bender, Erleben, and Trinkle, 2014).

These designers, engineers, modelers, and animators often use a specific subtype of modeling called interactive RBD simulation (Bender et al., 2012). The term “interactive” indicates that the engine delivers plausible simulation results instantaneously, so the user interacts steadily with updated results, at the cost of decreased accuracy and simplified calculations (Bender, Erleben, and Trinkle, 2014). In contrast, “off-line” RBD physics engines require hours and days to solve a specific simulation but deliver highly accurate solutions but are decoupled from user interaction (Bender et al., 2012). In RBD simulations, objects are only considered to be rigid bodies, this simplifies the calculation because internal stresses and strain were ignored and the only dissipative mechanism calculated in the simulation was the friction between bodies.

What distinguishes RBD from classical DEM is that penetration of colliding bodies in RBD is prohibited, but calculation of contact forces and changes in velocities occur instantaneously, while for DEM such as the standard spring-dashpot model forces are determined gradually and are dependent on small penetrations (Baraff, 1997; Cleary, 1998). The major advantage to the RBD is the computational efficiency since the computational

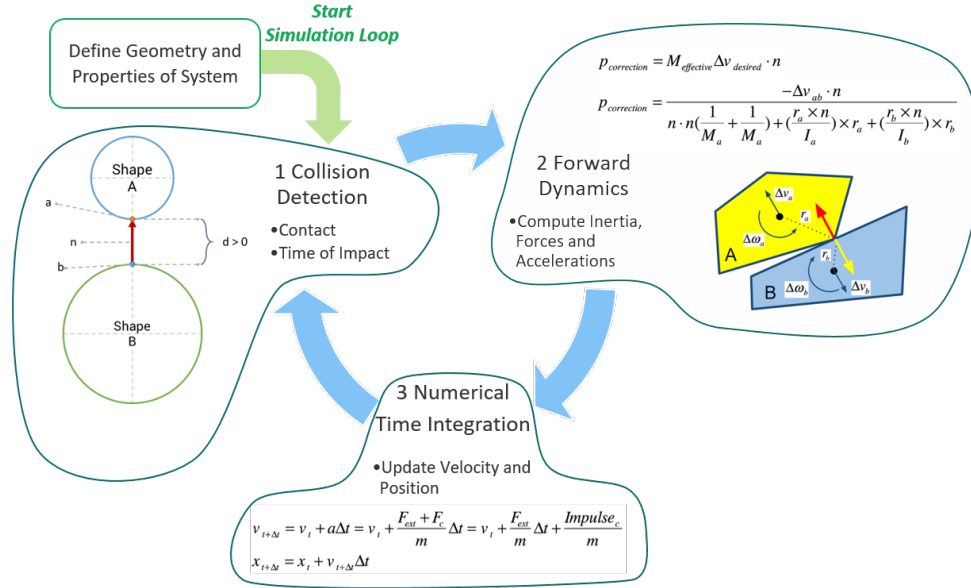


FIGURE 3.1: Overview of Bullet Physics engine simulation loop for RBD adapted from Coumans (2015). Geometry and properties of the system are defined before the rigid body simulation loop begins. The loop has 3 stages: (1) the collision detection stage, where I compute if, when and where contact between rigid bodies happen; contact point information includes a contact normal (n), pointing from Shape B towards Shape A, a distance (d) and two witness points, one on each object (a and b respectively). (2) The forward dynamics stage, which is when the computation of forces, inertia, and accelerations occur. And finally, (3) there is the numerical time integration, where the velocity and position of objects are updated by approximating the areas under the acceleration and velocity curve.

cost of contact handling is dramatically reduced (Williams and O'Connor, 1999).

The Bullet Physics engine's rigid body simulation loop can be broken down into 3 steps: collision detection, forward dynamics, and time integration. Refer to Figure 3.1 for a diagram of the rigid body simulation loop. The collision detection step is where Bullet computes if, when, and where contact between rigid bodies happen. The information calculated for contact points includes a contact normal (n), pointing from Shape B towards Shape A, a distance, and (d) two witness points, one on each object (a and b , respectively).

The distance in the contact point information also accounts for object overlap; negative closest distance values indicate that objects are overlapping while positive values indicate objects are not overlapping. The surface points (a and b) are also used to compute contact and friction forces for each object. In the forward dynamics step, Bullet calculates forces, inertia, and accelerations. The forces considered can be split into sub-types: (1) external forces, such as gravity, wind force field or other user forces, (2) constraint forces, such as contact (Newton's 2nd law), friction, and joints, (3) velocity-dependent forces (gyroscopic), and (4) position dependent forces (spring). The following equation in its simplified form is used to compute the correction impulse; see reference (Coumans, 2015, slides 38-41) for the derivation of equations:

$$p_{correction} = M_{effective} \Delta v_{desired} \cdot n \quad (3.1)$$

Where $M_{effective}$ is the effective mass inverse, which is the change in velocity due to an impulse, projected onto the contact normal, and $\Delta v_{desired}$ is the change in (relative) velocity for both objects for Equation 3.1. The inverse effective mass is defined as:

$$M_{inv.effective} = \frac{dot(\Delta u_{ab}, n)}{|Impulse|} \quad (3.2)$$

Where Δu_{ab} is defined as:

$$\Delta u_{ab} = \frac{n}{m_a} - \frac{-n}{m_b} + \left(\frac{r_a \times n}{I_a} \right) \times r_a - \left(\frac{r_b \times n}{I_b} \right) \times r_b \quad (3.3)$$

Substituting the Δv and the effective mass formula, the full equation for a single collision between two objects becomes the collision impulse formula Equation 9 (Coumans, 2015;

Hecker, 1997):

$$p_{correction} = \frac{-\Delta v_{ab} \cdot n}{n \cdot n \left(\frac{1}{M_a} + \frac{1}{M_b} \right) + \left(\frac{r_a \times n}{I_a} \right) \times r_a - \left(\frac{r_b \times n}{I_b} \right) \times r_b} \quad (3.4)$$

Where Δv_{ab} is the change in relative velocity for both objects, M is the effective mass, r is the relative position vector of the contact point to the center of mass, n is the normal vector, I is the moment of inertia, and subscripts a and b denote the corresponding object (shape A and shape B respectively). For a full derivation of equations and calculations, please refer to references (Coumans, 2015; Hecker, 1997).

For the final step of the rigid body simulation loop, the bullet physics engine calculates the numerical time integration. The velocity and position of objects are updated by approximating the areas under the acceleration and velocity curve. Bullet uses the symplectic Euler algorithm, also known as semi-implicit Euler or semi-explicit Euler, to perform time integration of velocity because of its stability and preservation of energy without introducing artificial damping. Using Newton's and Euler's 2nd law to essentially describe the motion in rigid body dynamics, the symplectic Euler method updates the velocity and position of objects using Equation 3.5 and Equation 3.6, respectively (Coumans, 2015).

$$\begin{aligned} v_{t+\Delta t} &= v_t + a\Delta t = v_t + \frac{F_{ext} + F_c}{m}\Delta t \\ &= v_t + \frac{F_{ext}}{m}\Delta t + \frac{Impulse_c}{m} \end{aligned} \quad (3.5)$$

$$x_{t+\Delta t} = x_t + v_{t+\Delta t}\Delta t \quad (3.6)$$

Where v is the velocity, a is the acceleration, F_{ext} is the external forces, F_c is the constraint forces, $Impulse_c$ is the contact impulse, m is the mass of the object, Δt is the time step, and t is the current time.

3.1.3 Blender

The Bullet Physics engine (Coumans, 2005) is linked to Blender’s modeling space so users can control and activate features of the engine directly from Blender’s main toolbar to model RBD. For RBD, Blender classifies objects as either passive or static objects. When an object is designated as passive, they become solid objects fixed in place, while active objects are affected dynamically by collisions and gravity. Blender can handle a large number of objects for a simulation that can be either independent or interconnected with basic constraints. Rigid body physics can be applied to any geometry, and the physical simulation is directly played back in the Blender’s viewport (van Gumster, 2015).

Collisions and contacts in rigid body simulations are calculated using a collision surface mesh. This means that creating a composite spherical representation of the surface is no longer required, which is not the case for some DEM simulations (Bai et al., 2009; Kodam et al., 2010). Instead, the collision shape can be defined as either primitive shapes (such as a sphere or cylinder) or mesh-based shapes, such as convex hull or mesh. For convex hulls, the collision surface is calculated based on the geometry of the object, where a surface is created that encompasses all of the object’s vertices and forms a convex representation of the object. This convex approximation of the object has good computational performance and stability in rigid body simulations. For objects that have hollow regions, such as a Raschig ring or the container used to hold the particles for packing, the convex hull method provides an unrealistic solution. Instead, “Mesh” type shapes should be used

because they can handle both concave and convex geometries. Blender also allows users the ability to define collision margins that set a gap between the objects. This margin can have the shape of any of the primitive objects or convex hulls. It is important to note that a zero margin makes the calculations slower and less stable, but it is necessary to obtain a realistic packed bed (van Gumster, 2015).

3.1.4 Python Scripts

3.1.4.1 Packed Bed Generator Script

As previously stated, Blender allows users to utilize the python scripting language to create customized scripts to automate the 3D modeling process. Two packages were written to help with the generation of a packed bed. The first package was an adaption of the python package developed by Partopour and Dixon (2017) to automate the workflow of packed bed generation for v2.80 of Blender. The second python package is an original script developed to automate contact point modification based on proximity for any particle shape.

Partopour and Dixon (2017) python package “Packed Bed Generator (PBG)” was designed for v2.77 of Blender to receive the rigid body simulation parameters as inputs, generate the bed container, measure its properties, and export the geometry as an STL file. These input parameters used in the script include: (1) particle type, (2) particle and packed bed container dimensions, (3) number of particles, (4) friction factor, (5) restitution factor, (6) collision margin [[[see section Geometry Software: Blender page 13]]], (7) linear and rotational damping, (8) steps per second or the time step for the time loop and, (9) solver iteration. The types of particles the PBG can generate include

'sphere', 'cylinder', 'Raschig Ring', 'f-point-star', 'three-holes', 'four-holes', 'tri-lobes', 'quadrilobes', and 'four-hole-sphere'. The friction factor is the particle's resistance to change (aka the ratio of the friction force to the collision force between the colliding objects). In contrast, the restitution factor is the particle's tendency to bounce or the collision elasticity. The linear and rotational damping is the amount of velocity the particle loses over time.

Once these parameters are set in input files to the desired values, the package randomly places and orients particles into a designated 3-dimensional region above the container (tube) and drops them one at a time into the container for the initial time steps. Once the defined number of particles is reached, the fulling procedure stops, but the rigid body simulation continues until a steady-state is reached. When all the calculated particle linear and rotational velocities become less than the desired tolerance $[[0.005/\text{frame}]]$, the system is then considered to be steady, and the simulation is terminated, and the bed properties are calculated.

Partopour and Dixon (2017) provide a few modules to calculate different bed properties. Particle angle distribution is calculated by finding the angle between the normal vector of the particle's top face and the vector of the Z-direction of the container, storing values into a Python dictionary and returns the frequency of each angle interval. Radial voidage is calculated by radially separating the 3-dimensional domain of the packed bed into 100 cylindrical sub-surfaces, where each sub-surface is made up of hundreds of tiny squares. By using a ray-casting algorithm (based on the Jordan Curve Theorem), this method can quickly determine if the center of these squares are located inside or outside of the packing (Partopour and Dixon, 2017). When a ray is cast in a Euclidian space from a point in a fixed direction, if it intersects with a shape's edge an even number

of times, the point is considered to be outside of the object. In contrast, an odd number of intersections indicate that the point is inside the object. The radial voidage is then calculated for each of the radial sub-surfaces by summing the areas of squares located outside of the packing in that section, which is then divided by the surface area of that cylindrical sub-surface. This method of radial voidage is a fast calculation that works with all kinds of packing.

3.1.4.2 Contact Point Modifier Script

While the previous python script solved generating a packed bed, the following section focuses on a script that preserves the geometry of a bed packed with unique particles while preventing the formation of regions of low-quality mesh.

Initially, a python script was developed to detect and modify contact points of cylinders, as presented by Wehinger, Fütterer, and Kraume (2017). The authors use equations, derived from Kodam et al. (2010), to define contact detection, contact location, contact overlap, and normal contact direction between cylindrical objects. Unfortunately, the python code was unable to accurately use the equations from Wehinger, Fütterer, and Kraume (2017) to implement the bridge/caps method with the required precision.

Instead, a new python script was developed that modifies the particles based on their proximity to the other objects. One of Blender's built-in functions, called Modifiers, allows users to perform various automatic operations on an object's geometry in a non-destructive way that would otherwise be too tedious if executed manually. Essentially, these operations influence how an object is displayed and rendered, without affecting the base geometry until the modifier is applied, and the changes become permanent. When multiple modifiers are added to geometry, they are calculated/applied in order of

the “modifier stack” from top to bottom. Modifiers can be categorized into four types: modify, generate, deform, and simulate.

Two of the crucial functions used by my script are the “Vertex Weight Proximity (VWP) Modifier” and the “Simple Deform Modifier,” which are part of the “modify” and “deform” groups, respectively. The VWP Modifier modifies the weights of an object’s vertex group based on the distance between the source object and the target object. This modifier allows you to choose between two types of proximity modes. The First mode, called the object distance, computes the distance between the source object’s vertices and the target’s origin. The second mode is the geometry distance, where the modifier calculates the distance between the source’s vertices and the target object’s geometry, which can be its vertices, edges, or faces. Additionally, the user can control the upper and lower bounds of the mapped distances as well as the type of mapping (e.g., linear, custom curve, Sharp, Smooth, Root, Sphere, Random, or Median Step). For a full list of all the parameters of the VWP Modifier and their descriptions, please see [Blender Manual: Weight Proximity Modifier].

The Simple Deform Modifier can then use the object’s new weighted vertex group to define the influence of the deformation in addition to the type and direction of the deformation [For a full and list of all the parameters of the Simple Deform Modifier, please see [Blender Manual: Simple Deform Modifier]. The direction of the deformation is set by creating an object called an “Empty,” which is a single coordinate point with an orientation axis, whose primary purpose is to serve as a reference for position and orientation. The Empty’s z-axis can be constrained to track the origin of the target object while its coordinate point is set to the location of the source object’s origin. Additionally, the Simple Deform Modifier allows users to set the deformation factor, aka the amount of

deformation.

The following steps can summarize the python script: (1) the setting for mesh modification parameters; (2) compile a list of objects and their origins; (3) create list of all combinations and compute the squared Euclidean distance between the origins using scipy (Millman and Aivazis, 2011; Virtanen et al., 2020) and filter out distances that are too large and the repeated combinations; (4) begin mesh modification loop over the resulting list of objects; (5) the file is saved as a new file to avoid any loss of data.

The mesh modification loop entails: (i) defining the source object and the target object; (ii) creating a vertex group with the name of the target object, adding a VWP modifier using the previously created vertex group; (iii) create an Empty object by copying location of the source object and constraining z-axis to 'Track To' the target geometry location; (iv) create a simple deform modifier that uses the VWP modifier and the Empty object to deform the source object; (v) repeat steps (i-iv) but with the source and target geometries switched, before going onto the next pair of objects. Figure 3.2 shows the coloring based on proximity and the simple deformation transformation on the source object relative to the target object.

3.1.5 CFD Modeling of Packed Bed

After implementing the previously described scripts to generate a packed bed for CFD in Blender, bed geometry was imported into ANSYS SpaceClaim, where facets were cleaned and repaired before transfer to ANSYS Fluent Meshing for mesh generation. A variety of packed beds with different shapes were generated. Particle shapes were calculated keeping the particle equivalent diameter equal across all shapes. Shapes were also restricted to simple geometries that could be generated using Stop-Flow lithography

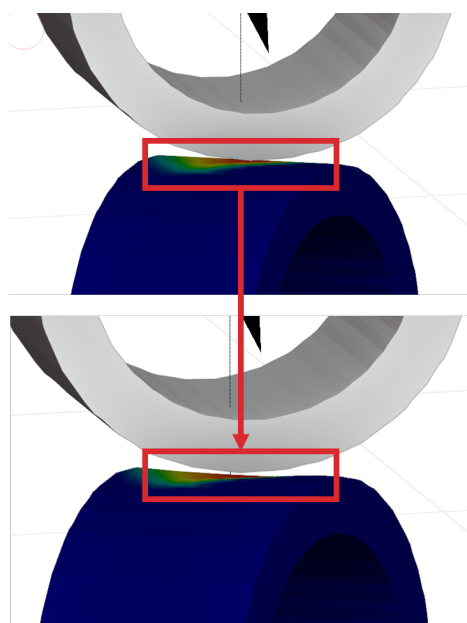
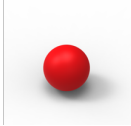

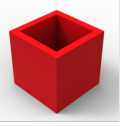
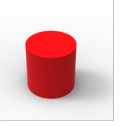

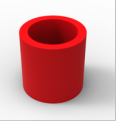


FIGURE 3.2: Contact modification based on proximity. The top picture shows the source object is colored by proximity, red indicates close proximity to the target, blue indicates low face proximity to the target object. The bottom shows the applied simple deformation on the source object with the previous proximity coloring.

fabrication technique (Panda et al., 2008). Refer to Table 3.1 for particle shape calculation parameters.

TABLE 3.1: Different particle shapes for bed packing

	Sphere	Cube	Hollow Cube	Cylinder	Hollow Cylinder (0.5)	Hollow Cylinder (0.75)
R (μm)	25.000	40.300	44.356	27.516	30.285	36.246
r (μm)					15.143	27.185
h (μm)				27.516	30.285	36.246
Thickness (μm)			11.089		15.143	9.062
Volume (μm^3)	65449	65449	65450	65449	65449	65450
Surface Area (μm^2)	7853.950	9744.404	14755.690	9514.383	12966.586	18057.117
Ratio Surface Area to Volume	0.120	0.149	0.225	0.145	0.198	0.276
Dp (μm) $\left(\sqrt[3]{\frac{6V}{\pi}}\right)$	50	50	50	50	50	50
Sphericity	1.000	0.806	0.532	0.825	0.606	0.435
						

Issues arose with both the geometry and meshing software due to the large number of faces. Additionally, it was deemed more important to explore the flow at the bead level with explicit bead porosity rather than a bed of particles. A packed bed that included an explicit bead porosity could not be generated due to the extreme number of faces as a result of combining both models.

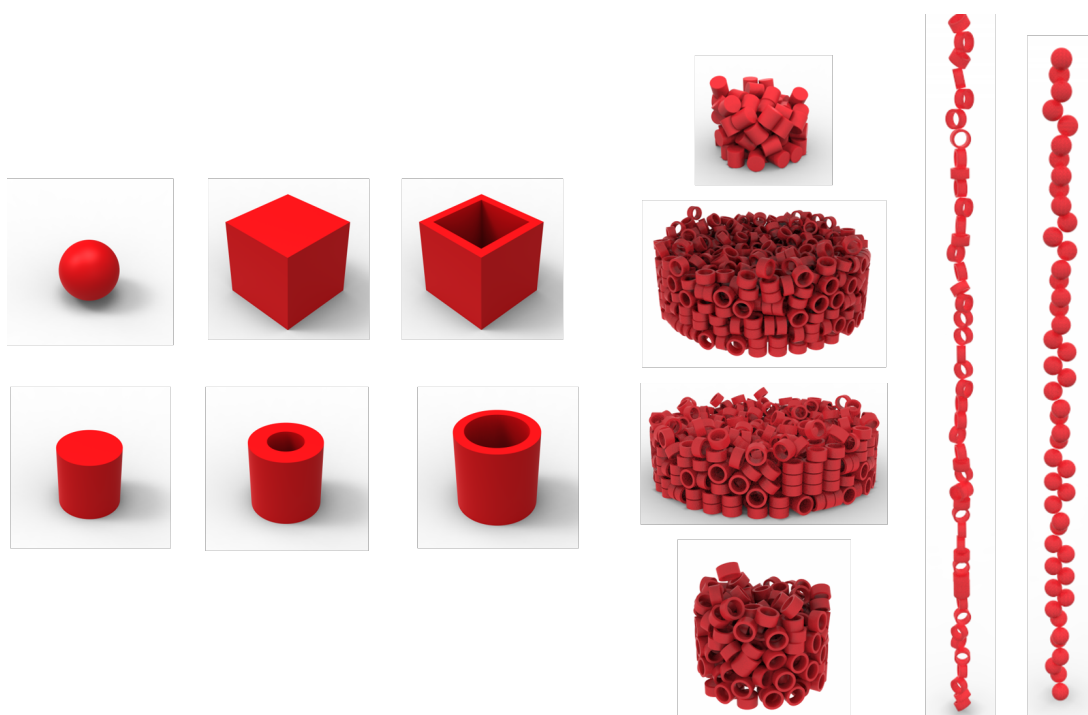


FIGURE 3.3: Results of bed packings for different beads shapes with different ratios of bed to particle diameters.

3.2 Porosity and Explicit Geometry Rendering

This study examined multiple porous geometries to determine the optimal rendering of porosity for CFD modeling perfusion resin loading. Because these perfusion resins have through-pores that capture part of the convective flow according to Afeyan et al. (1990), explicit geometry was used to model the porosity of the bead for CFD simulations.

One of the main factors in determining the porous rendering was the complexity of the geometry and the difficulty of meshing it. Meshing becomes increasingly difficult when microscopic renderings require multiple disconnected bodies or have sharp/right-angled structures. Porous geometries investigated include sphere clusters and 3D lattice infills generated using ANSYS SpaceClaim (ANSYS, Inc, 2019); see Table 3.2 for rendering and description of infill geometries considered.

For the sphere cluster rendering, porosity is captured by varying the number of small spherical particles (microparticles) distributed regularly inside a global sphere. Cluster porosity was defined using Equation 2.3. The flow-through particle porosity depends on the number of microparticles, the microparticle diameters, and the distance between the centers of two microparticles. Due to a large number of separated bodies required to mesh for the rendering of one porous bead, the clustered spheres approach was not selected for the model.

TABLE 3.2: Explicit pattern geometries considered for porous rendering. Geometries were generated using ANSYS SpaceClaim shell infill tool except for the Sphere/Ring Clustering, which was generated using Blender.

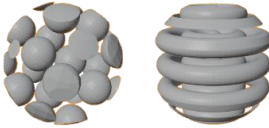

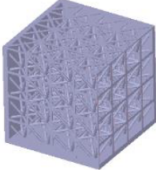

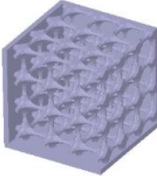

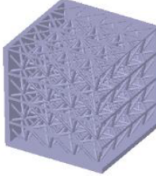

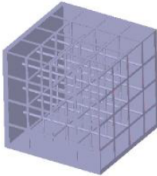

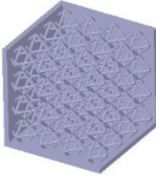

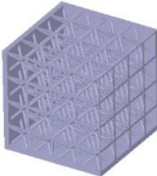

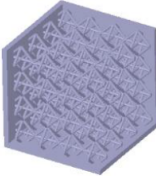
Description	Example	Description	Example
Sphere/Ring Clustering Spheres or Rings distributed regularly inside a global sphere, vary number to achieve the desired porosity. (Generated in Blender)		Cube Lattice With Bottom Center Cube lattice with supports that meet at the bottom face center of the enclosing cube 	
3D Lattice Three-dimensional lattice infill pattern 		Cube Lattice With Bottom Center Without Vertical Supports Cube lattice with supports that meet at the bottom face center of the enclosing cube but without vertical supports 	
Regular Cube Lattice Simple cube lattice infill pattern 		Double Pyramid Lattice Double pyramid lattice with lateral supports 	
Cube Lattice With Center Supports Cube lattice with supports from all corners of the cube intersecting at the center of the cube 		Double Pyramid Lattice With Cross Double pyramid lattice with internal cross supports 	

TABLE 3.2 Continued: Explicit pattern geometries considered for porous rendering.


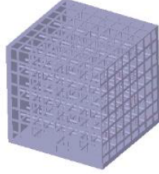

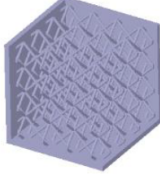

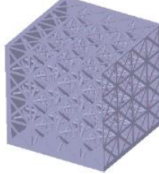

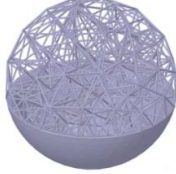

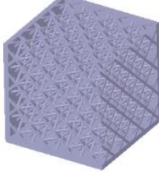

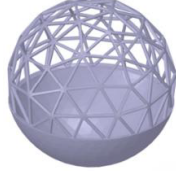

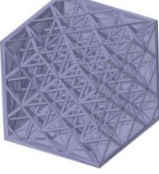

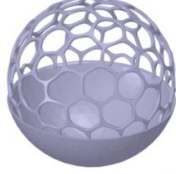
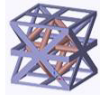
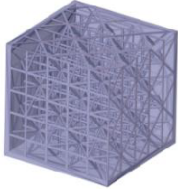
Description	Example	Description	Example
Cube Lattice With Side Cross Supports Cube lattice with cross supports between edges on each face of the enclosing cube 		Diamond Lattice Double pyramid without the lateral supports 	
Cube Lattice With Side Diagonal Supports Cube lattice with diagonal supports between vertices on each face of the enclosing cube 		Tetrahedral Faceted Lattice Boundary conforming lattice structure, where the lattice elements follow the edges of tetrahedra. 	
Double Pyramid Lattice And Face Diagonals Double pyramid in the center of the lattice with diagonals on faces of the enclosing cube 		Triangle Surface Faceted Lattice Uniform surface lattice, smoothly connected at the vertices with triangular holes. 	
Octahedral-1 		Hexagon Surface Faceted Lattice Smoothed Voronoi dual to the triangular surface mesh. Holes are on average hexagonal. 	

TABLE 3.2 Continued: Explicit pattern geometries considered for porous rendering.

Description	Example
Octahedral-2 	

Lattice structures provided a streamlined method of direct 3D porous rendering, however, close attention has to be paid to the lattice joints, since sharp joints can cause low mesh quality. The Shell tool in ANSYS SpaceClaim was originally intended for increasing the strength of 3D-printed objects by adding infill structures to faceted bodies. Refer to Table 3.2 for a detailed description of each of the lattice shell functions. Lattice structures considered from the Shell tool include lattice, regular cube lattice, cube lattice with center supports, cube lattice with side cross supports, cube lattice with side diagonal supports, cube lattice with bottom center, cube lattice with bottom center without vertical supports, double pyramid lattice, double pyramid lattice with cross, diamond lattice, double pyramid lattice and face diagonals, various octahedral configurations, and tetrahedral faceted lattice. The other lattice functions were not considered because of their adherence to the surface geometry and lack of internal structure that would mimic porosity. Of the lattice patterns, only the simple 3D lattice was suitable for meshing due to its rounded/ smoothed joints, since most of the other options were compromised with hard to mesh connected tubes with sharp joints.

Once the lattice pattern was chosen, parameters governing the porosity needed to be selected that captured a fluid flow similar to those of perfusion resins. The Thickness

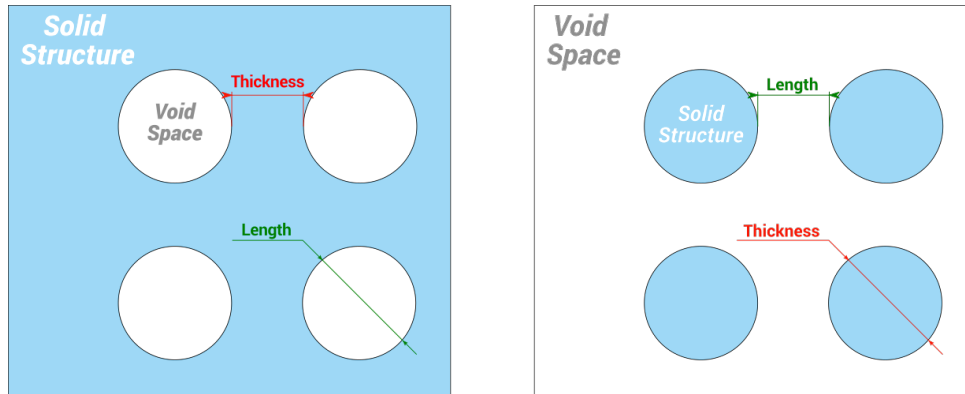


FIGURE 3.4: Depiction of the thickness and length lattice parameters for the Shell tool at different possible slices in the lattice. The blue shaded area represents the solid porous structure, while the white space represents the fluid void space.

and Length settings of the lattice function allow the user to set the diameter of the lattice and the void distance between the lattice structures, respectively. See Figure 3.4 for how the thickness and length parameters affect the lattice geometry. Thickness and Length setting were set equal to each other, and the parameters used for each case explored can be found in Table 3.3. Additionally, the Shell tool forms a thin wall, or shell, of specified thickness on either the outside or inside of the original body it was operating on. To avoid this undesired trait and open up the lattice structure to the fluid region, a shell of 0.1 cm (scaled, actual unit: μm) was formed on the outside of a sphere with radius 24.75 cm (μm). This shell was then removed using the Boolean intersection function of the previously mentioned sphere, and the resulting geometry was modified to remove unnecessary sharp edges or overhangs. Renderings of the geometry can be found in Table 3.3. the following sections expand on the geometry, meshing, and simulation setup for each of the lattice simulations.

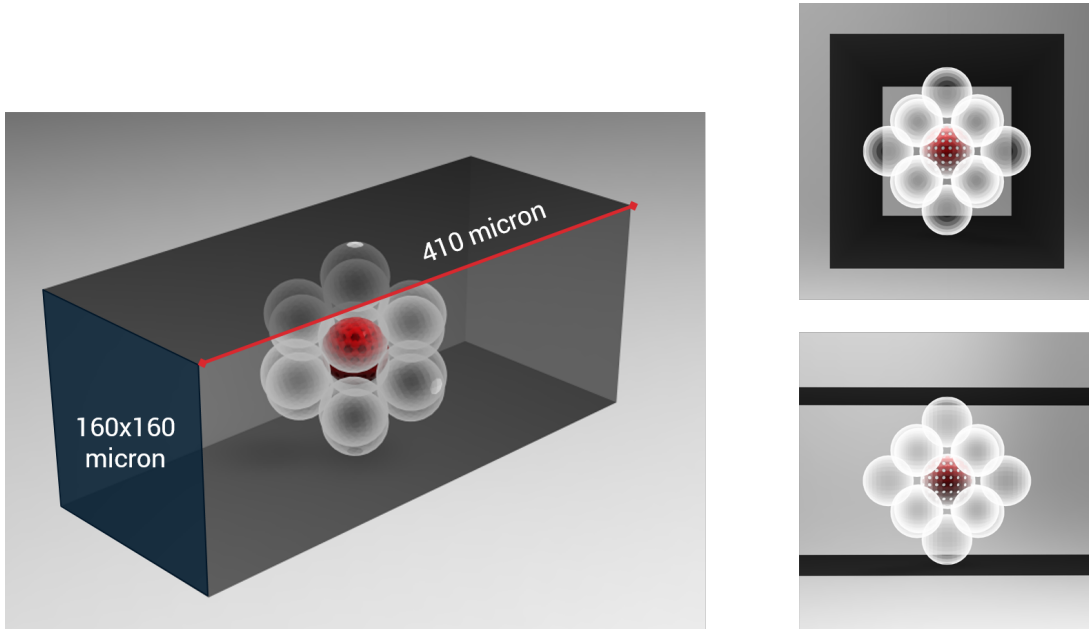


FIGURE 3.5: Flow cell geometry. The geometry consists of a single bead with explicit porous geometry surrounded (red bead) by 14 (white) non-porous spheres in a face centered cubic pattern to mimic a resin in a packed bed. The model is bounded by a rectangular prism with an inlet and outlet.

3.2.1 Case Setup for Different Porosity Settings (Geometry)

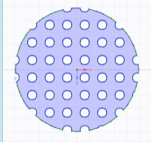
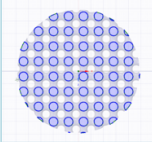
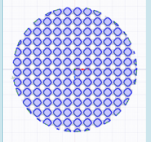
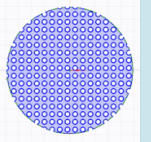
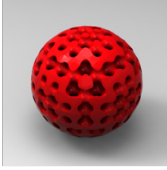
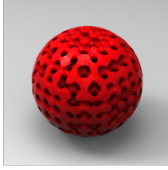
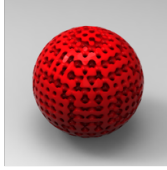
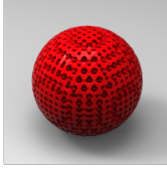
To simulate the flow around a perfusion resin particle with explicit geometric rendering of the through-pores, a parametric study of different porous geometries was conducted on a BOI to identify the best porous rendering for CFD modeling. Each explicit geometric rendering was split into two regions (an inner and outer part of the bead), and surrounded with 14 spheres to simulate flow in a packed bed and bounded by a rectangular prism to mimic a flow cell with a velocity inlet and pressure outlet. The following section goes into detail about how geometry and mesh were setup for CFD simulations. Refer to Figure 3.5 for a rendering of the flow cell geometry.

Each lattice structure was split into two different regions (an inner and outer

part of the bead) and surrounded by 14 spheres with $50\mu\text{m}$ diameters to simulate flow in a packed bed. These geometries were bounded by a rectangular prism to mimic a flow cell with a velocity inlet and pressure outlet. Additionally, the surrounding 17 beads were arranged in a body-centered cubic orientation to allow the BOI with the explicit geometric rendering to be at the center of the unit cell and fluid domain.

All geometries were created at 10^4 times the actual size to accurately capture geometry and mesh and avoid possible resolution errors from geometry and meshing software. The surrounding spheres and bounding flow-cell region were initially created using the open-source 3D computer graphics software Blender 2.80 (Blender Foundation, 2019). Contact points where two beads touch each other or where beads come into contact with a wall need to be flattened or removed to avoid known meshing problems of low cell quality as explained in Section 2.3.2. Beads were flattened locally at bead-bead contact points using the custom python script in Appendix B, while bead-wall contact points were merged with the flow-cell wall. See Section 3.1.4.2 for a description of the contact point modification script. The geometry was then transferred using an STL file into ANSYS SpaceClaim where facets were cleaned and repaired before generating the explicit pore geometry.

TABLE 3.3: Different BOI lattice geometries with their corresponding 2D slice and 3D renderings considered for explicit porous geometry rendering.

Thickness & Length	3.5 μm	3.0 μm	2.0 μm	1.5 μm
2D Slice				
3D Rendering				

The explicit pore geometry for the BOI was created using SpaceClaim's Shell fill feature. A lattice pattern with using the "3D lattice" pattern (from Table 3.2) with the length and thickness parameters in Table 3.3 was generated with a small outer shell. This outer shell was then removed using a Boolean intersection operation with a sphere of radius $25\mu\text{m}$ and the shelled geometry. The resulting geometry with exposed lattice structures was then smoothed and simplified to remove any unnecessary sharp overhangs. Final 3D geometries and corresponding 2D slices can be seen in Table 3.3.

3.2.2 Meshing

The geometry was then imported into ANSYS Fluent Meshing because of its ability to handle triangular surface mesh and generate high-quality polyhedral mesh quickly. Moreover, without compromising accuracy, polyhedral mesh requires less memory and provide faster solutions as compared with hexahedral and tetrahedral mesh elements due

to fewer cells. The imported geometry was remeshed using a curvature sizing method with a minimum size of 0.005, a maximum size of 0.1, a normal angle of 18° , and a growth rate of 1.3. The geometry was separated into two cell volume zones: the inner fluid region of the porous bead, and the outer region containing the inlet, outlet, and surrounding beads. All surrounding beads and the solid porous geometry of the BOI were labeled as empty or dead zones, with fixed positions to simplify the model and reduce the number of cells and complexity of the model.

In order to create an inner fluid region of the porous bead, a sphere with radius $23.8\text{ }\mu\text{m}$ (95% of BOI's radius) was created, and the pore geometry was subtracted from the sphere. The polyhedral mesh was then generated and using the previously mentioned sizing method. Cells and faces were automatically and manually smoothed and modified to improve mesh quality measures such as inverse orthogonal quality, FLUENT aspect ratio, size change, skewness, and warp. Once adequate values of the quality measures were reached, the mesh was transferred to the ANSYS Fluent Solution mode for the CFD simulations. See Table 4.2 for mesh statistics.

3.2.3 CFD Setup

For the CFD simulations, ANSYS Fluent 19R3 was used, which was a commercial CFD software based on the finite volume method. The mesh was scaled uniformly by 10^{-4} , so that bead radii were $25\text{ }\mu\text{m}$. The flow was assumed to be isothermal and incompressible with the physical properties of water. The interface between the solid and fluid phases, i.e., walls of the flow cell, surrounding beads, and BOI through-pore geometry were defined as 'no-slip' wall conditions, and the outlet was defined as a 'pressure-outlet' (0 bar). At the velocity-inlet, a constant velocity profile was set to 1000 cm h^{-1} .

First, the velocity profile of the system was solved using Fluent's pressure-based solver. The solver used the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithm to couple pressure and velocity. Additionally, the other numerical methods used to calculate the spatial discretization are: the standard scheme for pressure interpolation, the 2nd order upwind scheme for the momentum equations, the least-squares cell-based method for gradient evaluation, the Quadratic Upstream Interpolation for Convective Kinematics (QUICK) scheme for the energy equation, and a warped-face gradient-correction was enabled to improve the face gradient accuracy. The flow was considered to be laminar and steady-state.

All simulations were run using 16 processes on a computer with Intel® Xeon® Gold 6140 CPU at 2.30 GHz, 18 cores, and 32 GB of RAM. The steady-state calculations took 3-5 minutes and were transferred to ANSYS CFD-Post for post-processing.

3.3 DPM Flow Cell

3.3.1 Geometry

In order to simulate flow around a bead with through-pore geometry, a square flow cell with 14 spheres surrounding a bead with explicit pore geometry was created. This bead with explicit pore geometry will be referred to as the BOI. The surrounding beads have a radius of 25 μm and were arranged in a body-centered cubic orientation to allow the BOI to be at the center of the unit cell. The flow cell was broken up into three regions: the outer fluid region, an inner box region surrounding the bead of interest, and an inner bead region with 95% of the BOI particle radius .

Spheres and the flow cell domain were created at 10^4 times the actual size in order to accurately capture geometry and mesh and avoid possible rounding errors from geometry and meshing software. These geometries were created in Blender, then exported as an STL file into ANSYS SpaceClaim, where facets were cleaned and repaired. Beads were flattened locally at bead-bead contact points in order to avoid known meshing problems, while bead-wall contact points were merged with the wall. The explicit pore geometry for the BOI was created using the same methods described in Section 3.2.1. A velocity inlet, and pressure outlet boundary conditions were placed 2.07 m (unscaled) from the center of the BOI to minimize the influence of the boundary conditions on the region of interest, i.e., the BOI and surrounding beads.

In order to create an inner fluid region of the porous bead, a sphere with radius $23.8\text{ }\mu\text{m}$ (95% of BOI's radius) was created, and the pore geometry was subtracted from the sphere. The polyhedral mesh was then generated and using the previously mentioned sizing method. Cells and faces were automatically and manually smoothed and modified to improve mesh quality measures such as inverse orthogonal quality, FLUENT aspect ratio, size change, skewness, and warp. Once adequate values of the quality measures were reached, the mesh was transferred to the ANSYS Fluent Solution mode for the CFD simulations. See Table 4.2 for mesh statistics.

3.3.2 Mesh

The geometry was then imported into ANSYS Fluent Meshing because of its ability to handle triangular surface mesh and generate high-quality polyhedral mesh quickly. Moreover, without compromising accuracy, polyhedral mesh requires less memory and provide faster solutions as compared with hexahedral and tetrahedral mesh elements due

to fewer cells. The imported geometry was remeshed using a curvature sizing method with a minimum size of 0.005; maximum size of 0.1; normal angle of 18° ; and a growth rate of 1.3. The geometry was separated into three cell volume zones: (i) the inner fluid region of the porous bead, (ii) the box region capturing the fluid flow around the bead, and (iii) the outer region containing the inlet, outlet, and surrounding beads. All surrounding beads and the solid porous geometry of the BOI were labeled as empty or dead zones, with fixed positions to simplify the model and reduce the number of cells. In order to create an inner fluid region of the porous bead, a sphere with radius 95% of the bead was made, and the pore geometry was subtracted from the sphere. Polyhedral mesh was then generated and using the previously mentioned sizing method. Cells and faces were automatically and manually smoothed and modified to improve poor mesh quality measures such as inverse orthogonal quality, Fluent aspect ratio, size change, skewness, and warp. Once adequate values of the quality measures were reached, the mesh was transferred to the ANSYS Fluent Solution mode for the CFD simulations. See Table 4.2 for mesh statistics.

3.3.3 CFD Setup

For the CFD simulations, the finite volume method commercial software ANSYS Fluent[®] 19R3 was used (Inc., 2019b). The mesh was scaled uniformly by 10^{-4} , so that bead radii were 25 μm . The flow was assumed to be isothermal and incompressible with the physical properties of water. The interface between the solid and fluid (i.e., walls of the flow cell, surrounding beads and BOI throughpore geometry) were defined as ‘no-slip’ wall conditions, and the outlet was defined as a ‘pressure-outlet’. Multiple simulations were run where the velocity-inlet was set to constant velocity profile of 50 cm h^{-1} , 100 cm h^{-1} , 150 cm h^{-1} , 200 cm h^{-1} and 250 cm h^{-1} .

To simulate the capture and release of a tracer to and from the BOI, the model was broken up into three steps:

1. Calculation of the steady-state simulation for the velocity profile
2. Calculation of a transient simulation injecting a tracer from inlet using the DPM and capture/trap any particles that come into contact with the BOI using steady-state velocity profile, and
3. Calculation of a transient simulation releasing tracer from locations of the previous simulation and determine escaped residence time. These steps were performed for each of the inlet velocities.

First, the system's velocity profile was solved with Fluent's pressure-based solver before modeling the transient simulation. Fluent's solver used the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithm to couple pressure and velocity. Additionally, the other numerical methods used to calculate the spatial discretization were: the standard scheme for pressure interpolation, the 2nd order upwind scheme for the momentum equations, the least-squares cell-based method for gradient evaluation, the Quadratic Upstream Interpolation for Convective Kinematics (QUICK) scheme for the energy equation, and a warped-face gradient-correction was enabled to improve the face gradient accuracy. The flow was considered to be laminar and steady-state.

The simulation of the tracer's capture was subsequently performed by tracking transient transport of particles using the DPM, which follows an Euler-Lagrangian approach. This transient simulation was initialized with the values from the previously

calculated steady-state simulation. This simulation also used the same numerical methods for pressure-velocity coupling and spatial discretization schemes as previously mentioned for the steady-state case, but with a first-order implicit scheme for the temporal discretization. The DPM model solves energy and mass transfer equations with a point-particle approach. Particles have physical properties similar to IgG, i.e., density (ρ) = 1.410 g mL^{-1} and particle diameter (d_p) = 5.5 nm . Particles were injected from the inlet with the same velocity magnitude as the set inlet velocity using the inlet's normal direction vector. For the injection duration, 300 particles were injected from 300 starting points distributed randomly over the inlet boundary surface. Particles were injected over 33 particle time steps for a total of 10,000 particles injected into the domain. A UDF was written to set the number of particles per parcel, so only one IgG particle was present within a parcel. The model accounted for interaction with continuous phase by coupling calculations of the continuous and discrete phase flow. The DPM particles were tracked, and their sources were updated every 20 iterations of the continuous-phase calculation at the end of each time step.

Moreover, because this simulation was transient, the model performed unsteady particle tracking and updated the particle source term calculation every DPM iteration. Particles were injected at a particle time step that was 10x smaller than the fluid flow time step. The time step for the fluid flow was varied by inlet velocity and as seen in Table 3.4. The maximum number of steps tracked for the solution was 3,000,000, with a step length factor of 5.

The DPM in ANSYS Fluent uses a unique numerical and discretization scheme, which was different from the other numerics used by the program. The automatic tracking scheme, which switches between high order (trapezoidal) and low order (implicit) tracking

TABLE 3.4: Temporal parameters used for the transient DPM particle loading simulations for the 5 different inlet velocities.

Inlet Velocity	Particle Time Step (s)	Fluid Flow Time Step (s)	Injection length
50	1.000e-03	1.000e-01	3.330e-02
100	5.000e-04	5.000e-02	1.665e-02
150	3.333e-04	3.333e-02	1.110e-02
200	2.500e-04	2.500e-02	8.325e-03
250	2.000e-04	2.000e-02	6.660e-03

schemes based on the desired accuracy and stability range of each scheme, was selected. Accuracy control was enabled with a tolerance of $1e-05$ and a maximum number of 30 step size refinements in one single integration step. Further details on the definition can be found in the Fluent User Guide (ANSYS Inc., 2019c). The source terms for discrete phase momentum, energy, and species were linearized to allow the use of larger time steps.

Because IgG proteins are sub-micron, additional forces were incorporated into the model to simulate proper particle behavior. The Saffman lift force was included in the model as an extra force term because it accounts for the force that pushes sub-micron particles away from walls due to the shear stress field's presence (Li and Ahmadi, 1992). Brownian motion was also included in the model as an additional force term for sub-micron particles through the use of the Stokes-Cunningham drag law and Cunningham Correction factor of $C_c = 1.137129$. Gravity was neglected because the gravitational force acting on the particles was on the order of $10^{-22} \text{ kg m s}^{-2}$ and thus does not affect the particle's trajectory. A UDF was written to monitor the amount of shear stress particles experience along their trajectories based on the cells they passed through. The shear stress was integrated using the trapezoidal rule. See Appendix C for a breakdown of the code and calculation.

When DPM have reached a physical boundary, the particle's fate and its trajectory are determined by the discrete phase boundary condition set by the model in Fluent. The boundary can either reflect, pass through, escape or trap particles. Particles were ideally reflected at the walls and inlet boundary conditions. For the internal boundary conditions, particles passed through unimpeded. Particles were trapped on the inner and outer surfaces of the BOI's explicit porous geometry renderings. Particles that passed through the outlet were designated as escaped. The surface of the BOI, as well as the outlet, were monitored using Fluent's sampling of trajectories discrete phase report. This report allowed particle quantities to be written to a file for each of the boundary conditions for later analysis. A UDF was written to customize values saved to the files and macros were written to control the frequency at which these files were written. Values recorded in the report for both the trapped and escaped particle include: coordinates of trapped location (xyz), velocity vectors (uvw), diameter, temperature, parcel-mass, number of particles in parcel, residence time, flow time, injection time, and the shear stress integral.

For the "release" step simulation, coordinate values from the DPM sampling report were used to set the injection location of particles. Because of a slight rounding error, a UDF was written to relocate particle injection sites to the closest cell centroid of the mesh. Differences in location were on the order of 10^{-14} m. All particles were injected into the domain in one-time step, and the maximum number of steps tracked for the solution was 9,000,000 with a step length factor of 5. Transient simulations were then run until all particles escaped through the outlet boundary, and particle quantities were recorded using the previously mentioned discrete phase sampling report UDF and macros. This model used the previously calculated steady-state to set the initial condition of the

fluid flow to reduce fluid flow calculation time.

All simulations were run using 16 processes on a computer with Intel® Xeon® Gold 6140 CPU at 2.30 GHz, 18 cores, and 32 GB of RAM. The steady-state calculations took 3-5 minutes, and the transient calculation took about 2-5 days for each part. The Discrete Phase Model used a hybrid method for parallel DPM tracking that combines message passing and OpenMP dynamic load balancing; further details can be found in the Fluent User's Guide (ANSYS Inc., 2019c)

Chapter 4

Results and Discussion

4.1 Steady State Models

A parametric study of steady-state flow simulations for flow around a perfusion resin (BOI) with different explicit through-pores geometries was conducted to identify the best porous rendering for CFD modeling. Each explicit geometric rendering BOI was split into two regions (an inner and outer part of the bead) and surrounded with 14 spheres (arranged in a body-centered cubic orientation with $50\text{ }\mu\text{m}$ diameters) to simulate flow in a packed bed. These beads were bounded by a rectangular prism to mimic a flow cell's fluid domain with a velocity inlet and pressure outlet.

CFD steady-state simulations were meshed and solved using ANSYS Fluent 19R3. The polyhedral mesh was scaled uniformly by 10^{-4} , so that bead radii are $25\text{ }\mu\text{m}$. The flow is assumed to be isothermal and incompressible with the physical properties of water. The interface between the solid and fluid phases, i.e., walls of the flow cell, surrounding beads, and BOI through-pore geometry, are defined as 'no-slip' wall conditions,

TABLE 4.1: Mesh Independence

Model	number of cells resin	number of cells fluid	cell volume ave resin	cell volume max resin	cell volume min resin
Model 1	842506	682016	$1.978 \times 10^{-1} \mu\text{m}^3$	$8.738 \times 10^{-1} \mu\text{m}^3$	$2.202 \times 10^{-5} \mu\text{m}^3$
Model 2	5512810	4220680	$3.017 \times 10^{-2} \mu\text{m}^3$	$1.780 \times 10^{-1} \mu\text{m}^3$	$2.301 \times 10^{-5} \mu\text{m}^3$

and the outlet is defined as a 'pressure-outlet' (0 bar). At the velocity-inlet boundary, a constant velocity profile is set to 50 cm h^{-1} , 100 cm h^{-1} , 150 cm h^{-1} , 200 cm h^{-1} , 250 cm h^{-1} , 300 cm h^{-1} and 1000 cm h^{-1} . Results were post-processed with ANSYS CFD-POST (Inc., 2019a).

Because biomolecules can be extremely shear sensitive, I investigated the shear profiles within the model. Figure 4.1(a) shows the location of two CFD-Post volume probes inserted in CFD-Post to look at the shear stress distribution inside the BOI region (red) and the packed bed region (blue). Shear stress was calculated using Equation 2.30, where μ is the molecular viscosity ($\text{kg/m}\cdot\text{s}$) and $\dot{\gamma}$ is the strain rate ($1/\text{s}$). Figures 4.1(b-h) show charts of the shear stress profile along the (with the corresponding colors previously mentioned) at inlet velocities of 50 cm h^{-1} , 100 cm h^{-1} , 150 cm h^{-1} , 200 cm h^{-1} , 250 cm h^{-1} , 300 cm h^{-1} and 1000 cm h^{-1} . These simulations confer that within practical operating conditions of flow rate, the shear stresses present were low. Given that large, shear sensitive bioparticles, such as the measles virus, begin to degrade around 0.25 Pa , chromatography columns would have to operate at 250 cm h^{-1} or above before seeing that level of shear in a packed bed (Grein et al., 2019).

Figure 4.2 shows the velocity vector profile in the YZ-plane at $X = 0$ across the entire model for different inlet velocities (50 , 100 , 150 , 200 , 250 , 300 and 1000 cm h^{-1}). Surrounding beads and the BOI's explicit geometry are rendered as transparent surfaces for $X < 0$ for contextual orientation and visualization. The sub-figures correspond to the different set inlet velocity profiles. Figure 4.3 is the same velocity vector rendering as

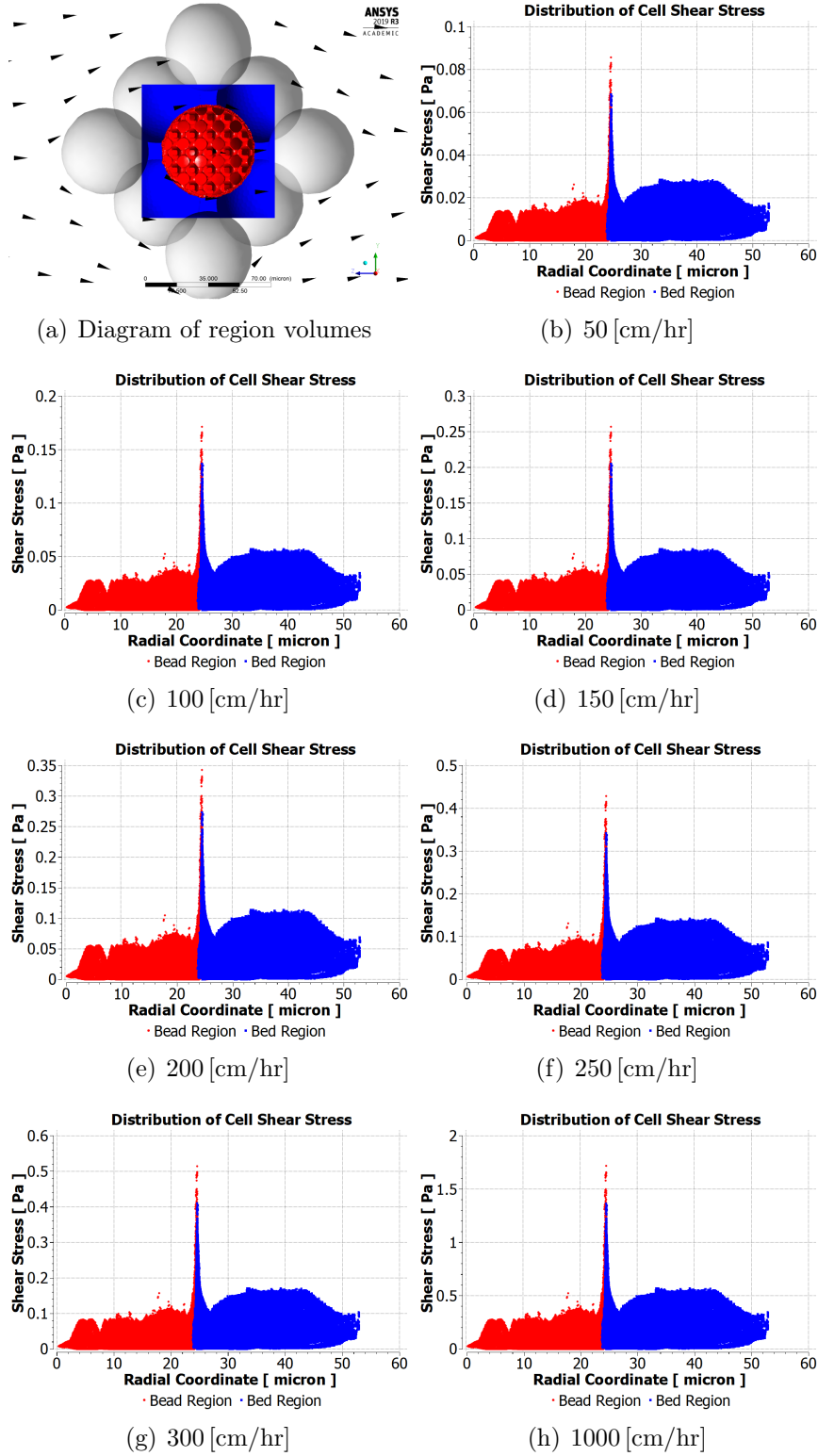


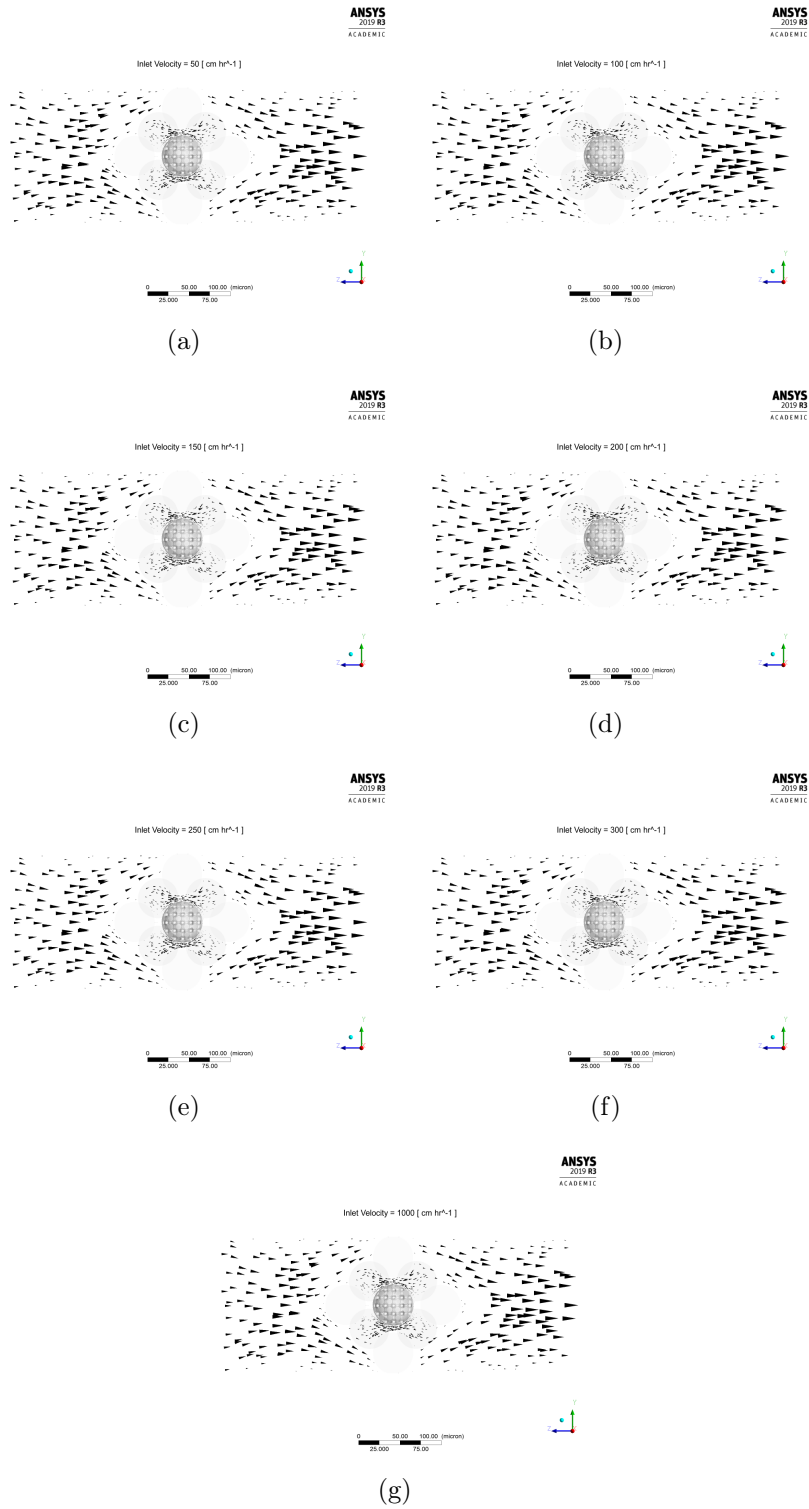
FIGURE 4.1: Shear stress distribution of mesh. (a) shows location of two volume probes, red corresponds to the BOI resin volume and blue corresponds to the packed bed region bounded to the , colors correspond to chart line colors in (b-h).

Figure 4.2 but zoomed in to the BOI and surrounding beads, and Figure 4.4 is zoomed in even further to a fraction of the BOI and some of the surrounding beads.

Figure 4.5 and Figure 4.6 are the same viewpoints, geometry renderings and vector profile renderings as Figure 4.3 and Figure 4.4 (respectively) but with the addition of velocity magnitude contours in the YZ-plane at $X = 0$.

Figure 4.7 shows the volume rendering of shear stress in the fluid domain for inlet velocities (50, 100, 150, 200, 250, 300 and 1000 cm h^{-1}). This figure shows the location of regions of high shear stress in the domain. Figure 4.8 shows the shear stress contours in the YZ-plane at $X = 0$ at with a similar viewpoint and geometry rendering as Figure 4.3 for the inlet velocities (50, 100, 150, 200, 250, 300 and 1000 cm h^{-1}). Figure 4.7 and Figure 4.8 show the low range of shear stress in the fluid domain for the inlet velocities (50, 100, 150, 200, 250, 300 and 1000 cm h^{-1}).

In addition to looking at the shear stress in the fluid, I also looked at wall shear stress. Figure 4.9 shows the wall shear stress profile on the walls of the 17 surrounding beads and the BOI. The maximum value of wall shear stress is 1.558 Pa when the inlet velocity is 1000 cm h^{-1} . Figure 4.10 is a zoomed-in view of the wall shear contours with black velocity vectors showing the direction of flow around the BOI. Figure 4.11 is the same wall shear contours and velocity vectors, but with the point of view on the positive X-axis looking at the origin with the contour surfaces and velocity vectors rendered for $X < 0$.

FIGURE 4.2: Velocity Vectors in YZ-plane at $X = 0$.

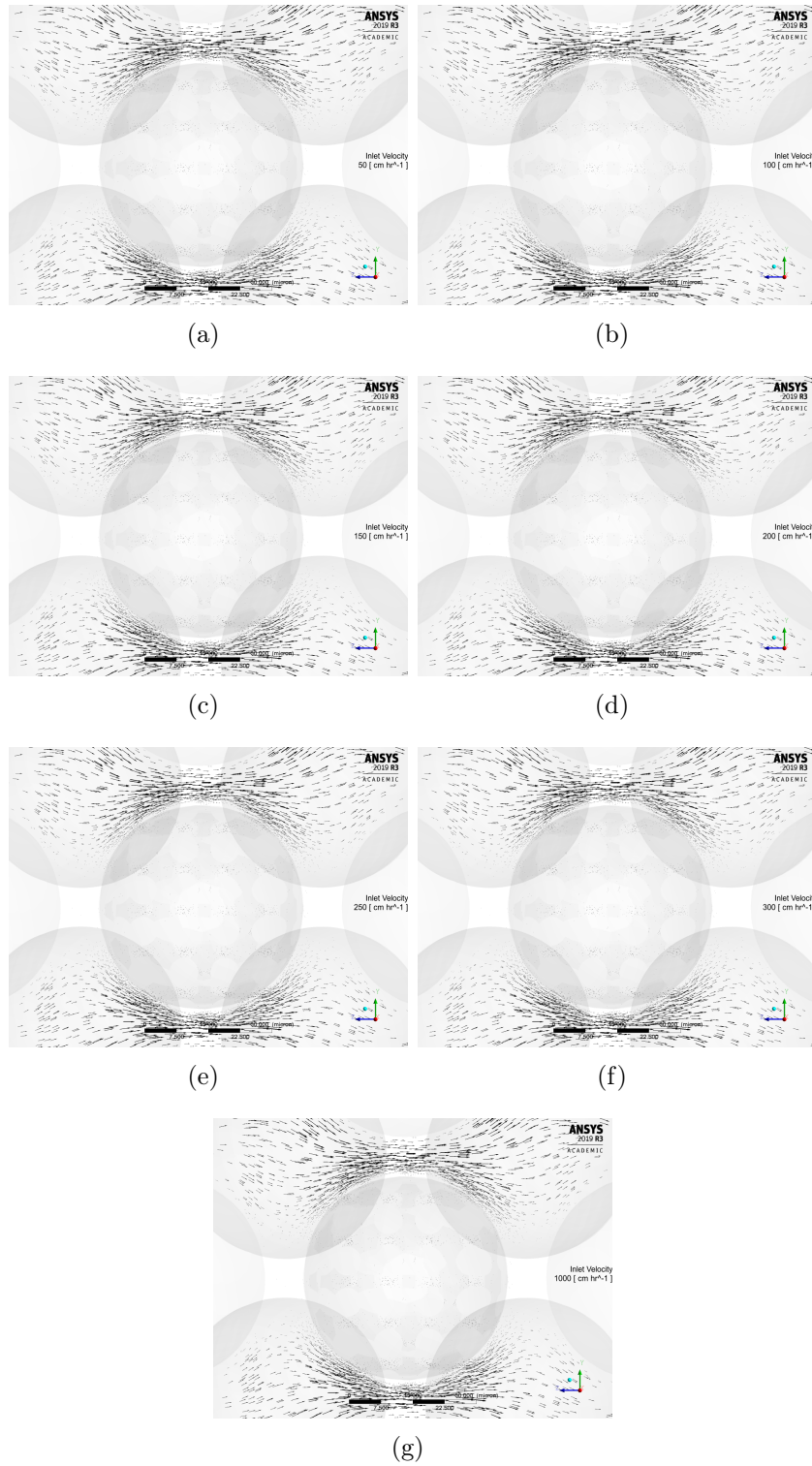


FIGURE 4.3: Close up view of velocity vectors form Figure 4.2 YZ-plane at $X = 0$.

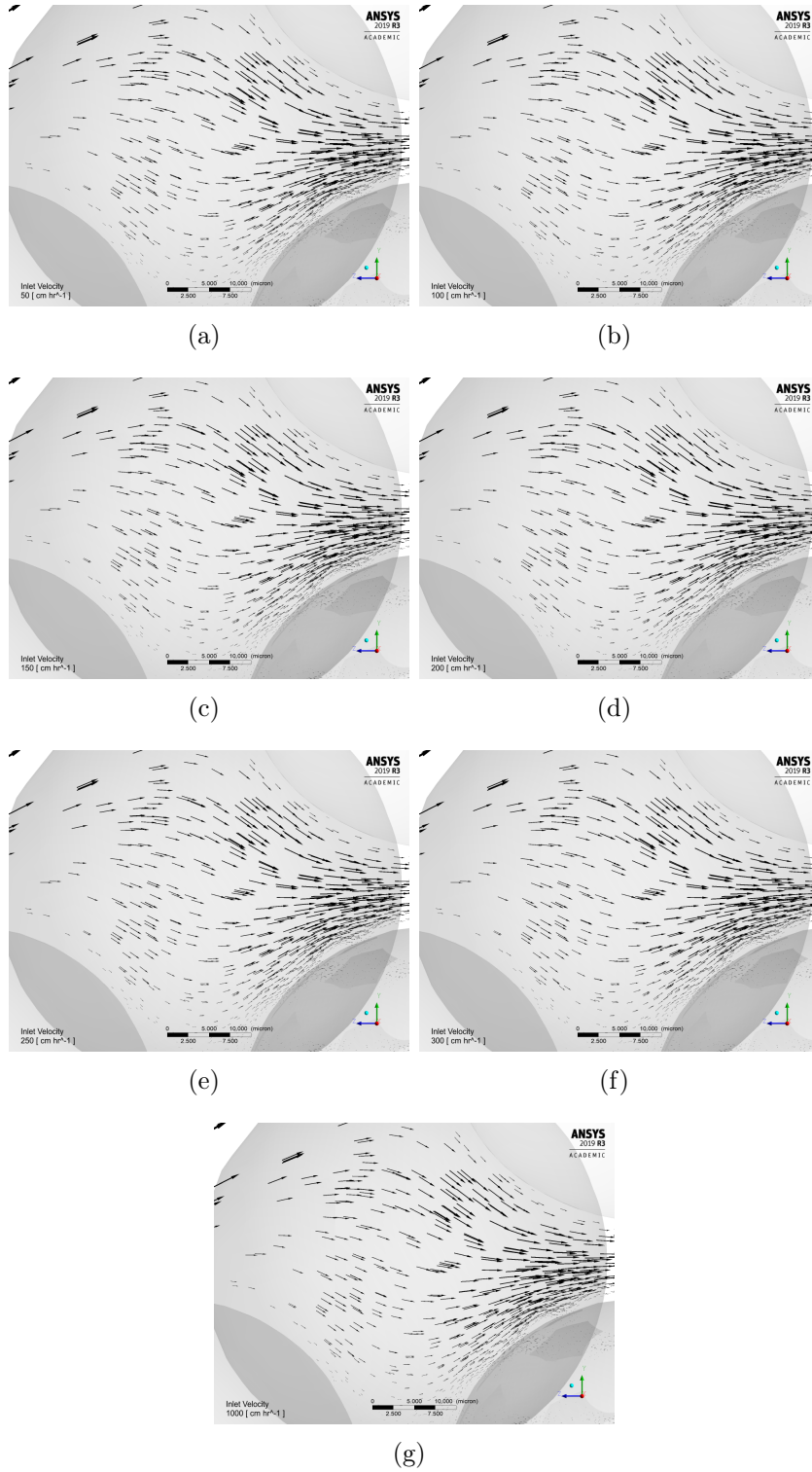


FIGURE 4.4: Closer view of velocity vectors between beads form Figure 4.2 YZ-plane at $X = 0$.

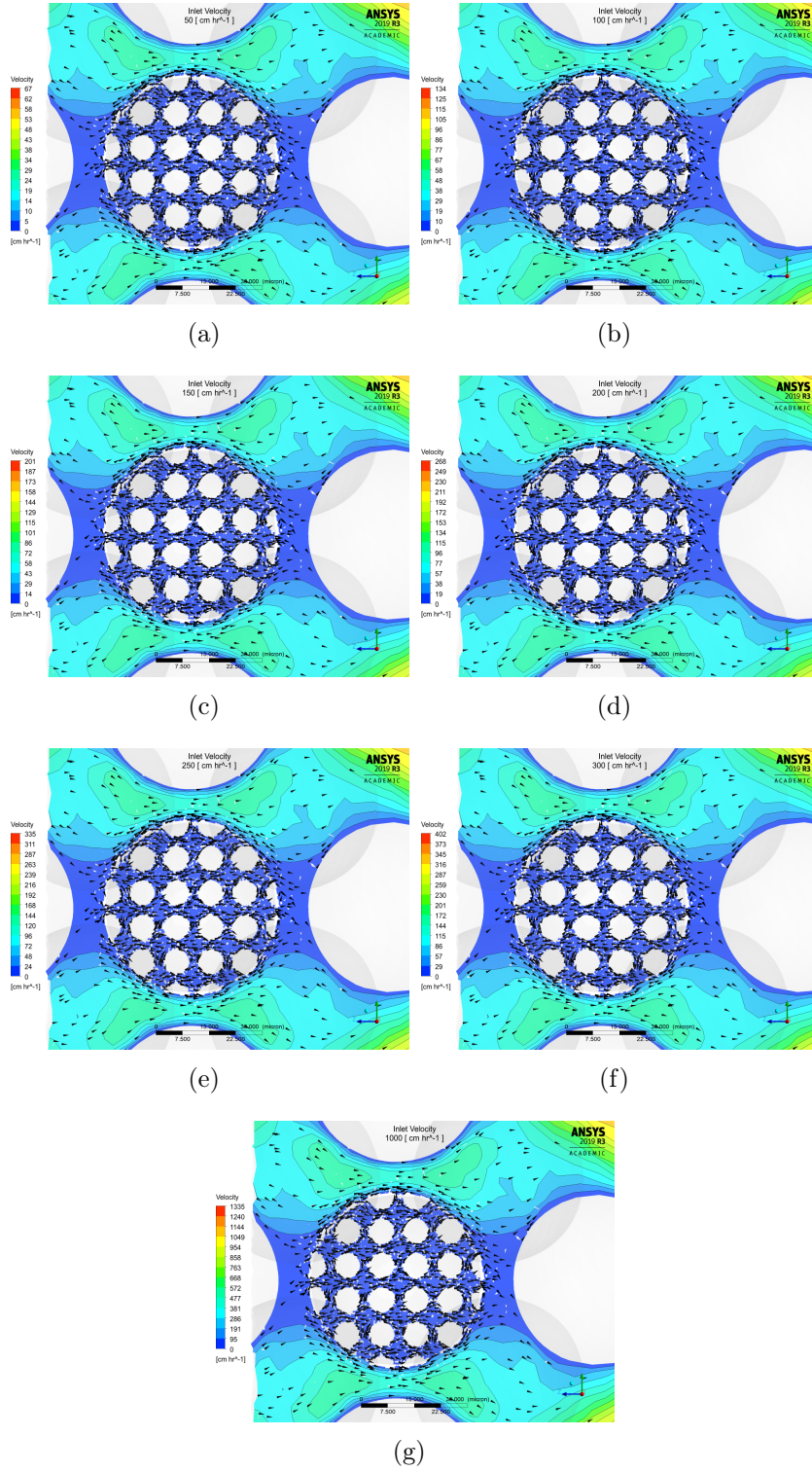


FIGURE 4.5: Velocity vectors with contours of the velocity magnitude form Figure 4.2 YZ-plane at $X = 0$.

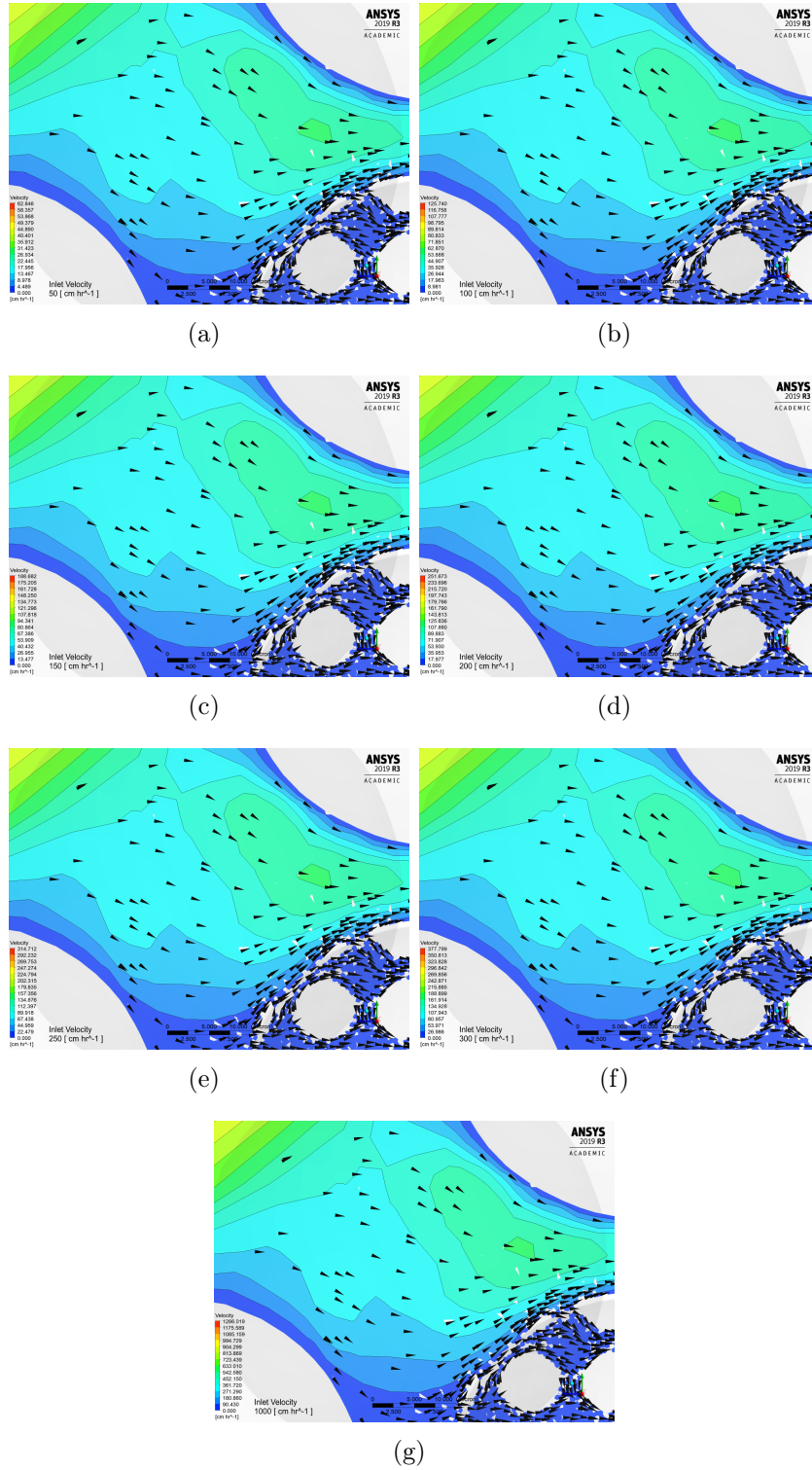


FIGURE 4.6: Closer view of velocity vectors between beads along with the contours of the velocity magnitude on YZ-plane at $X = 0$.

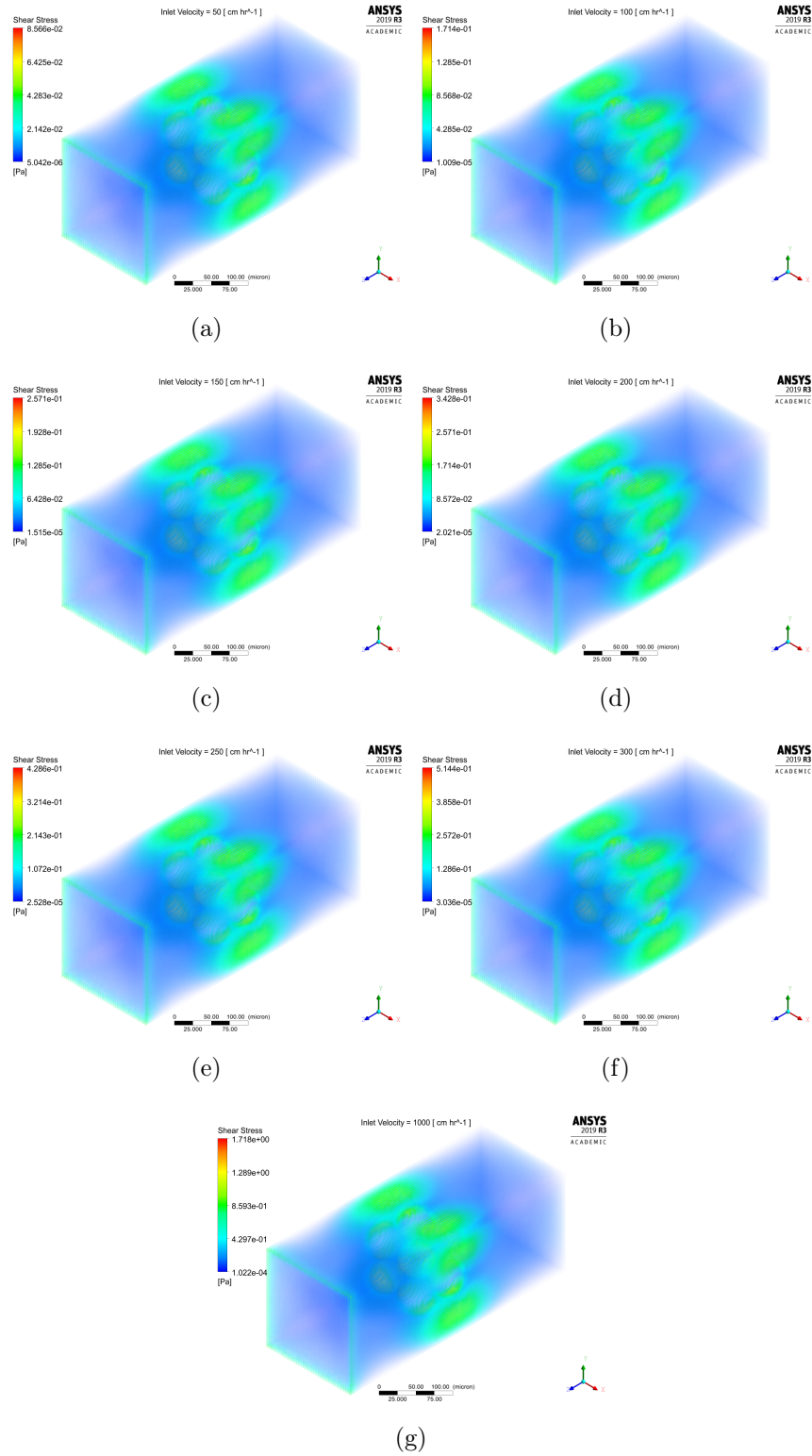


FIGURE 4.7: Volume rendering of shear stress in the fluid domain for steady-state simulations with inlet velocities of 50, 100, 150, 200, 250, 300 and 1000 cm h^{-1} .

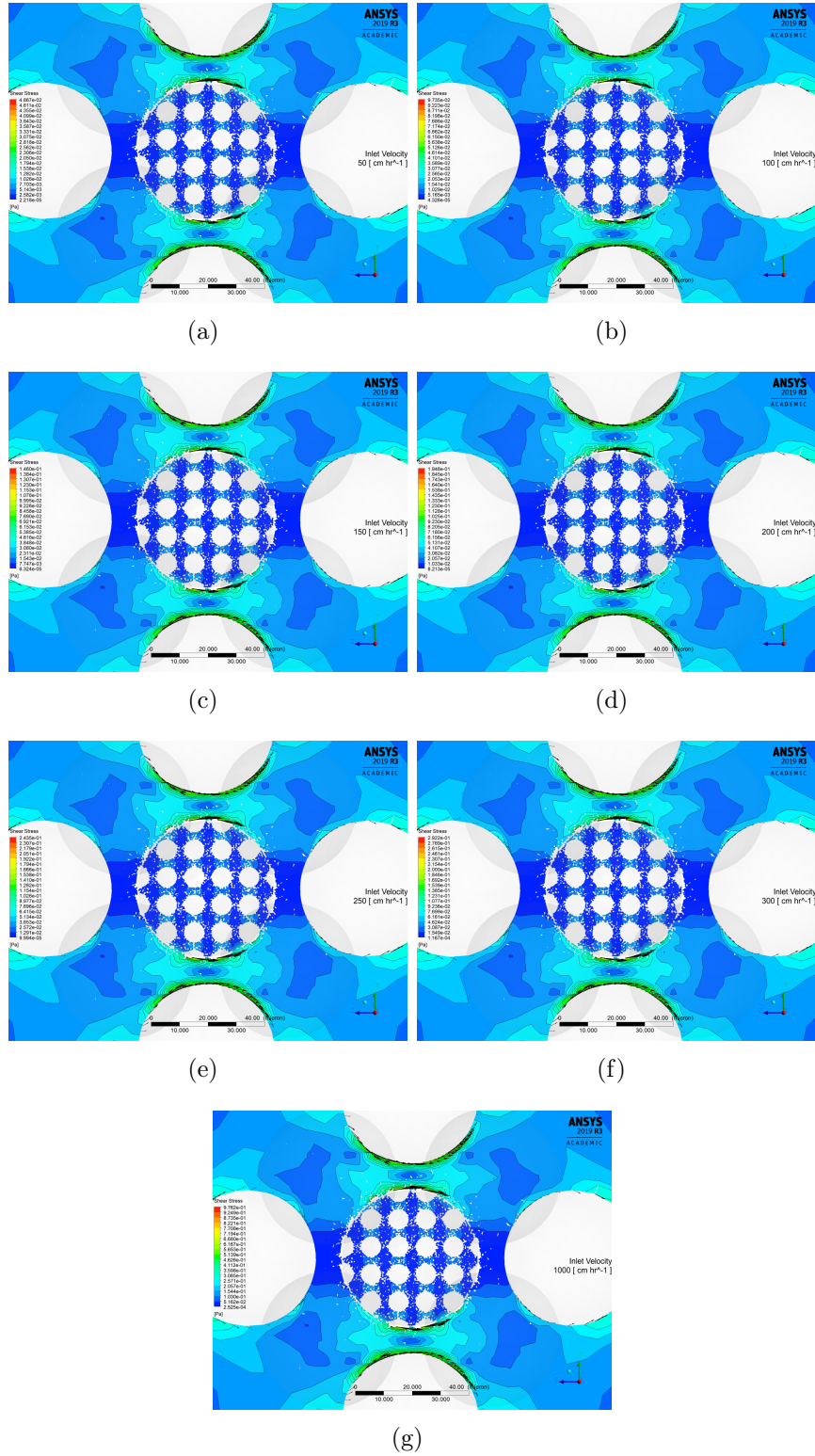


FIGURE 4.8: Contours of shear stress on YZ-plane at $X = 0$ for steady-state simulations with inlet velocities of 50, 100, 150, 200, 250, 300 and 1000 cm h^{-1} .

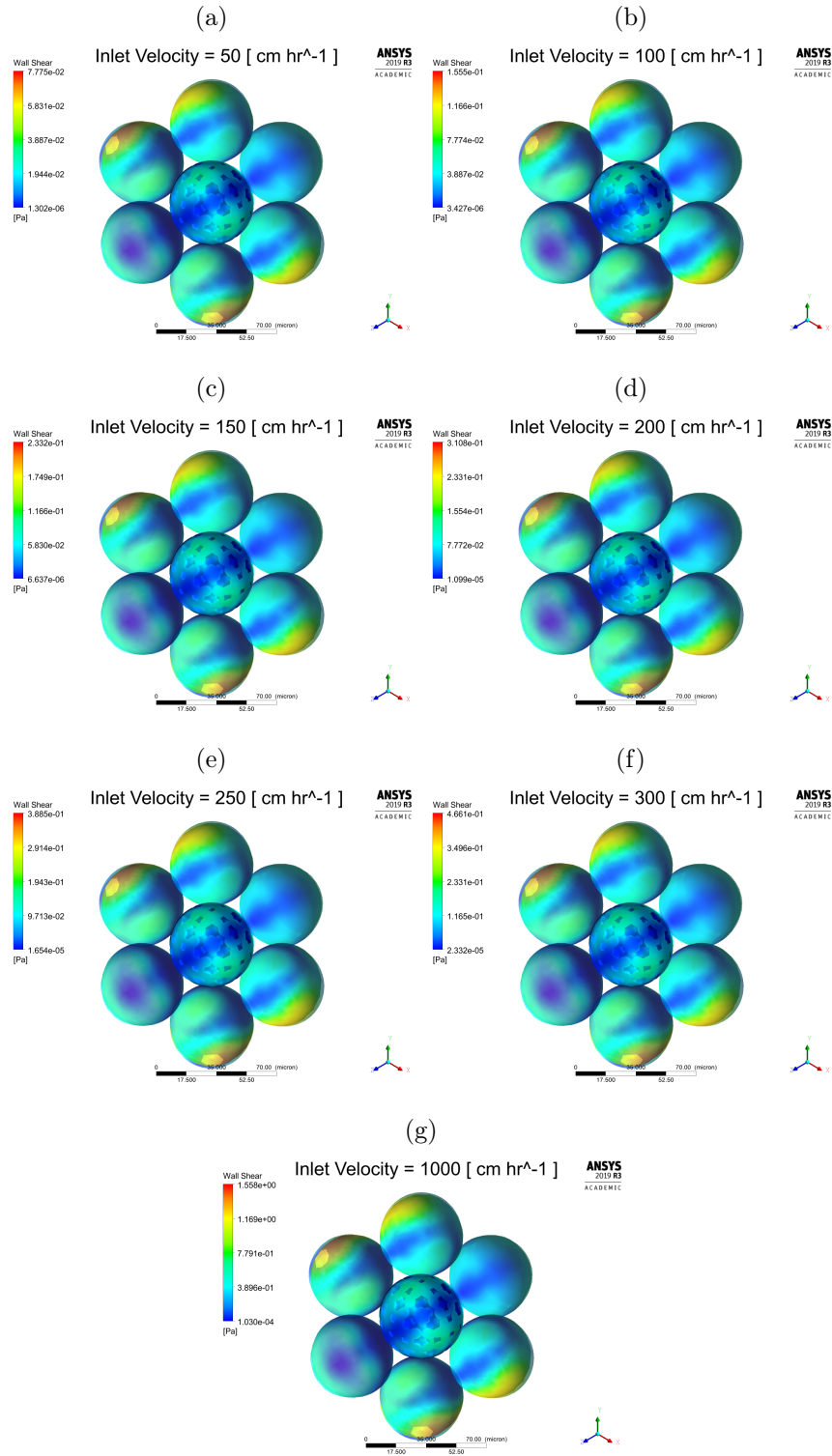


FIGURE 4.9: Contours of wall shear on the surrounding beads as well as on the explicit geometry of the BOI.

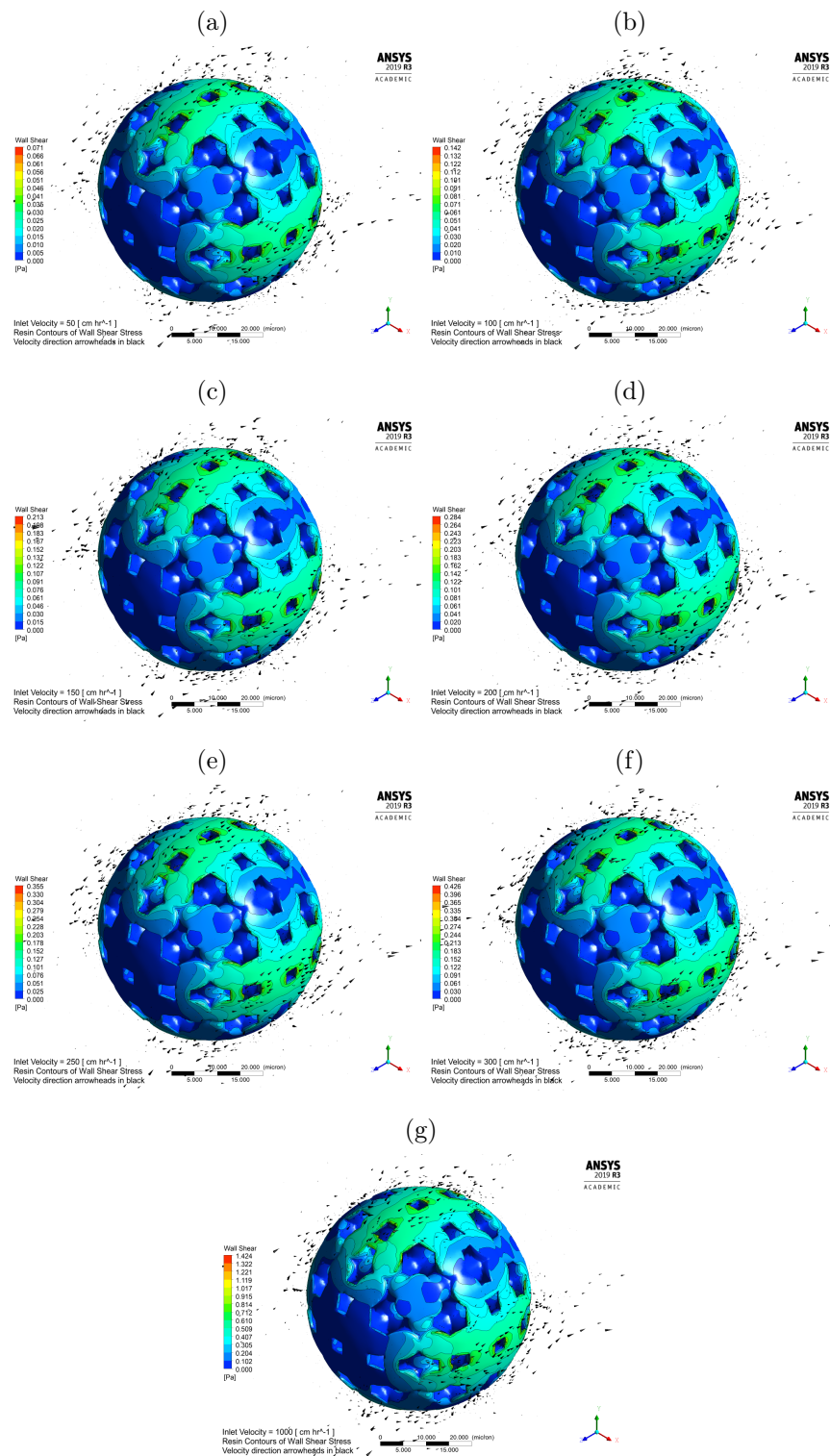


FIGURE 4.10: Closeup of contours of wall shear stress on explicit geometry of the BOI with velocity vectors in black.

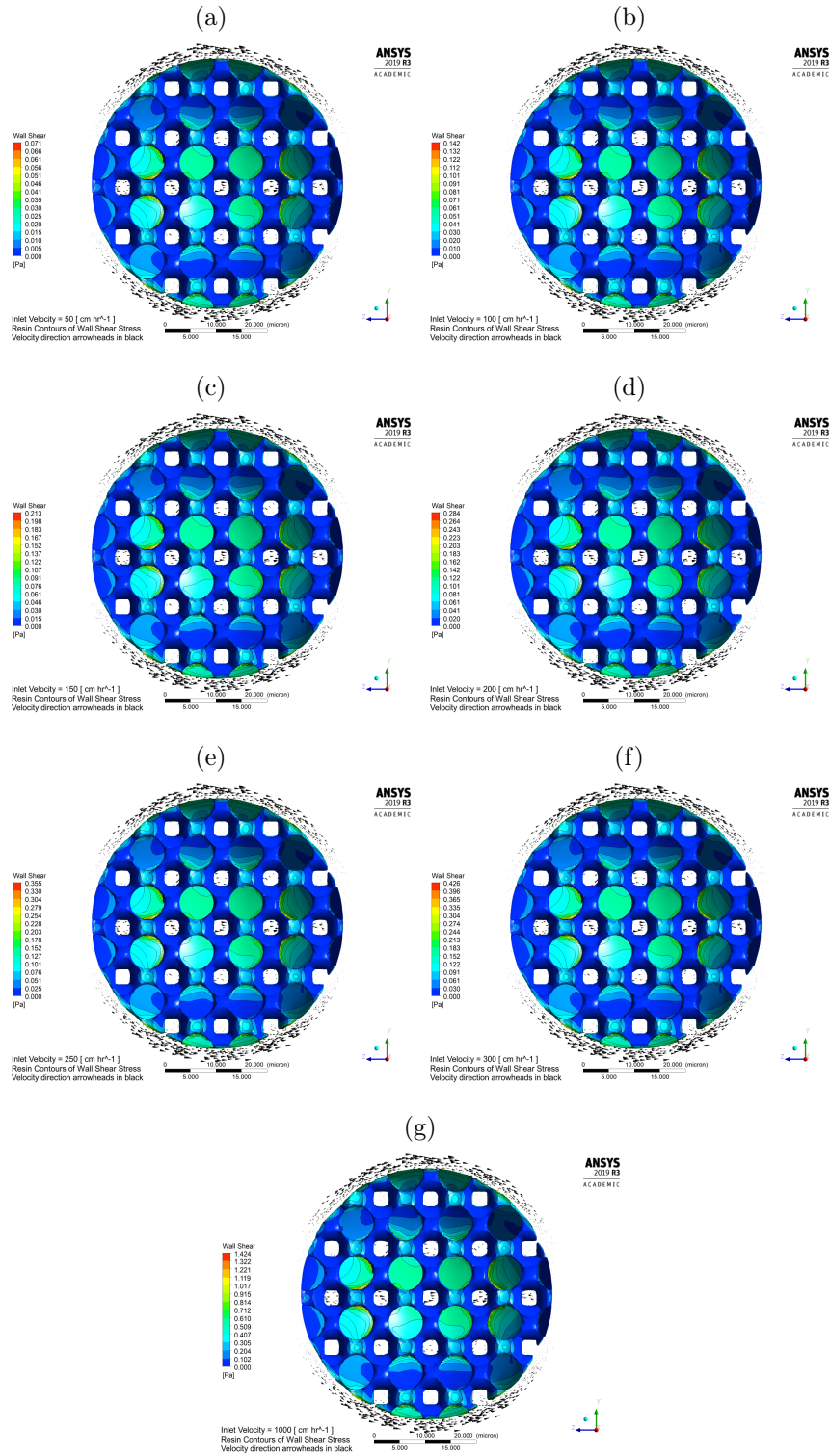


FIGURE 4.11: Sliced view of contours of wall shear stress on explicit geometry of the BOI with velocity vectors in black.

4.1.1 Pore Lattice Exploration

Afeyan et al. (1990) and Rodrigues et al. (1992) suggested that a flow-through particle's permeability could be used to estimate the ratio of intra-particle to outer particle mobile phase velocity using the following relationship:

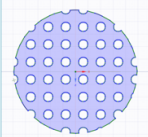
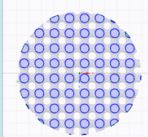
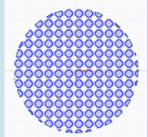
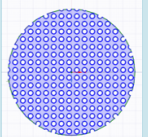
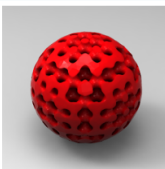

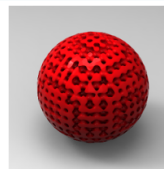
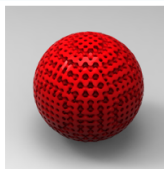
$$\frac{u_{part}}{u_i} = \frac{K_{vpart}}{K_{vbed}} \frac{1 - \epsilon_{bed}}{\epsilon_{part}} \quad (4.1)$$

where u_{part} is the velocity in the flow through region, u_i is the interstitial pore velocity, K_{vpart} is the permeability of the flow-through particle and K_{vbed} is the bed permeability if the particles would be fully porous. Komiyama and Inoue (1974) suggested a simpler equation for low Reynolds numbers.

$$\frac{u_{part}}{u_i} = \frac{K_{vpart}}{K_{vbed}} \quad (4.2)$$

Afeyan et al. (1990) estimated the intra-particle velocity of the commercially available POROS[®] particles to be about 5% of the superficial mobile phase velocity. The use of CFD allows for modeling fluid flow through and around these beads without making assumptions about the particle's permeability. The ratio of intra-particle to outer particle fluid velocity was calculated using the volume-weighted average of the bead's inner fluid velocity and the volume-weighted average of the interstitial velocity. Length/thickness of the resin lattice was varied and the corresponding computational mesh size and density can be found in Table 4.2. A lattice with a length/thickness between 3.0 μm and 3.5 μm could achieve a similar percent of the superficial mobile phase velocity as estimated by Afeyan et al. (1990).

TABLE 4.2: Different BOI lattice geometries with their corresponding 2D slice and 3D renderings considered for explicit porous geometry rendering.

Thickness & Length	3.5 μm	3.0 μm	2.0 μm	1.5 μm
2D Slice				
3D Rendering				
Number of nodes resin	1.408e+6	1.679e+6	2.229e+6	8.425e+6
Number of nodes fluid	8.146e+5	8.616e+5	9.963e+5	3.466e+6
Resin Cell volume average	1.483e-1 [micron ³]	1.222e-1 [micron ³]	8.688e-2 [micron ³]	2.117e-2 [micron ³]
Fluid Cell volume average	5.039e+0 [micron ³]	4.848e+0 [micron ³]	4.768e+0 [micron ³]	1.721e+0 [micron ³]
Resin cell volume max	5.541e-1 [micron ³]	4.555e-1 [micron ³]	2.179e-1 [micron ³]	5.764e-2 [micron ³]
Fluid cell volume max	2.129e+1 [micron ³]	1.904e+1 [micron ³]	1.987e+1 [micron ³]	8.347e+0 [micron ³]
Resin cell volume min	1.337e-5 [micron ³]	1.900e-4 [micron ³]	4.730e-6 [micron ³]	2.763e-5 [micron ³]
Fluid cell volume min	4.333e-5 [micron ³]	3.364e-5 [micron ³]	2.171e-6 [micron ³]	2.712e-6 [micron ³]
Velocity Average Resin	1.539e+1 [cm hr ⁻¹]	1.216e+1 [cm hr ⁻¹]	5.712e+0 [cm hr ⁻¹]	3.076e+0 [cm hr ⁻¹]
Velocity Average Fluid	1.874e+2 [cm hr ⁻¹]	1.873e+2 [cm hr ⁻¹]	1.844e+2 [cm hr ⁻¹]	1.817e+2 [cm hr ⁻¹]
Percent of fluid flow into resin	8.211	6.489	3.097	1.693

Péclet cell numbers were also calculated for each of the lattice structures using the approximate kinematic viscosity of water ($1 \times 10^6 \mu\text{m}^2 \text{s}^{-1}$) and the diffusion coefficient of IgG ($40 \mu\text{m}^2 \text{s}^{-1}$ in a dilute solution at 25°C) (Wrzosek et al., 2013; Young, Carroad, and Bell, 1980). In Figure 4.12, the relationship between the cell Péclet number distribution and the radial coordinate can be seen for the various 3D pore lattice structures. Lattice structures with length/thickness parameters 3.5 microns and below have Péclet numbers $\lesssim 1$, which indicated the fluid flow for IgG particles were governed mainly by diffusion.

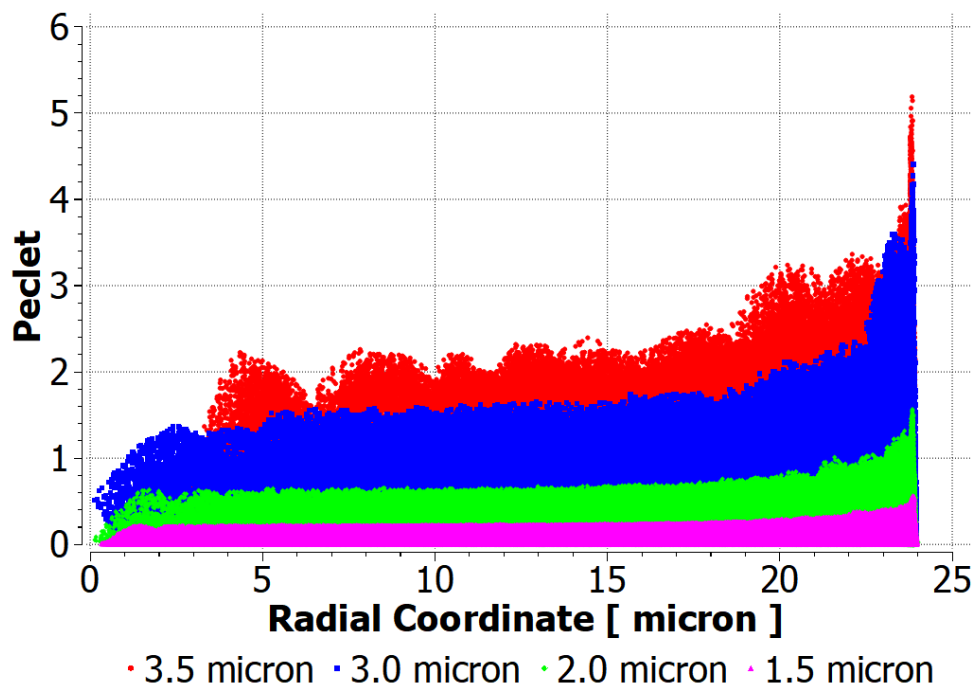


FIGURE 4.12: The relationship of the cell Péclet number distribution for IgG vs the radial coordinate of the BOI resin for pore lattice structures of 3.5 μm , 3.0 μm , 2.0 μm and 1.5 μm . The diffusion coefficient used was for IgG protein in dilute solution at 25 °C where $D = 40 \mu\text{m}^2 \text{s}^{-1}$.

This data opposes the manufacturer's claims that the through-pores' presence allowed the convective fluid flow to enhance protein penetration into the bead. The more likely reason for improved performance was that these through pores provided a larger pore for the protein to diffuse into the bead. While this type of fluid flow works for proteins that can diffuse quickly into the center of the porous bead, the same does not apply to larger, slower bioparticles such as AAV. In Figure 4.13, relationship between the radial coordinate of the resin and the distribution cell Péclet number's distribution for AAV can be seen. The Péclet number's were calculated using both an observed diffusion coefficient ($D = 7.5 \mu\text{m}^2 \text{s}^{-1}$ (Seisenberger et al., 2001))) and the diffusion coefficient calculated from the Stokes Einstein equation $D = 11 \mu\text{m}^2 \text{s}^{-1}$ which assumed AAV was a spherical particle

with radius of 13 nm. Clearly, the transport of AAV particles within the resin through-pores would mainly be governed by convection.

Additionally, these results confirm that CFD models can successfully capture the essential characteristic of flow-through macroporous resin effectively using reported literature values. Moreover, these models provide insight into crucial bioprocess unit operations that could not be characterized using traditional empirical experiments. The low fluid shear within the packed bed, in addition to the relatively uniform and lack of eddy formation fluid flow at typical operating conditions, suggest that mechanical stresses are not the cause of low yields during chromatography unit operations. Instead, high salt conditions and detergents are the more likely the cause of large bioparticle degradation.

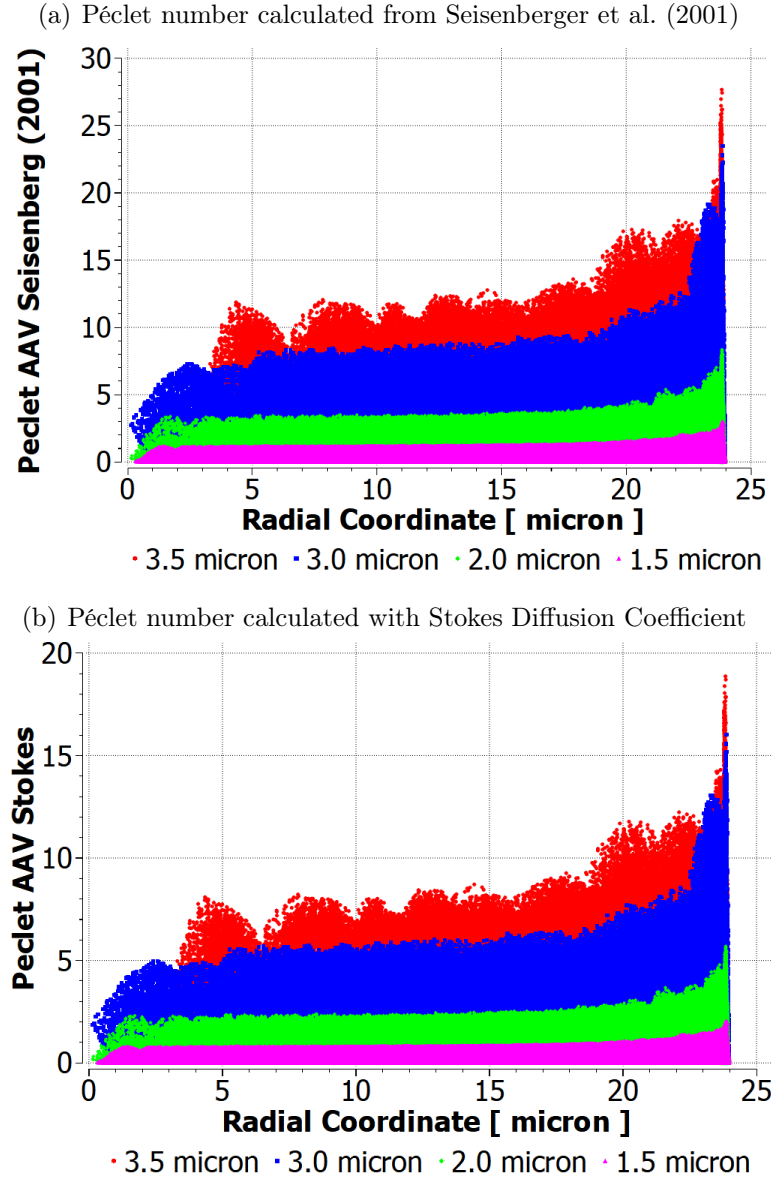


FIGURE 4.13: The relationship of the AAV cell Péclet number distribution vs the radial coordinate of the BOI resin for pore lattice structures of 3.5 μm , 3.0 μm , 2.0 μm and 1.5 μm with an inlet velocity of 1000 cm h^{-1} . (a) The diffusion coefficient used was based on the Stokes Einstein equation $D = 11 \mu\text{m}^2 \text{s}^{-1}$ for AAV assuming a spherical particle with radius of 13 nm. (b) The diffusion coefficient used was based on the Seisenberger et al. (2001) observed diffusion coefficient $D = 7.5 \mu\text{m}^2 \text{s}^{-1}$.

4.2 DPM Models

The CFD-DPM simulations were run using Ansys Fluent and with the previously described geometry and mesh (see Figure 3.5 and Section 3.2.1). The mesh was scaled uniformly by 10^{-4} , so that bead radii are 25 μm . The flow was assumed to be isothermal and incompressible with the physical properties of water. The interface between the solid and fluid (i.e., walls of the flow cell, surrounding beads and BOI throughpore geometry) were defined as ‘no-slip’ wall conditions, and the outlet was defined as a ‘pressure-outlet’. At the velocity-inlet a constant velocity profile was set to 50 cm h^{-1} , 100 cm h^{-1} , 150 cm h^{-1} , 200 cm h^{-1} and 250 cm h^{-1} .

To simulate the capture and release of a tracer from the BOI, the model was broken up into three parts and each part was performed for each of the inlet velocities:

- (i) Use the converged steady-state velocity profile to initialize the transient model.
- (ii) Capture Simulation: transient simulation where DPM tracer particles were injected over 33 timesteps from the inlet using the DPM model. Particles had IgG properties. Particles that came into contact with the BOI surfaces and with the outlet were considered either be trapped or escaped (respectively), otherwise all walls reflected particles and internal boundaries allowed particles to pass through.
- (iii) Release Simulation: a transient simulation was initialized with the converged steady state velocity profile, and DPM tracer particles were injected from the capture locations (on BOI) derived from the capture simulation. Particle fates were recorded for particles that escaped through the outlet.

4.2.1 Capture Simulation

For each of the inlet velocities (50 cm h^{-1} , 100 cm h^{-1} , 150 cm h^{-1} , 200 cm h^{-1} and 250 cm h^{-1}), the simulation was initialized using the converged steady state velocity profile. Particles were injected into the transient simulation from the inlet over 33 time steps, where 300 particles were injected each time step. These particles treated every surface in the model as walls (where they were reflected) except for the BOI resin surfaces and the outlet which were designated as trap surfaces. A custom DPM report for monitoring surfaces was generated for tracking particles trapped on the BOI resin surfaces and the outlet (see Appendix C for custom report code). Each surface report recorded information about each particle as it hit the surface, which includes: time of injection, xyz coordinates of location particle was trapped, uvw velocity coordinates of the particle, the diameter of the particle, the mass of the particle, the particle ID, the flow time, the residence time, shear stress integral, and the strain rate integral.

Figure 4.24 shows the radial location of particles captured on the resin BOI normalized by the resin's radius versus the particle's residence time. Particles do not make it to the center of the bead and a majority of them are captured on the outer 5% of the resin BOI. This is clearly shown in Figure 4.22, where the Outer Region makes up the outer 5% BOI's radius and where 90% of particles are trapped on the resin are located.

To better understand and characterize the particle macromixing amount in the model, the age distribution functions were calculated for both particles captured on the resin and those exiting through the outlet. These functions include exit age distribution function (E-curve, $E(t)$), cumulative distribution function (F-curve, $F(t)$), internal age distribution function ($I(t)$), and intensity function ($\Lambda(t)$) and are related as defined by the following equations:

The E-curves were calculated using the python packages seaborn and SciPy to estimate the Kernel Density Estimation (KDE) (Waskom and team, 2020; Virtanen et al., 2020).

$$E(t), s^{-1} = \frac{dF(t)}{dt} \quad (4.3)$$

$$F(t) = \int_0^t E(t) dt = 1 - E(t) \quad (4.4)$$

$$\Lambda(t) = \frac{E(t)}{\bar{t}_m I(t)} = \frac{E(t)}{1 - F(t)} = \frac{E(t)}{W(t)} \quad (4.5)$$

where t is the residence time and \bar{t}_m is the mean residence time. These functions can then be converted to their dimensionless forms by substituting $t = \bar{t}_m \theta$ and multiplying the whole equation by the mean residence time, \bar{t}_m , using the following equations:

$$E(t) dt = E_\theta(\theta) d\theta \quad \therefore \quad (4.6)$$

$$E_\theta(\theta) = \frac{E(t)}{d\theta} = \bar{t}_m E(t \theta) \quad (4.7)$$

$$F_\theta(\theta) = F(\bar{t}_m \theta) \quad (4.8)$$

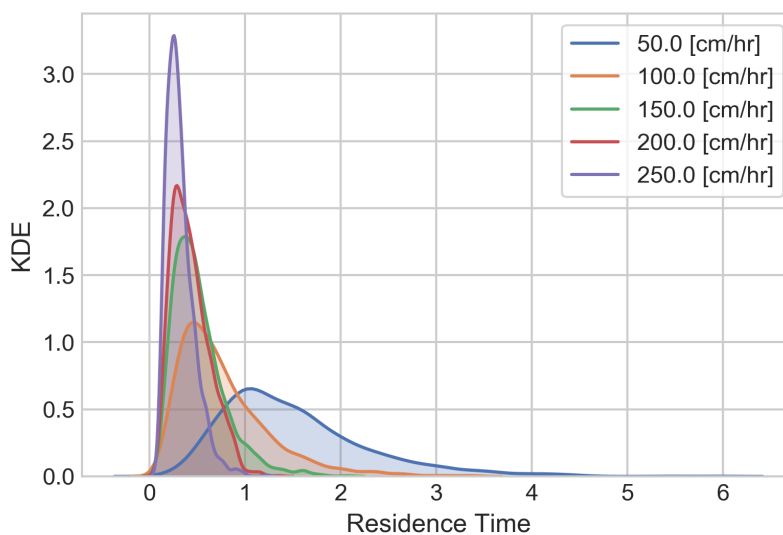
$$W_\theta(\theta) = W(\bar{t}_m \theta) \quad (4.9)$$

$$I_\theta(\theta) = \bar{t}_m I(\bar{t}_m \theta) \quad (4.10)$$

$$\Lambda_\theta(\theta) = \bar{t}_m \Lambda(\bar{t}_m \theta) \quad (4.11)$$

In Figure 4.15, particle RTDs for each of the inlet velocities were represented as histograms separated by particle fate (i.e., trapped on BOI resin or flow through to outlet). The corresponding KDE (aka the E-curve) of these histograms can be seen in

(a) Chromatogram of particles captured on resin



(b) Boxplot of captured particle residence times

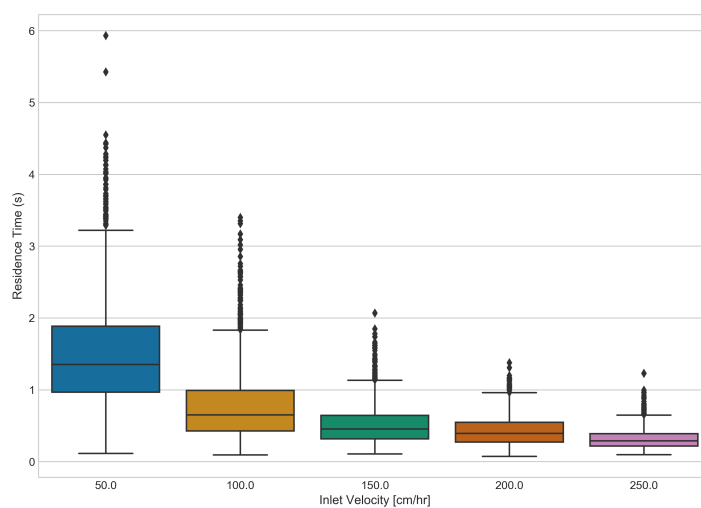


FIGURE 4.14: Distribution of particle residence time for the capture simulation trapped on BOI rendered as (a) a histogram and (b) a boxplot.

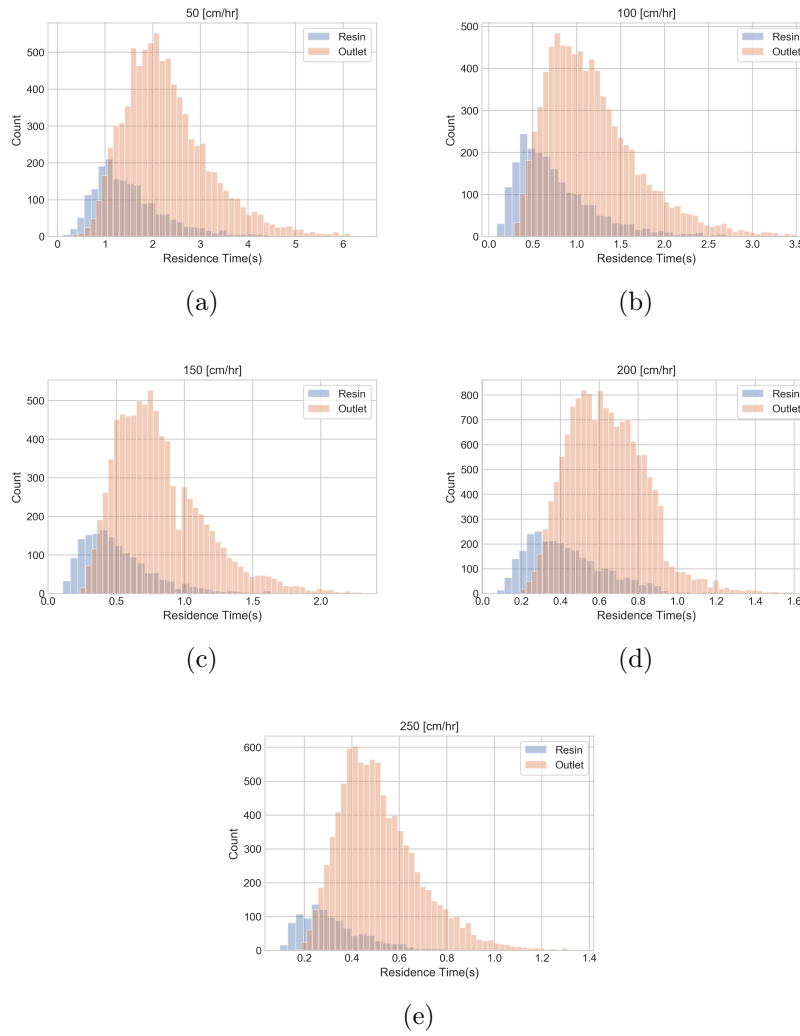


FIGURE 4.15: Histograms of Residence Time Distribution of DPM simulations separated by particle fate location (i.e., particles captured on resin or particles that flowed through to outlet).

Figure 4.21. Figure 4.16 and Figure 4.17 also show the E-curves at the different velocities, in addition to, the other age distribution functions ($E(t)$, $F(t)$, $W(t)$, $I(t)$, and $\Lambda(t)$) and the corresponding dimensionless functions (E_θ , F_θ , W_θ , I_θ , and Λ_θ) for particles captured on the resin and outlet, respectively.

The first and second moments of the E-curve are often used to compare different RTDs. The first moments of the E-curve calculates the mean residence time (\bar{t}_m) which is equal to space time (τ) for constant volumetric flow. The mean residence time is defined as:

$$\tau = \bar{t}_m = \int_0^\infty tE(t)dt \quad (4.12)$$

The second moment of the E-curve is called the variance, and is the square of the standard deviation. The variance is defined as:

$$\sigma^2 = \int_0^\infty (t - \bar{t}_m)^2 E(t)dt \quad (4.13)$$

and describes the spread of the distribution. Using the E-curves and Equation 4.12, the mean residence time was calculated for particles escaping through the outlet during the capture simulation. Figure 4.18 shows the dependence of mean residence time on the volumetric flow rate (Q) using the following equation:

$$t_m = a Q^{-1} \quad (4.14)$$

where a is the adjusted coefficient that represents the active volume of the system. This adjusted coefficient was estimated by fitting Equation 4.14 to the outlet t_m data using

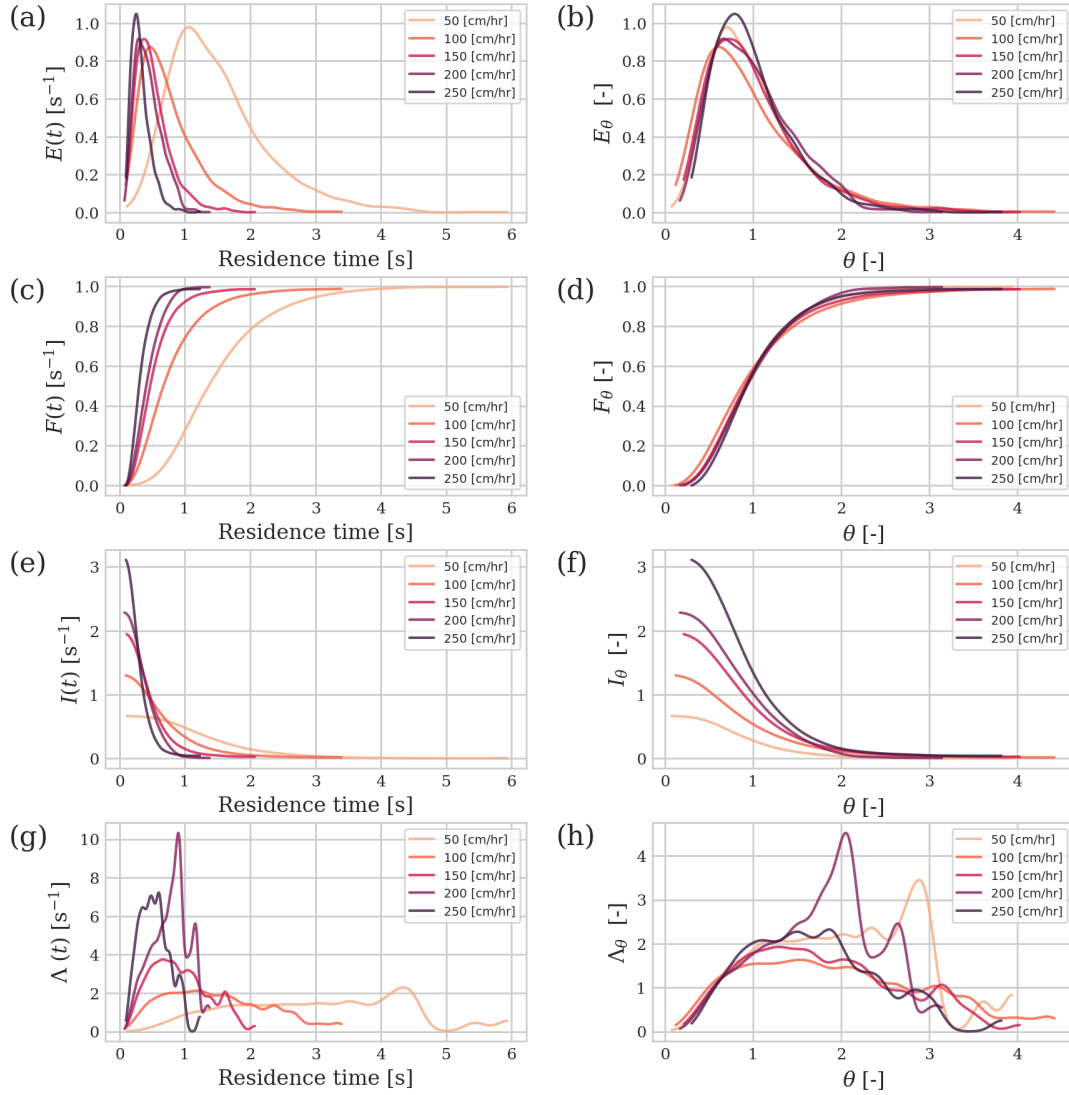


FIGURE 4.16: Age distribution functions of DPM particles captured on the BOI resin during the the capture simulation at 5 inlet velocities. (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively. (b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.

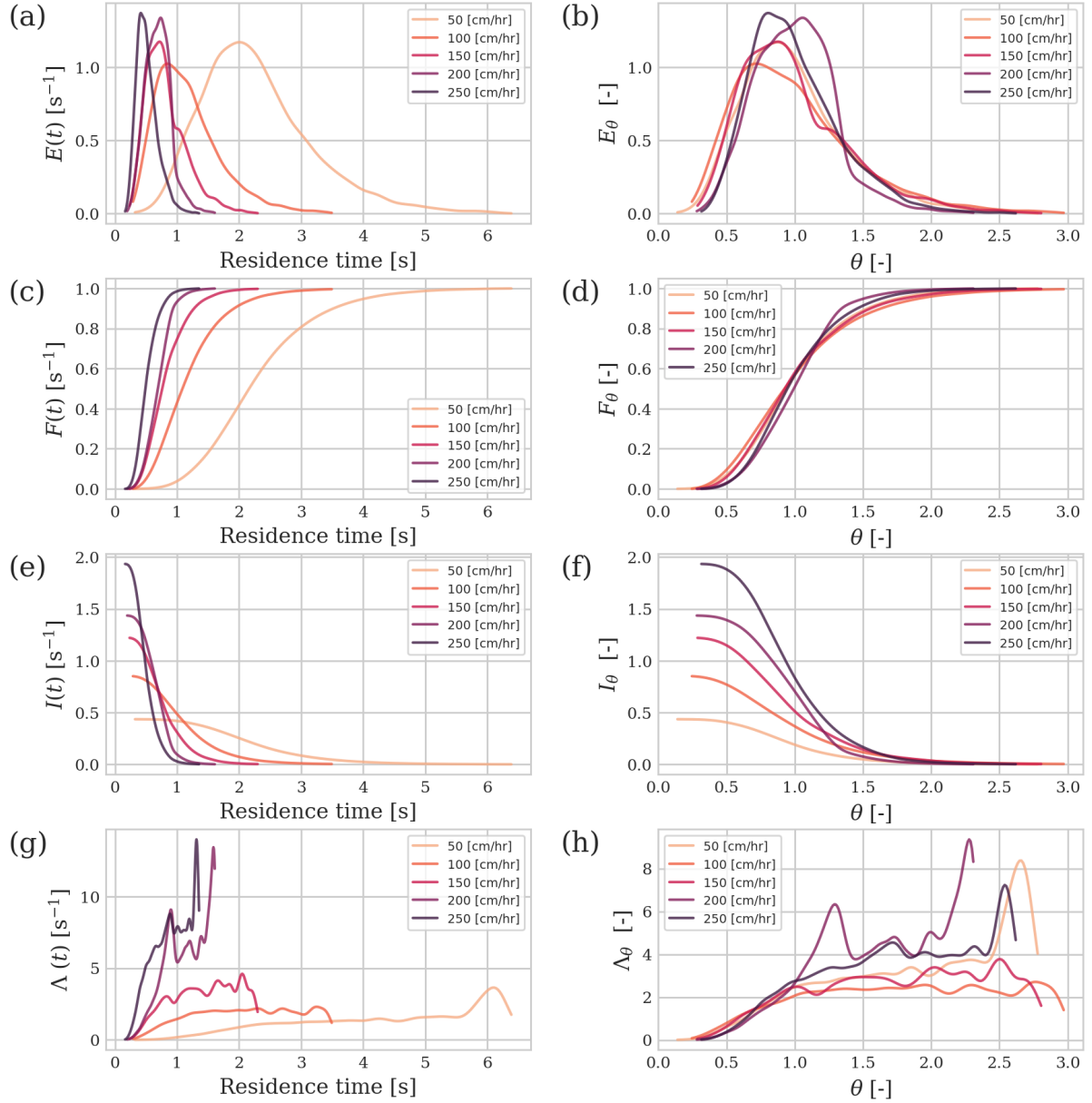


FIGURE 4.17: Age distribution functions of DPM particles that escaped through the outlet during the capture simulation at 5 inlet velocities. (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively. (b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.

the non-linear least squares `scipy.optimize.curve_fit` function (Virtanen et al., 2020). The resulting active volume of the system was estimated to be $8.353 \times 10^{-3} \mu\text{L}$ with an $R^2 = 0.991$. While this estimate deviates from the nominal value of the model's fluid region volume, which is $1.0244 \times 10^{-2} \mu\text{L}$, this estimate is still within 18.5% of the modeled fluid volume.

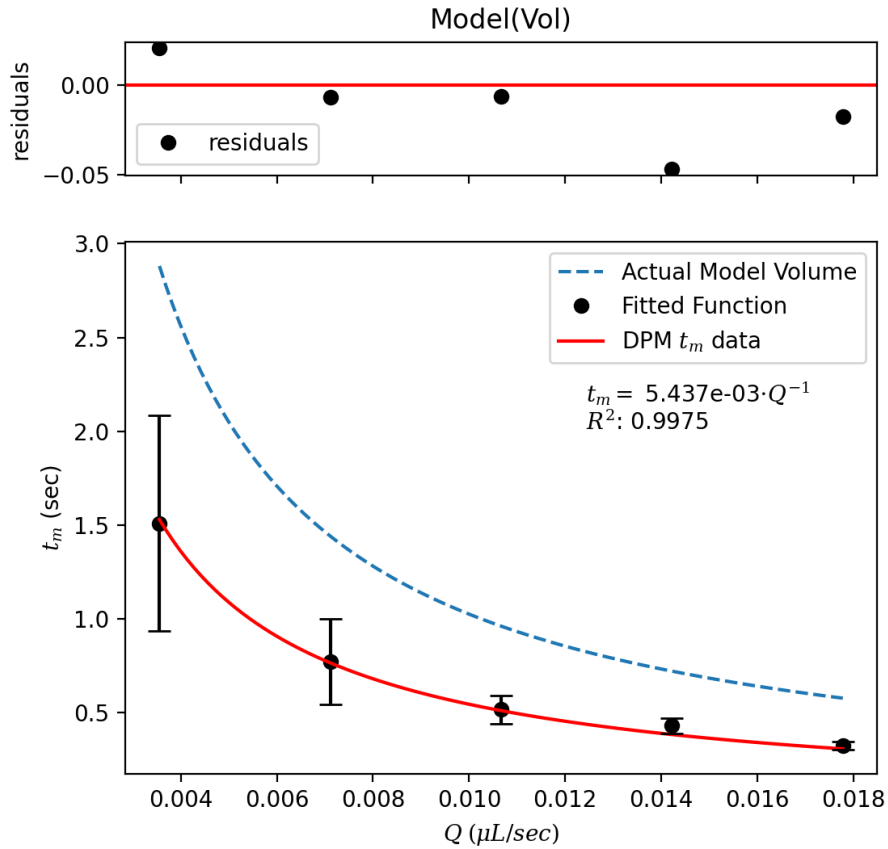


FIGURE 4.18: Relationship of the mean residence time (t_m) of outlet particles with volumetric flow rate (Q). The black error bars represent the variance of the residence time distribution.

After calculating the σ^2 and the \bar{t}_m from the respective exit age distribution curves, the dimensionless variance of the distribution is equal to:

$$\sigma_\theta^2 = \frac{\sigma^2}{\bar{t}_m} \quad (4.15)$$

The Péclet number can be estimated and expressed according to van Gelder and Westerterp (1990):

$$\sigma_\theta = \frac{2}{Pe} - \frac{2}{Pe^2} (1 - e^{-Pe}) \quad (4.16)$$

The DPM simulation E-curves were fit to two single-parameter RTD models to characterize the fluid flow pattern within this packed bed flow-cell. The single-parameter RTD models used are the axial dispersion model with open-open boundary conditions described by Levenspiel and Smith (1957) and the N-CSTR (Continuously Stirred Tank Reactor) or Tanks in Series model described by Levenspiel (1999). These models were optimized using the `scipy.optimize.curve_fit` function to minimize the sum of squared errors (SSE) on E using nonlinear regression to find optimal values of Pe and D_{ax} . The formula used to calculate the SSE is defined as:

$$SSE = \sum_{i=1}^n (E_{\theta_{exp.i}}(\theta) - E_{\theta_{pred.i}}(\theta))^2 \quad (4.17)$$

For the Tanks in Series model, the Péclet number (Pe) was calculated using the following equations:

$$Pe = 2(n_T - 1), \quad (4.18)$$

$$E_\theta = \frac{n_T^{n_T}}{(n_T - 1)! \cdot \theta_t^{n-1} \cdot e^{-n_T \cdot \theta}} \quad (4.19)$$

$$\theta = \frac{t}{\bar{t}_m} \quad (4.20)$$

where n_T represents the number of tanks in series, θ is dimensionless time, \bar{t}_m is the mean residence time and t is time. Figure 4.19 shows the results of the tanks in series RTD model fit to the particles trapped on the resin during the capture DPM simulations at under various inlet velocities. Figure 4.20 shows the corresponding model fits for particles captured at the outlet in the capture DPM simulations with the same inlet velocities.

For the axial dispersion RTD model with open-open boundary conditions, Levenspiel and Smith (1957) proposed the following analytical solution:

$$E(t) = \frac{\sqrt{Pe}}{\tau \sqrt{4\pi\theta}} \exp \left[\frac{-Pe(1 - \theta)^2}{4\theta} \right], \quad (4.21)$$

$$\theta = \frac{t}{\bar{t}_m} \quad (4.22)$$

where Pe is the dimensionless model parameter ($Pe = L \cdot U / D_{ax}$, (-)), L is the length of the tube, U is the average interstitial velocity (L/\bar{t}_m), D_{ax} is the axial flow dispersion coefficient ($\text{m}^2 \text{s}^{-1}$), and θ is dimensionless time (-).

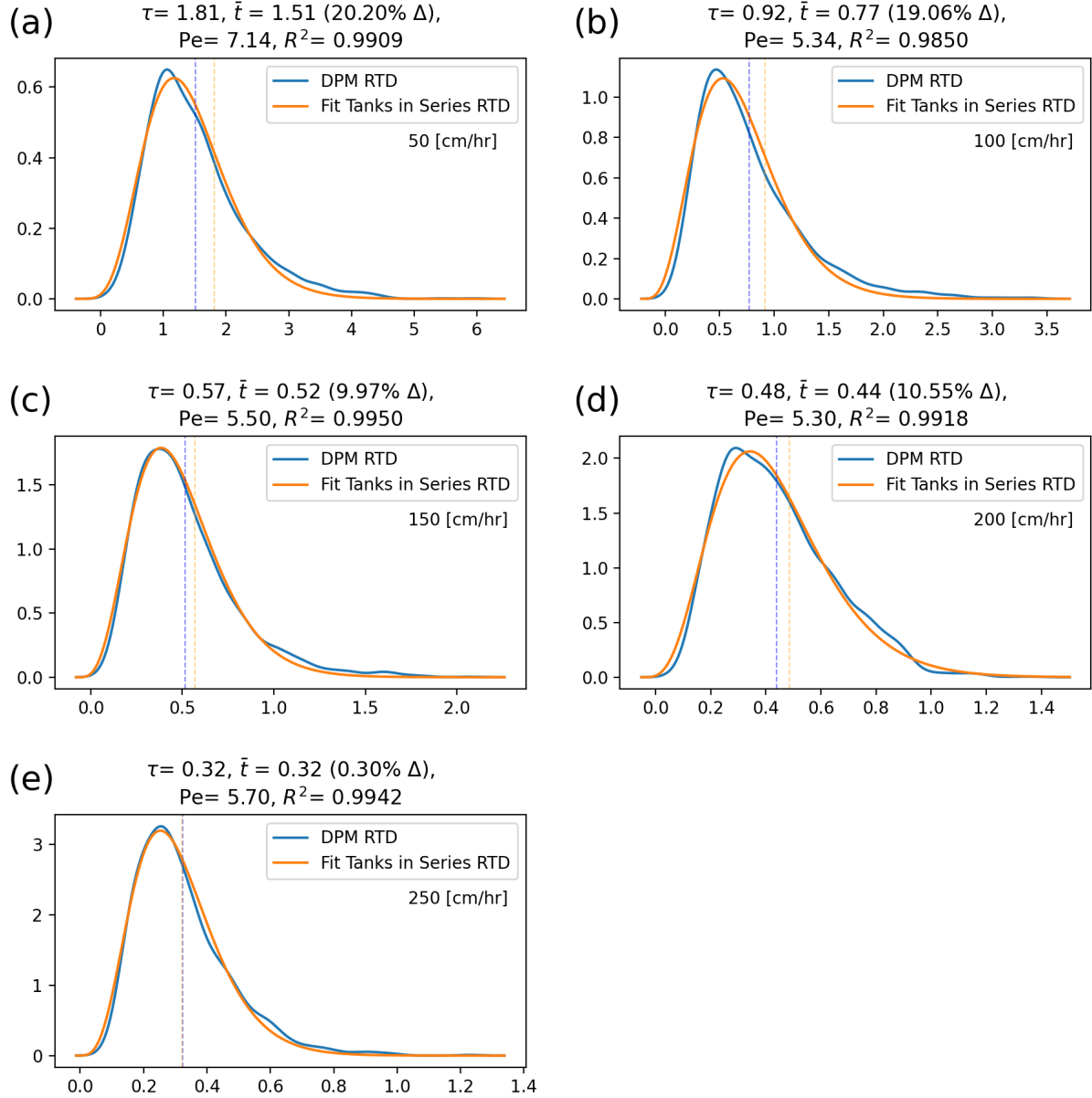


FIGURE 4.19: The results of the tanks in series RTD model (solid orange curve) fit to the particles trapped at the resin during the capture DPM simulations (solid blue curve) at various inlet velocities. The blue vertical dash line indicates the mean residence time of the DPM simulation; the orange vertical dash line indicates mean residence time of the tanks in series RTD model.

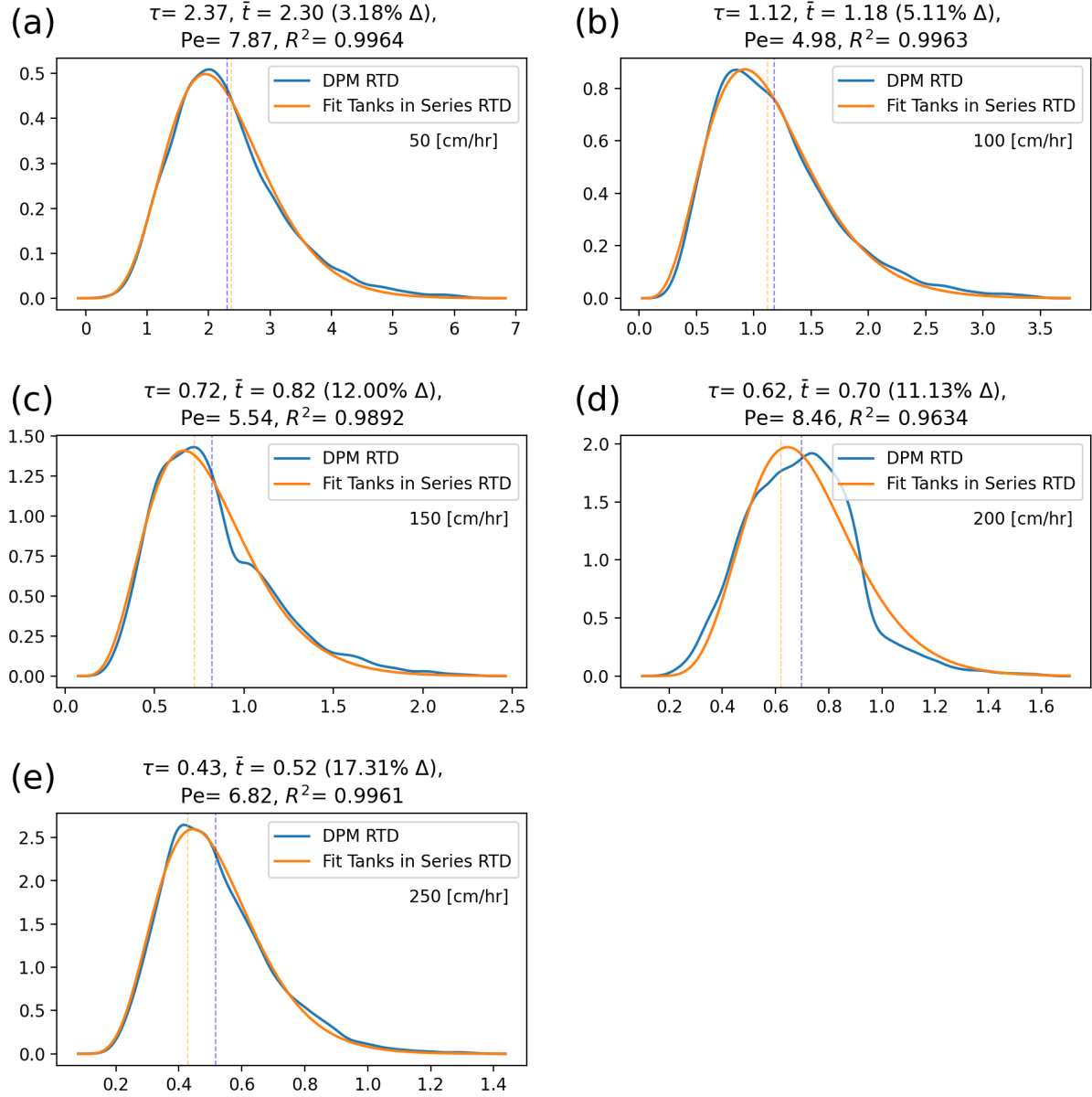


FIGURE 4.20: The results of the tanks in series RTD model (solid orange curve) fit to the particles that escape through the outlet during the capture DPM simulations (solid blue curve) at various inlet velocities. The blue vertical dash line indicates the mean residence time of the DPM simulation; the orange vertical dash line indicates mean residence time of the tanks in series RTD model.

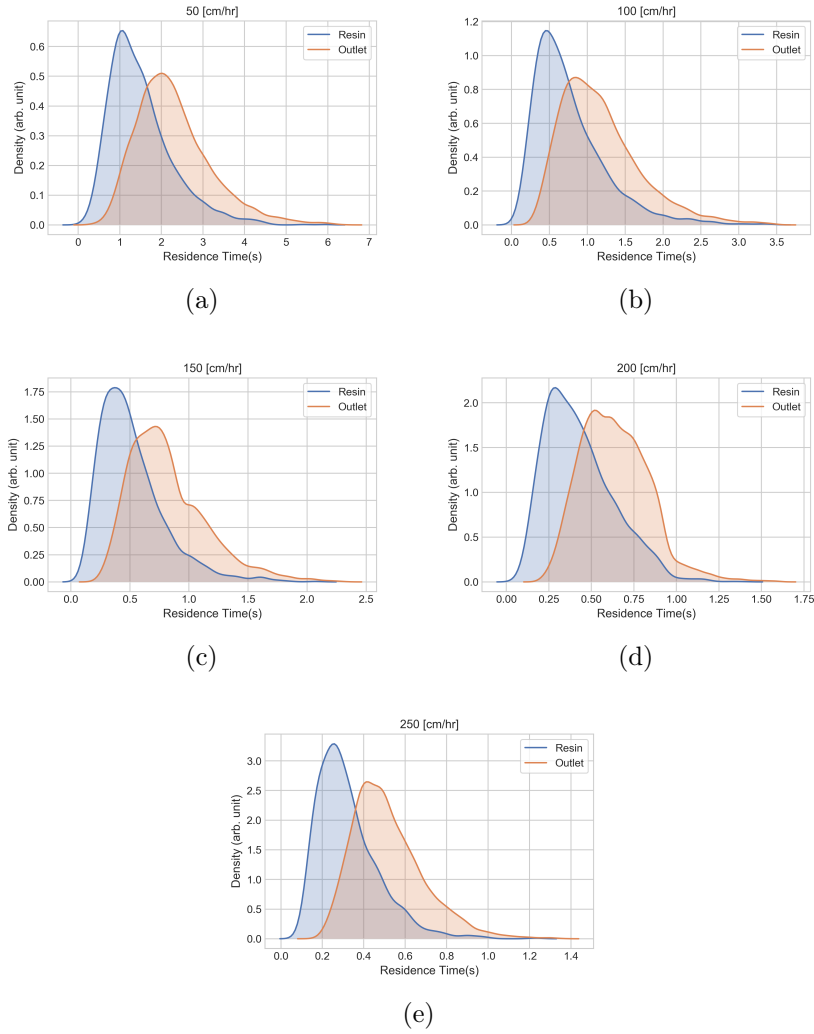


FIGURE 4.21: Kernel Density Estimate of Residence Time Distribution of DPM simulations separated by particle fate location. (i.e., particles captured on resin or particles that flowed through to outlet).

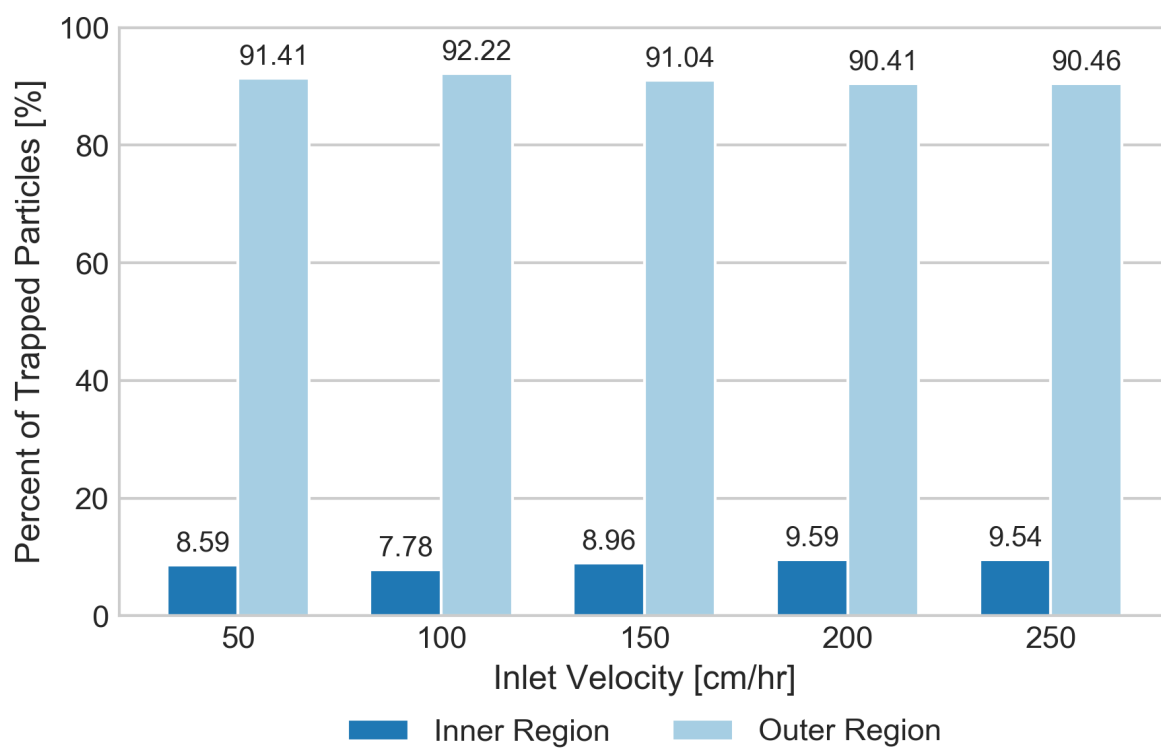


FIGURE 4.22: Percent of trapped particles captured on BOI categorized by inner and outer regions (dark blue and light blue respectively).

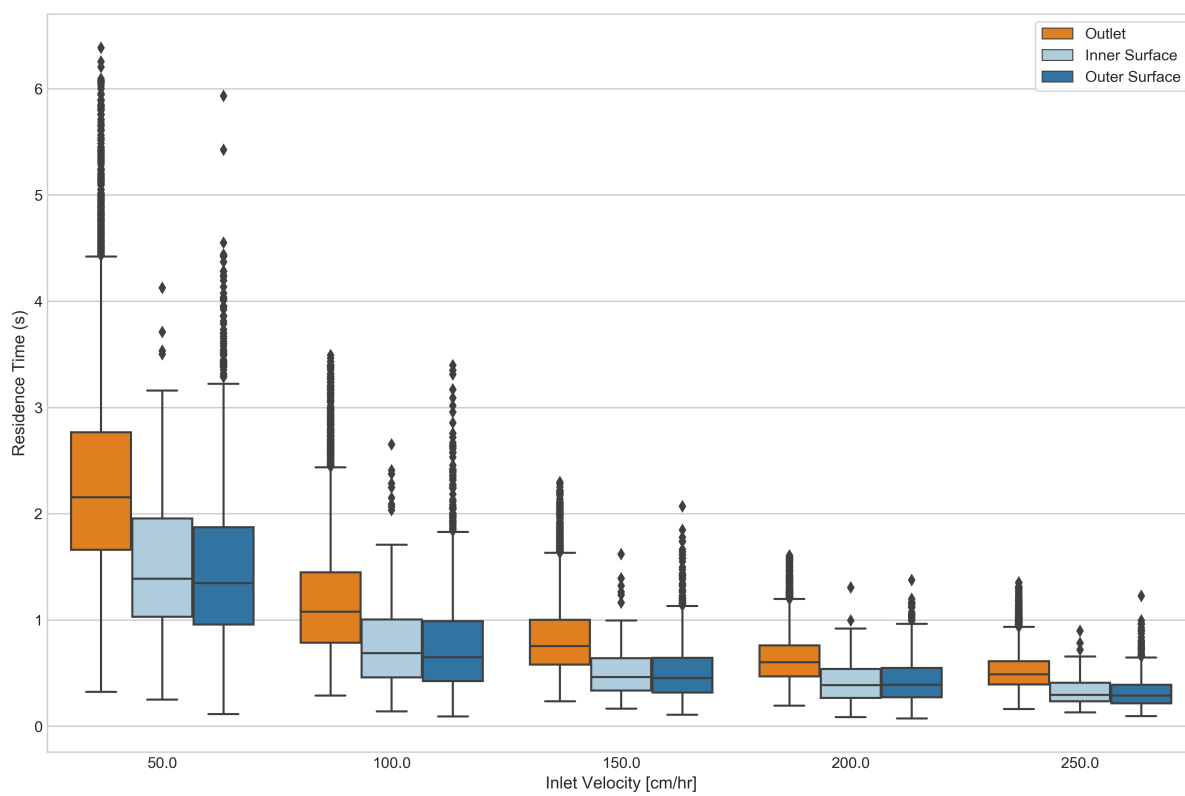


FIGURE 4.23: Boxplot showing the Residence Time Distribution of particles separated out by surface.

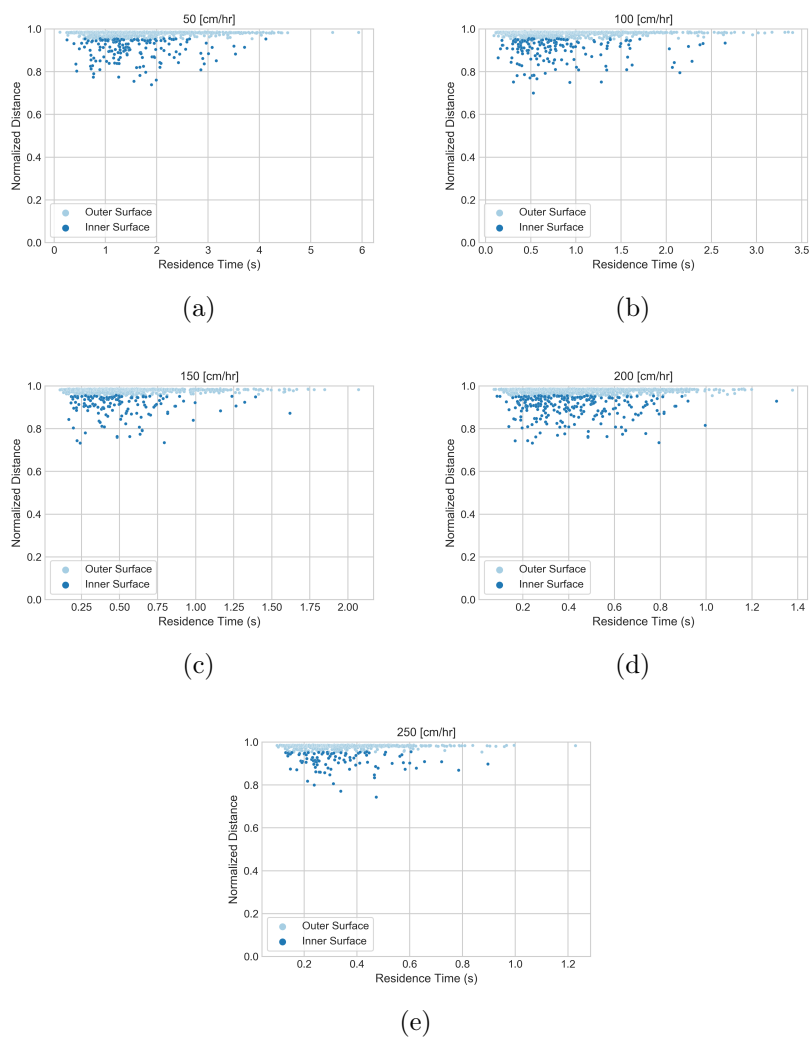
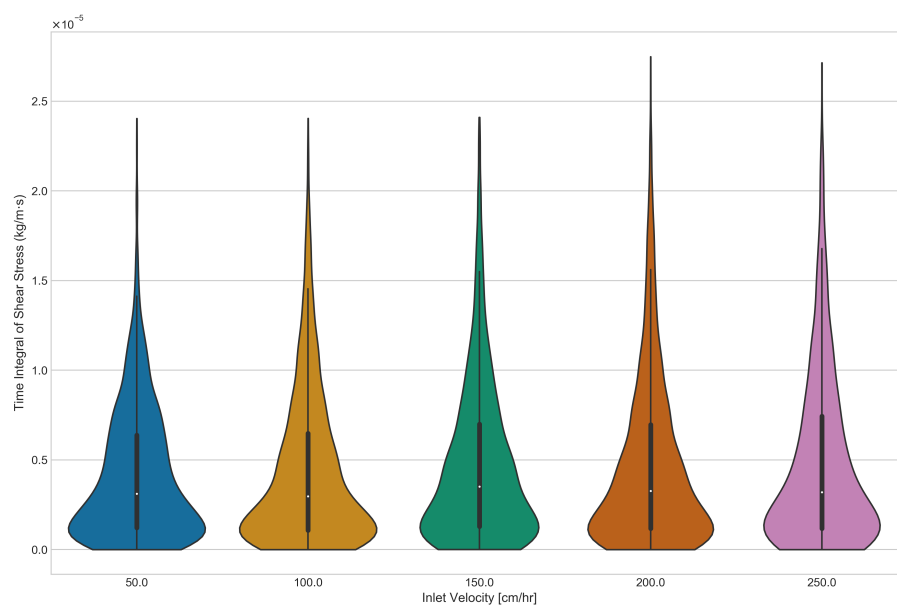
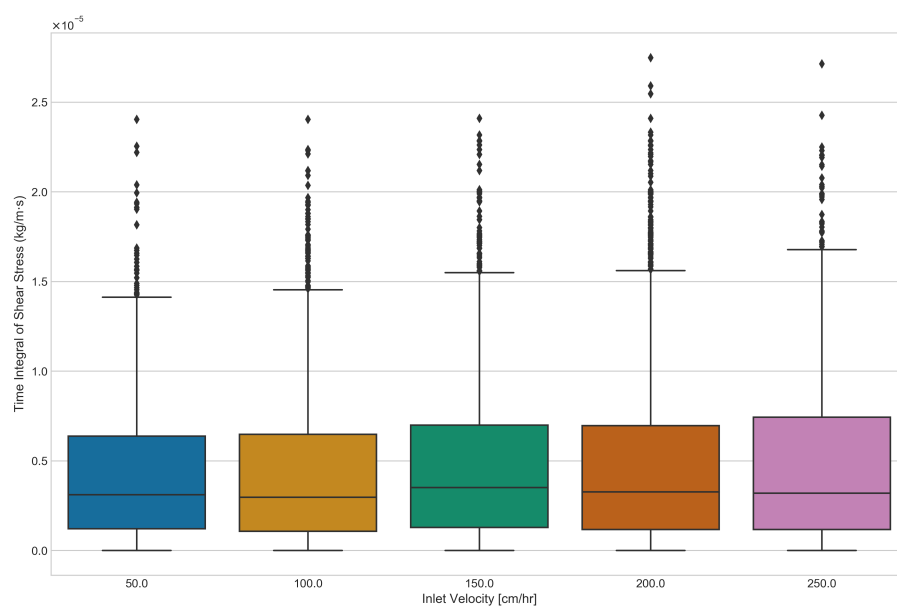


FIGURE 4.24: Residence Time of particles vs the particle capture location normalized to the particle's radius. Points are colored by the capture surface of the BOI.



(a)



(b)

FIGURE 4.25: (a) Particle Shear Stress Distribution in Terms of Percent Bound to Resin. (b) Particle Shear Stress integral Distribution

4.2.2 Release Simulation

The second part of the model is the the transient “release” simulation that mimics a tracer molecule eluting from a chromatography bead. The model was initialized with the converged steady state velocity profiles for each of the five inlet velocities, and particles were released into the transient simulation in a single time step from the BOI trap locations recorded during the capture simulations. All surfaces were treated as walls except the outlet which was designated as trap surface. A custom DPM report for monitoring surfaces was generated for tracking particle fates (see Appendix C for custom report code). The surface report recorded information about each particle as the particle came into contact with the the outlet surface. The values of the report consist of: time of injection, xyz coordinates of location particle was trapped, uvw velocity coordinates of the particle, the diameter of the particle, the mass of the particle, the particle ID, the flow time, the residence time, shear stress integral, and the strain rate integral. For general statistics of the particles, refer to Figures 4.27 and Table 4.3. Figure 4.26 shows the plots of particle distributions.

To better understand and characterize the particle macromixing in the model,

TABLE 4.3: DPM Particle exit statistics for release simulations

Inlet Velocity	50 cm h ⁻¹	100 cm h ⁻¹	150 cm h ⁻¹	200 cm h ⁻¹	250 cm h ⁻¹
Average	3.672	0.776	0.487	0.433	0.331
Min	0.078	0.093	0.089	0.069	0.095
Max	7.699	3.392	1.718	1.209	1.08
StdDev	2.032	0.401	0.254	0.183	0.144
Count	1805	2119	1566	2501	1069

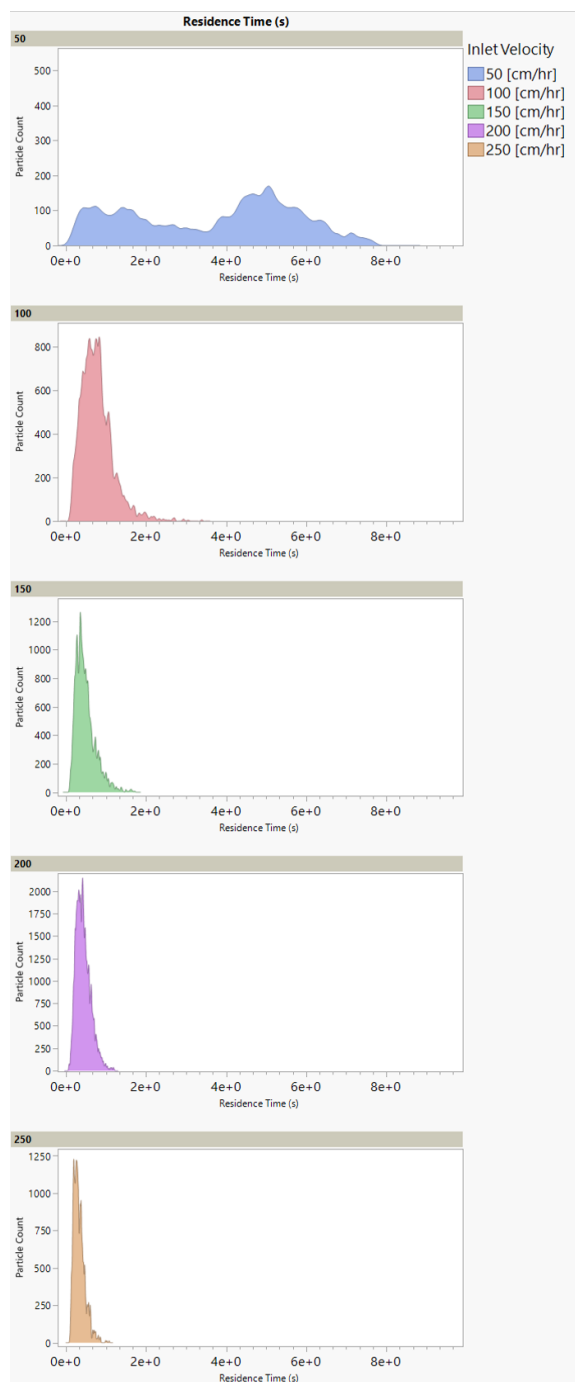


FIGURE 4.26: Individual chromatograms from Figure 4.27

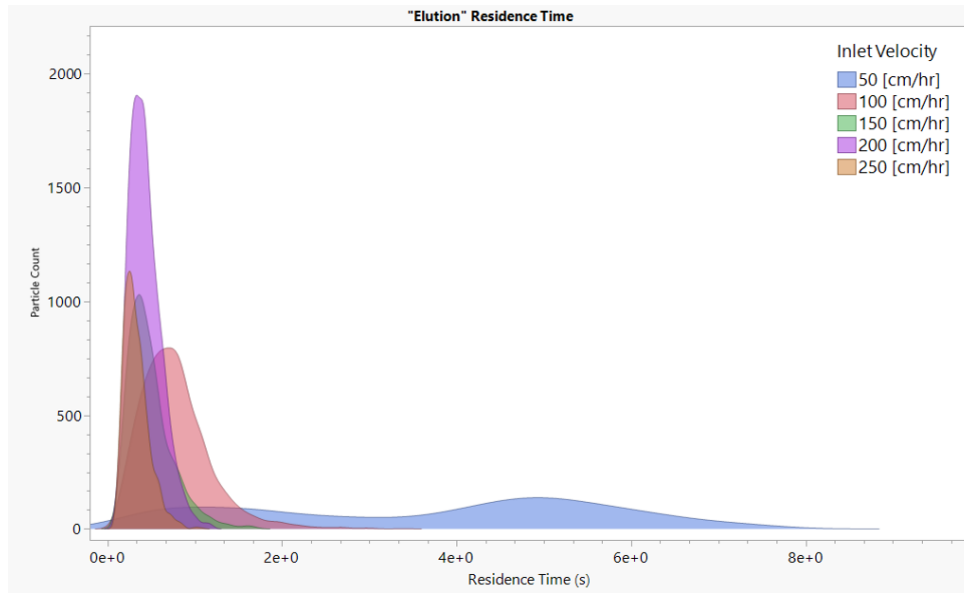


FIGURE 4.27: Stacked release chromatograms, see Table 4.3 for particle statistics

the age distribution functions ($E(t)$, $F(t)$, $I(t)$, and $\Lambda(t)$) and their corresponding dimensionless function were calculated for the released particles exiting through the outlet using Equations (4.3) and their plots can be seen in Figure 4.28.

The release DPM simulation E-curves were fit to the same two single-parameter RTD models and methods described in Section 4.2.1 to characterize the fluid flow pattern within this packed bed flow-cell. A summary of the fit parameters and the the resulting τ and Pe numbers can be found in Table 4.4.

The E-curves of the DPM particles display typical characteristics of a packed-bed reactor, such as increasing peak amplitudes and decreasing variance with increasing velocities. The E-curve variance can be used as an evaluation factor to estimate the flow inhomogeneity since smaller variance results in narrower E-curves and more uniform flow. At 50 cm h^{-1} the E-curve has two peaks where the principal peak occurred before

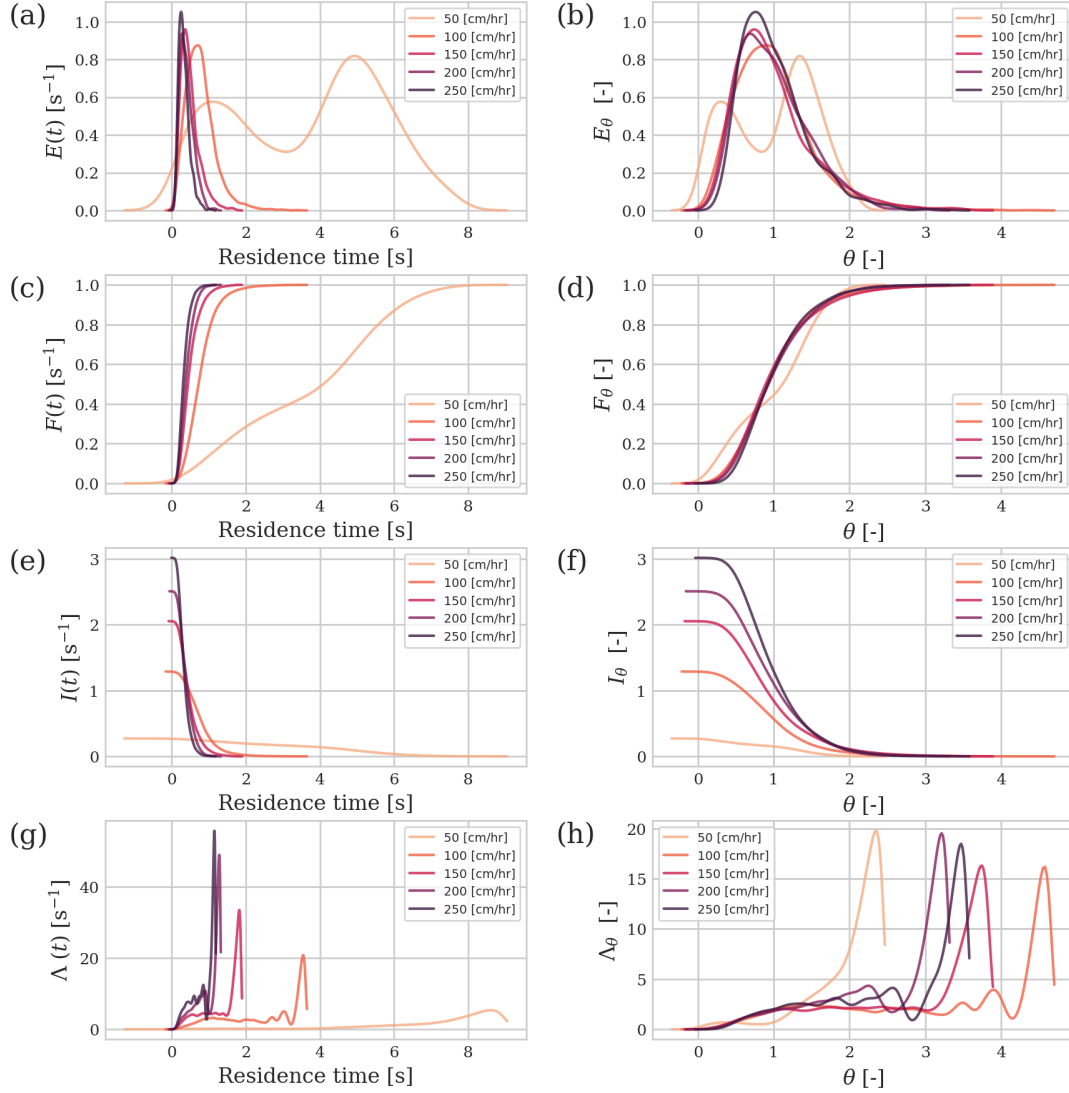


FIGURE 4.28: Age distribution functions of DPM particles that escaped through the outlet during the release simulation for 5 inlet velocities; (a,c,e,g) are the plots for the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.(b,d,f,h) are the dimensionless plots of the exit age, cumulative distribution, internal age distribution, and intensity functions, respectively.

the mean residence time (\bar{t}_m) while a larger second peak occurred after the mean residence time. This double peak was indicative of a packed bed reactor with channeling and dead zones. The dead zone was most likely due to flow resistance brought on by the bead's through-pore structure, and can be seen in the bead cross section contours of Z-velocity profile of the steady-state solution (Figure 4.29(a)). Particles that were captured on resin's outer surface reached the outlet before those released in the throughpore channels because of their proximity to regions with higher Péclet numbers and thus higher convective flow. The Péclet cell number was calculated using the following equations:

$$Pe = Re Sc \quad (4.23)$$

$$Sc = \frac{\nu}{D} \quad (4.24)$$

where Re is the cell Reynolds number, Sc is the Schmidt number, ν is the kinematic viscosity with the value of $1 \times 10^{-6} \text{ m}^2 \text{ s}^{-1}$ and D is the diffusion coefficient of IgG in dilute aqueous solution at 25°C is $4 \times 10^{-11} \text{ m}^2 \text{ s}^{-1}$ (Wrzosek et al., 2013). Cell Reynolds number was defined as:

$$Re \equiv \frac{\rho u d}{\mu} \quad (4.25)$$

where ρ was the density, u was the velocity magnitude, μ was the effective viscosity (laminar plus turbulent) and d was the **Cell Volume**^{1/3} for 3D simulations (Inc., 2019b). In Figure 4.30(a), the contours of the cell Péclet number showed the spacial distribution of the cell Péclet number for the BOI's cross section. In Figure 4.31, the volume sampled distribution of the cell Péclet number is plotted by the radial coordinate with respect to the center of the BOI. The values of the cell Péclet numbers within the bead indicate that

the flow is governed mainly by diffusion while in the bed region the flow is dominated by convection.

TABLE 4.4: Curve fitting results for the release simulation

Inlet Velocity	CFD DPM				N-Tanks			Axial Dispersion open-open		
	\bar{t}_m	σ^2	σ_θ^2	Pe_{σ_θ}	$\tau_{N-tanks}$	$Pe_{N-tanks}$	$R_{N-tanks}^2$	$\bar{\tau}_{ooD_{ax}}$	$Pe_{ooD_{ax}}$	$R_{ooD_{ax}}^2$
50 cm h ⁻¹	3.672	4.33	0.321	5.218	5.723	4.962	0.491	4.29	4.602	0.419
100 cm h ⁻¹	0.776	0.168	0.28	6.274	0.933	10.597	0.996	0.822	10.913	0.986
150 cm h ⁻¹	0.487	0.068	0.286	6.129	0.544	9.991	0.991	0.476	10.356	0.997
200 cm h ⁻¹	0.399	0.035	0.221	8.386	0.456	9.948	0.993	0.398	10.375	0.997
250 cm h ⁻¹	0.331	0.022	0.201	9.397	0.333	9.853	0.994	0.291	10.232	0.999

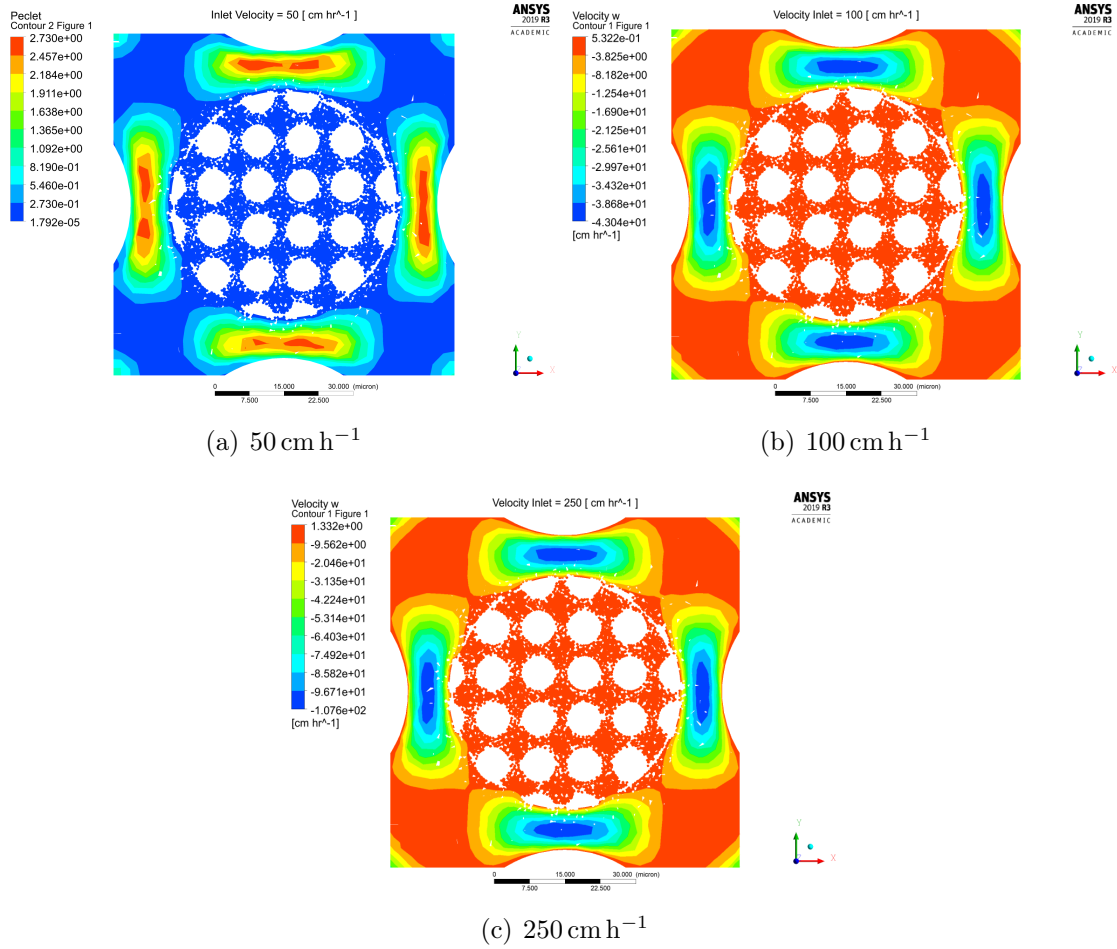


FIGURE 4.29: Contours of the Z-velocity of in an XY-plane cross section through the center of the BOI. Negative velocity values point towards the outlet.

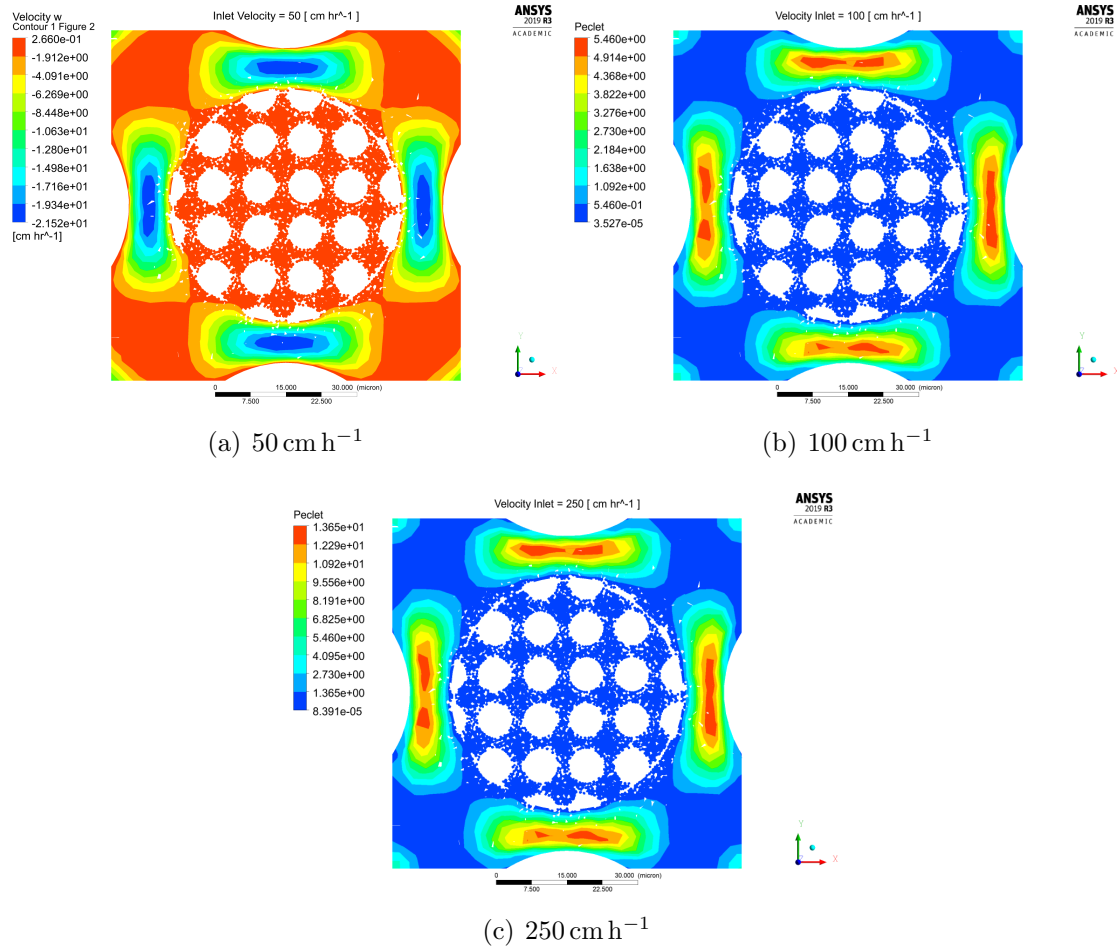


FIGURE 4.30: Distribution of the cell Péclet number within a 50 micron radius of the center of the BOI.

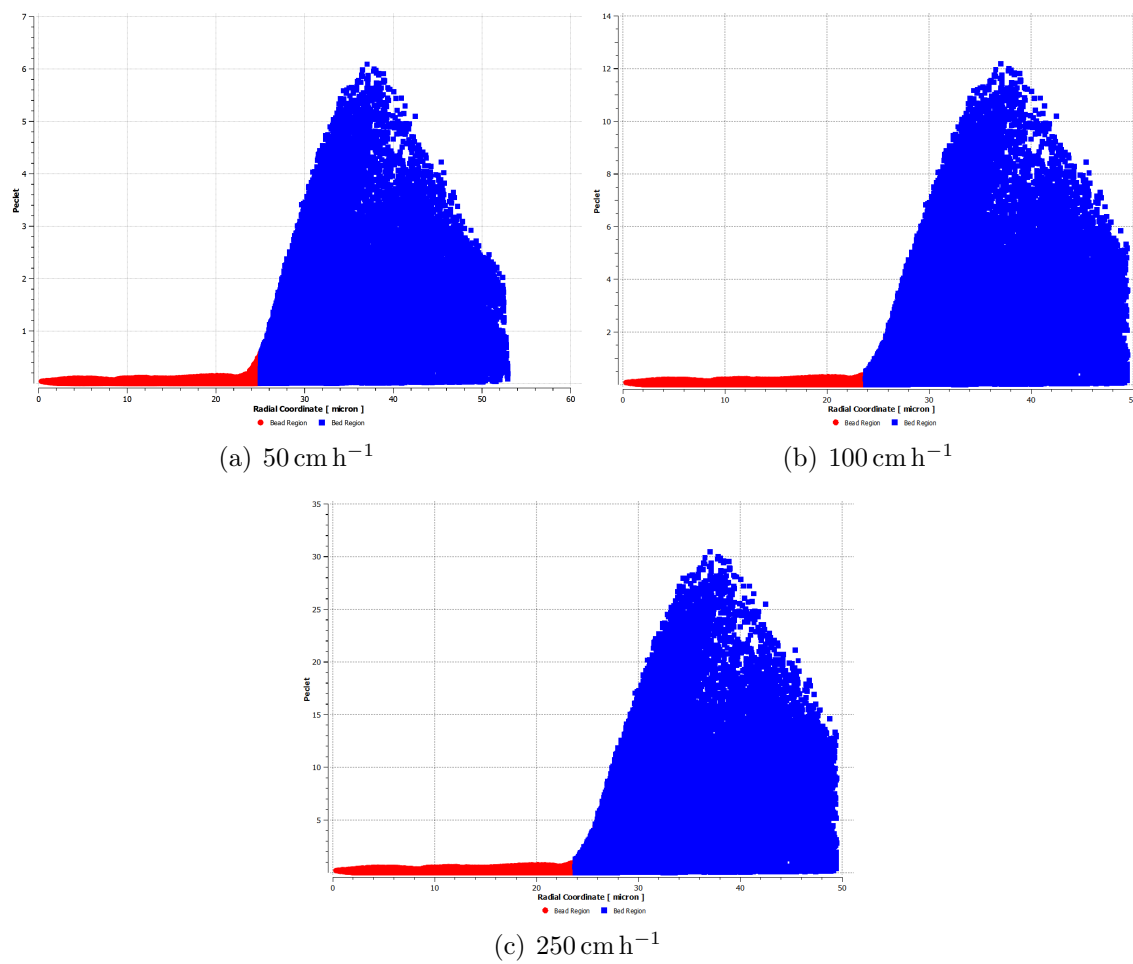


FIGURE 4.31: Contours of the Péclet number in an XY-plane cross section through the center of the BOI.

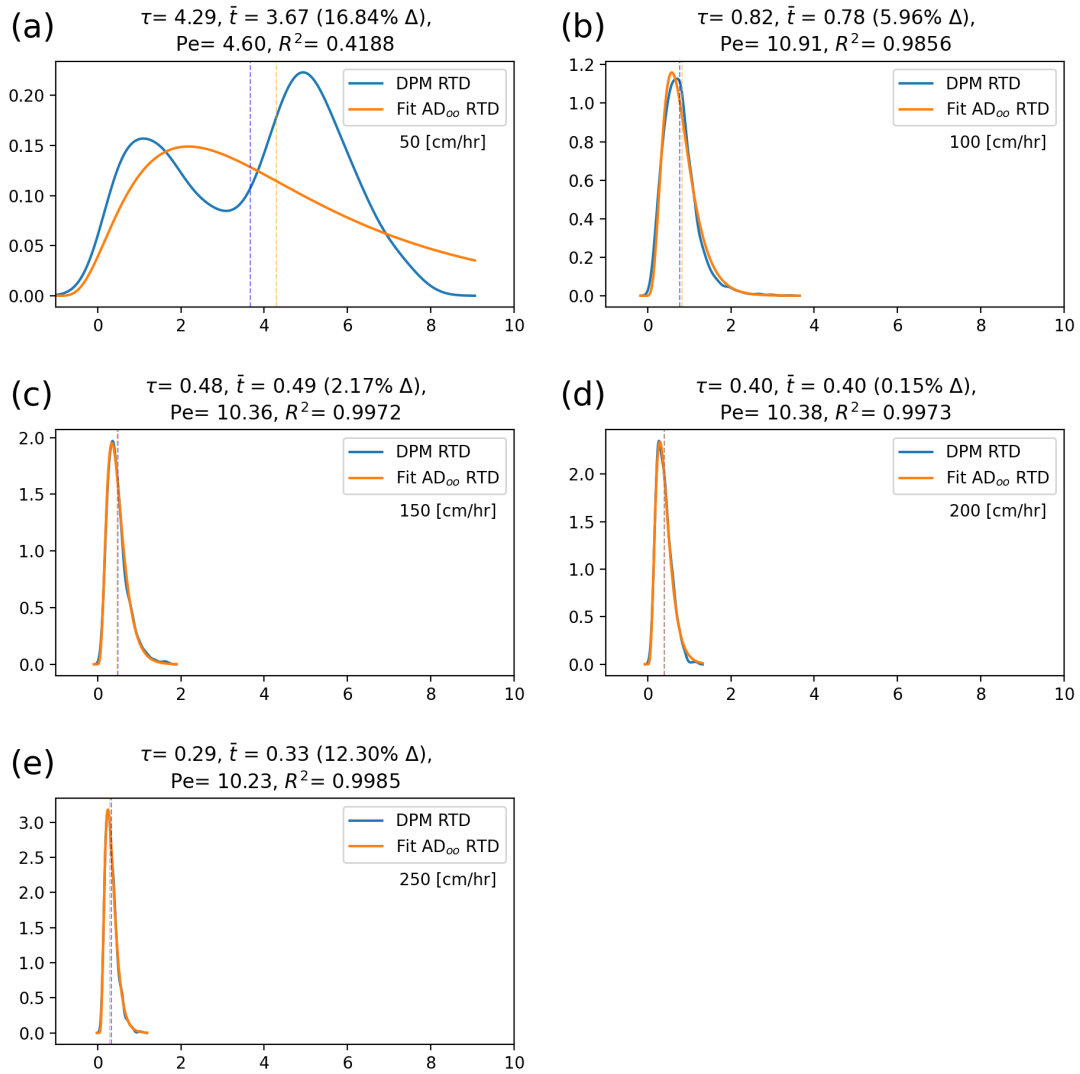


FIGURE 4.32: Levenspiel and Smith's (1957) axial dispersion model with open-open boundary condition RTD fit to the DPM release simulation RTDs using the open-source rtdpy python package (Flamm, 2019).

Chapter 5

Conclusions

I addressed the objective of developing CFD modeling to generate a first-principles model of the chromatographic process while minimizing model parameter estimation's physical resource demand. Specifically, I utilized explicit geometric rendering to develop a CFD steady-state model that simulated fluid flow patterns through and around a perfusion porous resin using a pseudo-packed bed flow cell to predict fluid velocities and shear rates.

I utilized different methods of explicit porous geometry creation and generated computational mesh for CFD simulations. Geometries with continuous lattice structures were the most efficient shapes for mesh generation. Formation of packed-beds using rigid body physics engines was investigated, and contact-point modification handling tools were developed for subsequent CFD modeling steps. However, due to the computational load and desired throughpore resolution, this avenue was ultimately abandoned, and the flow cell model was investigated.

I created a pseudo-packed bed flow cell with a center bead with various explicit porous geometry structures. I then compared the simulated particle permeability with

reported literature values of a commercially available resin in addition to assessing shear within the packed bed. Shear stresses were low within the operating ranges investigated. Additionally, the velocity was relatively uniform, and there was little evidence of eddy formation within the interstitial pores, suggesting that mechanical stresses are not the cause of low yields during chromatography unit operations for large shear sensitive biotherapeutic particles.

I developed a two-part transient CFD DPM based on the aforementioned steady-state model to simulate a tracer protein capture and release from a single bead. Age distribution functions of particle fates were calculated to characterize the macromixing in the model and compared with existing single parameter models. The DPM models showed a packed-bed reactor's distribution profile and provided additional information about the shear forces acting on the particles.

Based on the results presented in this thesis, the following future work is suggested:

- i CFD DPM model development that uses particles with VPL properties to model the transport of viral particles. This model would help assess and develop appropriate unit operations for large biotherapeutic particles without the constraints of facility or material requirements.
- ii CFD DPM models that incorporate porous zones and reaction kinetics into the model. Such an addition would allow modeling of a protein adsorption front and factor in the resin's binding site availability and capacity.
- iii Use of CFD DPM models to explore alternative overall resin topologies and bed packings.

Appendix A

References for Figure 1.2

- [1] A. P. Green et al., “A New Scalable Method for the Purification of Recombinant Adenovirus Vectors,” *Human Gene Therapy*, vol. 13, no. 16, pp. 1921–1934, Nov. 2002, doi: 10.1089/10430340260355338.
- [2] A. Kamen and O. Henry, “Development and optimization of an adenovirus production process,” *The Journal of Gene Medicine*, vol. 6, no. S1, pp. S184–S192, Feb. 2004, doi: 10.1002/jgm.503.
- [3] M. Lusky, “Good Manufacturing Practice Production of Adenoviral Vectors for Clinical Trials,” *Human Gene Therapy*, vol. 16, no. 3, pp. 281–291, Mar. 2005, doi: 10.1089/hum.2005.16.281.
- [4] F. Blanche et al., “An improved anion-exchange HPLC method for the detection and purification of adenoviral particles,” *Gene Therapy*, vol. 7, no. 12, Art. no. 12, Jun. 2000, doi: 10.1038/sj.gt.3301190.
- [5] J. O. Konz, A. L. Lee, J. A. Lewis, and S. L. Sagar, “Development of a Purification Process for Adenovirus: Controlling Virus Aggregation to Improve the Clearance of

- Host Cell DNA,” *Biotechnology Progress*, vol. 21, no. 2, pp. 466–472, 2005, doi: 10.1021/bp049644r.
- [6] G. Vellekamp et al., “Empty Capsids in Column-Purified Recombinant Adenovirus Preparations,” *Human Gene Therapy*, vol. 12, no. 15, pp. 1923–1936, Oct. 2001, doi: 10.1089/104303401753153974.
- [7] B. G. Huyghe et al., “Purification of a Type 5 Recombinant Adenovirus Encoding Human p53 by Column Chromatography,” *Human Gene Therapy*, vol. 6, no. 11, pp. 1403–1416, Nov. 1995, doi: 10.1089/hum.1995.6.11-1403.
- [8] C. Peixoto, T. B. Ferreira, M. J. T. Carrondo, P. E. Cruz, and P. M. Alves, “Purification of adenoviral vectors using expanded bed chromatography,” *Journal of Virological Methods*, vol. 132, no. 1, pp. 121–126, Mar. 2006, doi: 10.1016/j.jviromet.2005.10.002.
- [9] C. Peixoto, T. B. Ferreira, M. F. Q. Sousa, M. J. T. Carrondo, and P. M. Alves, “Towards purification of adenoviral vectors based on membrane technology,” *Biotechnology Progress*, vol. 24, no. 6, pp. 1290–1296, 2008, doi: 10.1002/btpr.25.
- [10] S. Zolotukhin et al., “Production and purification of serotype 1, 2, and 5 recombinant adeno-associated viral vectors,” *Methods*, vol. 28, no. 2, pp. 158–167, Oct. 2002, doi: 10.1016/S1046-2023(02)00220-7.
- [11] M. Urabe et al., “Removal of Empty Capsids from Type 1 Adeno-Associated Virus Vector Stocks by Anion-Exchange Chromatography Potentiates Transgene Expression,” *Molecular Therapy*, vol. 13, no. 4, pp. 823–828, 2006, doi: 10.1016/j.ymthe.2005.11.024.

-
- [12] G. Qu et al., “Separation of adeno-associated virus type 2 empty particles from genome containing vectors by anion-exchange column chromatography,” *Journal of Virological Methods*, vol. 140, no. 1, pp. 183–192, 2007, doi: 10.1016/j.jviromet.2006.11.019.
- [13] N. Kaludov, B. Handelman, and J. A. Chiorini, “Scalable Purification of Adeno-Associated Virus Type 2, 4, or 5 Using Ion-Exchange Chromatography,” *Human Gene Therapy*, vol. 13, pp. 1235–1243, 2002, doi: 10.1089/104303402320139014.
- [14] G. Gao et al., “Purification of Recombinant Adeno-Associated Virus Vectors by Column Chromatography and Its Performance in Vivo,” *Human Gene Therapy*, vol. 11, no. 15, pp. 2079–2091, Oct. 2000, doi: 10.1089/104303400750001390.
- [15] C. R. O’Riordan, A. L. Lachapelle, K. A. Vincent, and S. C. Wadsworth, “Scaleable chromatographic purification process for recombinant adeno-associated virus (rAAV),” *The Journal of Gene Medicine*, vol. 2, no. 6, pp. 444–454, Nov. 2000, doi: 10.1002/1521-2254(200011/12)2:6<444::AID-JGM132>3.0.CO;2-1.
- [16] N. Brument et al., “A Versatile and Scalable Two-Step Ion-Exchange Chromatography Process for the Purification of Recombinant Adeno-associated Virus Serotypes-2 and -5,” *Molecular Therapy*, vol. 6, no. 5, pp. 678–686, Nov. 2002, doi: 10.1006/mthe.2002.0719.
- [17] A. M. Davidoff et al., “Purification of recombinant adeno-associated virus type 8 vectors by ion exchange chromatography generates clinical grade vector stock,” *Journal of Virological Methods*, vol. 121, no. 2, pp. 209–215, Nov. 2004, doi: 10.1016/j.jviromet.2004.07.001.

-
- [18] R. H. Smith, C. Ding, and R. M. Kotin, "Serum-free production and column purification of adeno-associated virus type 5," *Journal of Virological Methods*, vol. 114, no. 2, pp. 115–124, Dec. 2003, doi: 10.1016/j.jviromet.2003.09.002.
- [19] T. Vicente, C. Peixoto, M. J. T. Carrondo, and P. M. Alves, "Purification of recombinant baculoviruses for gene therapy using membrane processes," *Gene Therapy*, vol. 16, no. 6, Art. no. 6, Jun. 2009, doi: 10.1038/gt.2009.33.
- [20] M. Scherr et al., "Efficient gene transfer into the CNS by lentiviral vectors purified by anion exchange chromatography," *Gene Therapy*, vol. 9, no. 24, Art. no. 24, Dec. 2002, doi: 10.1038/sj.gt.3301848.
- [21] K. Yamada, D. M. McCarty, V. J. Madden, and C. E. Walsh, "Lentivirus Vector Purification Using Anion Exchange HPLC Leads to Improved Gene Transfer," *BioTechniques*, vol. 34, no. 5, pp. 1074–1080, May 2003, doi: 10.2144/03345dd04.
- [22] V. Slepishkin et al., "Large-scale purification of a lentiviral vector by size exclusion chromatography or Mustang Q ion exchange capsule. Bioproc. J. September," *Bioprocessing Journal*, vol. 2, no. 5, pp. 89–95, 2003.
- [23] R. H. Kutner, S. Puthli, M. P. Marino, and J. Reiser, "Simplified production and concentration of HIV-1-based lentiviral vectors using HYPERFlask vessels and anion exchange membrane chromatography," *BMC Biotechnol*, vol. 9, no. 1, p. 10, Feb. 2009, doi: 10.1186/1472-6750-9-10.
- [24] T. Rodrigues, A. Carvalho, M. Carmo, M. J. T. Carrondo, P. M. Alves, and P. E. Cruz, "Scaleable purification process for gene therapy retroviral vectors," *The Journal of Gene Medicine*, vol. 9, no. 4, pp. 233–243, 2007, doi: 10.1002/jgm.1021.

-
- [25] T. Rodrigues, A. Carvalho, A. Roldão, M. J. T. Carrondo, P. M. Alves, and P. E. Cruz, “Screening anion-exchange chromatographic matrices for isolation of onco-retroviral vectors,” *Journal of Chromatography B*, vol. 837, no. 1, pp. 59–68, Jun. 2006, doi: 10.1016/j.jchromb.2006.03.061.
- [26] S. Zolotukhin et al., “Recombinant adeno-associated virus purification using novel methods improves infectious titer and yield,” *Gene Therapy*, vol. 6, no. 6, Art. no. 6, Jun. 1999, doi: 10.1038/sj.gt.3300938.
- [27] P. S. Chahal, M. G. Aucoin, and A. Kamen, “Primary recovery and chromatographic purification of adeno-associated virus type 2 produced by baculovirus/insect cell system,” *Journal of Virological Methods*, vol. 139, no. 1, pp. 61–70, Jan. 2007, doi: 10.1016/j.jviromet.2006.09.011.
- [28] C. Wu, K. Y. Soh, and S. Wang, “Ion-Exchange Membrane Chromatography Method for Rapid and Efficient Purification of Recombinant Baculovirus and Baculovirus gp64 Protein,” *Human Gene Therapy*, vol. 18, no. 7, pp. 665–672, Jun. 2007, doi: 10.1089/hum.2007.020.
- [29] M. Kuiper, R. M. Sanches, J. A. Walford, and N. K. H. Slater, “Purification of a functional gene therapy vector derived from Moloney murine leukaemia virus using membrane filtration and ceramic hydroxyapatite chromatography,” *Biotechnology and Bioengineering*, vol. 80, no. 4, pp. 445–453, 2002, doi: 10.1002/bit.10388.
- [30] R. H. Smith, J. R. Levy, and R. M. Kotin, “A Simplified Baculovirus-AAV Expression Vector System Coupled With One-step Affinity Purification Yields High-titer rAAV

- Stocks From Insect Cells,” *Molecular Therapy*, vol. 17, no. 11, pp. 1888–1896, Nov. 2009, doi: 10.1038/mt.2009.128.
- [31] D. Grimm, A. Kern, K. Rittner, and J. A. Kleinschmidt, “Novel Tools for Production and Purification of Recombinant Adenoassociated Virus Vectors,” *Human Gene Therapy*, vol. 9, no. 18, pp. 2745–2760, Dec. 1998, doi: 10.1089/hum.1998.9.18-2745.
- [32] R. Clark, X. Liu, J. Mcgrath, and P. Johnson, “Highly Purified Recombinant Adeno-Associated Virus Vectors Are Biologically Active and Free of Detectable Helper and Wild-Type Viruses,” *Human gene therapy*, vol. 10, pp. 1031–9, May 1999, doi: 10.1089/10430349950018427.
- [33] R. Anderson, I. Macdonald, T. Corbett, A. Whiteway, and H. G. Prentice, “A method for the preparation of highly purified adeno-associated virus using affinity column chromatography, protease digestion and solvent extraction,” *Journal of Virological Methods*, vol. 85, no. 1, pp. 23–34, 2000, doi: 10.1016/S0166-0934(99)00150-0.
- [34] J. D. Harris, S. G. Beattie, and J. G. Dickson, “Novel Tools for Production and Purification of Recombinant Adeno-Associated Viral Vectors,” in *Viral Vectors for Gene Therapy: Methods and Protocols*, C. A. Machida, Ed. Totowa, NJ: Humana Press, 2003, pp. 255–267.
- [35] M. de las M. Segura, A. Kamen, P. Trudel, and A. Garnier, “A novel purification strategy for retrovirus gene therapy vectors using heparin affinity chromatography,” *Biotechnology and Bioengineering*, vol. 90, no. 4, pp. 391–404, 2005, doi: 10.1002/bit.20301.

-
- [36] M. M. Segura, A. Garnier, Y. Durocher, H. Coelho, and A. Kamen, "Production of lentiviral vectors by large-scale transient transfection of suspension cultures and affinity chromatography purification," *Biotechnology and Bioengineering*, vol. 98, no. 4, pp. 789–799, 2007, doi: 10.1002/bit.21467.
- [37] M. W. Wolff et al., "Affinity chromatography of cell culture derived vaccinia virus," Salt Lake City, 2007.
- [38] M. W. Wolff et al., "Capturing of cell culture derived vaccinia virus by membrane adsorbers," Philadelphia, 2008.
- [39] M. Wolff, C. Siewert, S. Lehmann, S. P. Hansen, R. Faber, and U. Reichl, "Cellufine Sulfate and heparin affinity chromatography to capture ce culture-derived Vaccinia virus particles," presented at the 21th ESACT Meeting, 2009, Accessed: Dec. 21, 2020.
- [40] A. Auricchio, E. O'Connor, M. Hildinger, and J. M. Wilson, "A Single-Step Affinity Column for Purification of Serotype-5 Based Adeno-associated Viral Vectors," *Molecular Therapy*, vol. 4, no. 4, pp. 372–374, Oct. 2001, doi: 10.1006/mthe.2001.0462.
- [41] J. T. Koerber, J.-H. Jang, J. H. Yu, R. S. Kane, and D. V. Schaffer, "Engineering Adeno-Associated Virus for One-Step Purification via Immobilized Metal Affinity Chromatography," *Human Gene Therapy*, vol. 18, no. 4, pp. 367–378, Apr. 2007, doi: 10.1089/hum.2006.139.
- [42] Y.-C. Hu, C.-T. Tsai, Y.-C. Chung, J.-T. Lu, and J. T.-A. Hsu, "Generation of chimeric baculovirus with histidine-tags displayed on the envelope and its purification

- using immobilized metal affinity chromatography,” *Enzyme and Microbial Technology*, vol. 33, no. 4, pp. 445–452, Sep. 2003, doi: 10.1016/S0141-0229(03)00143-1.
- [43] C. Jiang, J. C. Glorioso, and M. Ataai, “Presence of imidazole in loading buffer prevents formation of free radical in immobilized metal affinity chromatography and dramatically improves the recovery of herpes simplex virus type 1 gene therapy vectors,” *Journal of Chromatography A*, vol. 1121, no. 1, pp. 40–45, Jul. 2006, doi: 10.1016/j.chroma.2006.04.071.
- [44] C. Jiang et al., “Immobilized Cobalt Affinity Chromatography Provides a Novel, Efficient Method for Herpes Simplex Virus Type 1 Gene Vector Purification,” *Journal of Virology*, vol. 78, no. 17, pp. 8994–9006, Sep. 2004, doi: 10.1128/JVI.78.17.8994-9006.2004.
- [45] K. Ye, S. Jin, M. M. Ataai, J. S. Schultz, and J. Ibeh, “Tagging Retrovirus Vectors with a Metal Binding Peptide and One-Step Purification by Immobilized Metal Affinity Chromatography,” *JVI*, vol. 78, no. 18, pp. 9820–9827, Sep. 2004, doi: 10.1128/JVI.78.18.9820-9827.2004.
- [46] J. H. Yu and D. V. Schaffer, “Selection of Novel Vesicular Stomatitis Virus Glycoprotein Variants from a Peptide Insertion Library for Enhanced Purification of Retroviral and Lentiviral Vectors,” *JVI*, vol. 80, no. 7, pp. 3285–3292, Apr. 2006, doi: 10.1128/JVI.80.7.3285-3292.2006.
- [47] M. C. Cheeks, N. Kamal, A. Sorrell, D. Darling, F. Farzaneh, and N. K. H. Slater, “Immobilized metal affinity chromatography of histidine-tagged lentiviral vectors using monolithic adsorbents,” *Journal of Chromatography A*, vol. 1216, no. 13, pp.

- 2705–2711, Mar. 2009, doi: 10.1016/j.chroma.2008.08.029.
- [48] M. D. Stachler and J. S. Bartlett, “Mosaic vectors comprised of modified AAV1 capsid proteins for efficient vector purification and targeting to vascular endothelial cells,” *Gene Therapy*, vol. 13, no. 11, Art. no. 11, Jun. 2006, doi: 10.1038/sj.gt.3302738.
- [49] M. B. Parrott, K. E. Adams, G. T. Mercier, H. Mok, S. K. Campos, and M. A. Barry, “Metabolically biotinylated adenovirus for cell targeting, ligand screening, and vector purification,” *Molecular Therapy*, vol. 8, no. 4, pp. 688–700, Oct. 2003, doi: 10.1016/S1525-0016(03)00213-2.
- [50] S. L. Williams, D. Nesbeth, D. C. Darling, F. Farzaneh, and N. K. H. Slater, “Affinity recovery of Moloney Murine Leukaemia Virus,” *Journal of Chromatography B*, vol. 820, no. 1, pp. 111–119, Jun. 2005, doi: 10.1016/j.jchromb.2005.03.016.
- [51] S. L. Williams, M. E. Eccleston, and N. K. H. Slater, “Affinity capture of a biotinylated retrovirus on macroporous monolithic adsorbents: Towards a rapid single-step purification process,” *Biotechnology and Bioengineering*, vol. 89, no. 7, pp. 783–787, 2005, doi: 10.1002/bit.20382.
- [52] M. Potter, K. Chesnut, N. Muzyczka, T. Flotte, and S. Zolotukhin, “[24] Streamlined large-scale production of recombinant adeno-associated virus (rAAV) vectors,” in *Methods in Enzymology*, vol. 346, Elsevier, 2002, pp. 413–430.
- [53] J. Transfiguracion, H. Jorio, J. Meghrou, D. Jacob, and A. Kamen, “High yield purification of functional baculovirus vectors by size exclusion chromatography,” *Journal of Virological Methods*, vol. 142, no. 1, pp. 21–28, Jun. 2007, doi: 10.1016/j.jviromet.2007.01.002.

-
- [54] J. Transfiguracion, D. E. Jaalouk, K. Ghani, J. Galipeau, and A. Kamen, “Size-Exclusion Chromatography Purification of High-Titer Vesicular Stomatitis Virus G Glycoprotein-Pseudotyped Retrovectors for Cell and Gene Therapy Applications,” *Human Gene Therapy*, vol. 14, no. 12, pp. 1139–1153, Aug. 2003, doi: 10.1089/104303403322167984.

Appendix B

Contact Modification Python Script

```

1  import bpy
2  import os
3  import sys
4  import mathutils
5  import math
6  import numpy as np
7  import time
8  from scipy.spatial.distance import cdist
9  os.system("cls")
10 filePath = bpy.data.filepath
11 fileDir = os.path.dirname(filePath)
12 particleD= 0.5
13
14 def RelFileNaming (directory, NewFileName):
15     newfile_name = os.path.join( directory , NewFileName)
16     return newfile_name
17 def nonrepeating(a):
18     a=a.reshape(a.shape[:2])
19     #print('combinations',a.shape)
20     #calculate the sqeuclidean distance of all combos
21     dists = cdist(a, a, 'sqeuclidean')
22     #print('cdist',dists)
23     #mask values by distance values that are too far or 0 distance
24     m=np.ma.masked_outside(dists,0.01,0.77).filled(888)
25     #get list of unique values(otherwise repeated)

```

```

26     c,index=np.unique(m,return_index=True)
27     #make sure fill in value of mask is filtered out of the index list
28     minindex=np.ma.masked_where(c==888, index,copy=True)
29     #unravel_index=Converts a flat index or array of flat indices into a tuple of
30     ↪ coordinate arrays
31     #use compressed() to remove masked values
32     b1,b2=np.unravel_index(minindex.compressed(),dists.shape)
33     #print('b1&2',b1,b2)
34     return(b1,b2,m)
35
36 def VertGroupAdd(TargetObj):
37     #make sure active object is selected and string
38     o= bpy.context.object
39     vg = o.vertex_groups.new(name=str('VG_'+TargetObj.name))
40     verts=[]
41     for vert in o.data.vertices:
42         verts.append(vert.index)
43     vg.add(verts, 1.0, 'ADD')
44     return
45
46 def EmptyEasies(obj,act_obj):
47     a_name=str(act_obj.name)
48     #Create Empty with 'Copy Location' and 'Track To' Constraints
49     e=bpy.data.objects.new(str(act_obj.name + '>'+obj.name), None )
50     e.empty_display_type = 'SINGLE_ARROW'
51     L_Constraint = e.constraints.new('COPY_LOCATION')
52     L_Constraint.target = act_obj
53     TT_Constraint = e.constraints.new('TRACK_TO')
54     TT_Constraint.target = obj
55     TT_Constraint.track_axis = 'TRACK_Z'
56     TT_Constraint.up_axis = 'UP_Y'
57     bpy.context.collection.objects.link( e )
58     bpy.context.view_layer.objects.active = bpy.data.objects[a_name]
59     e.select_set(False)
60     bpy.data.objects[a_name].select_set(True)
61     return str(act_obj.name + '>'+obj.name)
62
63 def MeshMods(Obj_1, Obj_2, mDist, DeformFactor, DisplayProxy):
64     Obj_1_name=str(Obj_1.name)
65     Obj_2_name=str(Obj_2.name)

```

```

65     modi=bpy.data.objects[Obj_1_name].modifiers
66     #Select Obj_1
67
68     #bpy.ops.object.select_all(action='DESELECT')
69     bpy.context.view_layer.objects.active= Obj_1
70     bpy.data.objects[Obj_1_name].select_set(True)
71
72     VertGroupAdd(Obj_2) #target object
73
74
75     #Create, select and assign vertex group
76     #bpy.data.objects[Obj_1_name].vertex_groups.new(name=str('VG_'+Obj_2_name))
77     #bpy.ops.mesh.select_mode(type="VERT")
78     #bpy.ops.mesh.select_all(action='SELECT')
79     #bpy.ops.object.vertex_group_select()
80     #bpy.ops.object.vertex_group_assign()
81     #bpy.ops.object.vertex_group_set_active(group=str('VG_'+Obj_2_name))
82
83     #Make vertex weighted proximity modifier and settings
84     SmodiVWP = modi.new(str('VWP_'+Obj_2_name), 'VERTEX_WEIGHT_PROXIMITY')
85
86     #bpy.ops.object.modifier_add(type='VERTEX_WEIGHT_PROXIMITY')
87     # SmodiVWP=modi['VertexWeightProximity']
88     SmodiVWP.vertex_group=str('VG_'+Obj_2_name)
89     SmodiVWP.show_in_editmode= True
90     SmodiVWP.show_on_cage=True
91     SmodiVWP.target = Obj_2
92     SmodiVWP.proximity_mode = 'GEOMETRY'
93     SmodiVWP.proximity_geometry = {'FACE'}
94     SmodiVWP.max_dist = 0.0
95     SmodiVWP.min_dist = mDist
96     SmodiVWP.falloff_type =VWP_falloff_type1
97
98
99     #Make simple deform
100     EmptS=EmptyEasies(bpy.data.objects[Obj_2_name],bpy.data.objects[Obj_1_name])
101     #bpy.ops.object.modifier_add(type='SIMPLE_DEFORM')
102
103
104     SmodiSD=modi.new(str('SD_'+Obj_2_name), 'SIMPLE_DEFORM')

```

```

105     SmodiSD.vertex_group=str('VG_'+Obj_2_name)
106     SmodiSD.deform_method='STRETCH'
107     SmodiSD.factor = DeformFactor
108     SmodiSD.show_in_editmode= True
109     SmodiSD.show_on_cage=True
110     SmodiSD.lock_x=True
111     SmodiSD.lock_y=True
112     SmodiSD.origin = bpy.data.objects[EmptS]
113     #Add Second VWP Modifier
114     if DisplayProxy:
115         SmodiVWP=modi.new(str('VWP_'+Obj_2_name),'VertexWeightProximity')
116         SmodiVWP.vertex_group=str('VG_'+Obj_2_name)
117         SmodiVWP.show_in_editmode= True
118         SmodiVWP.show_on_cage=True
119         SmodiVWP.target=Obj_2
120         SmodiVWP.proximity_mode = 'GEOMETRY'
121         SmodiVWP.proximity_geometry = {'FACE'}
122         SmodiVWP.max_dist = 0.0
123         SmodiVWP.min_dist = mDist
124         SmodiVWP.falloff_type = VWP_falloff_type2
125
126     #Switch back to Obj_2
127     ##bpy.ops.object.modifier_apply(apply_as='DATA', modifier=str('VWP_1'+Obj_2_name))
128     bpy.data.objects[Obj_1_name].select_set(False)
129
130     return
131 TotalStart=time.time()
132 #Mesh Mod Settings
133 VWP_falloff_type1= 'SMOOTH' # 'LINEAR', 'SHARP', 'SMOOTH', 'ROOT',
    ↪ 'ICON_SPHERECURVE', 'RANDOM', 'STEP'
134 VWP_falloff_type2= 'STEP'
135 mDist= 0.06
136 DisplayProxy = False
137 DeformFactor = -0.01
138
139 #list of visible objects and distances
140 naming=[]
141 obj = bpy.context.visible_objects
142 size = len(obj)
143 a=0

```

```

144 origin_xyz= np.ones([size,4,1])
145 for ob in obj:
146     naming+=[ob.name]
147     origin_xyz[a,:,:]=np.array(ob.matrix_world.translation.to_4d()).reshape((4,1))
148     a+=1
149 #print(np.transpose(np.where(np.linalg.norm(origin_xyz - origin_xyz[:,None],
150 ↪ axis=-1)<2.0)).shape)
151 b1,b2,m=nonrepeating(origin_xyz[:,:,:])
152 n_objects=b1.size
153 tracking=[]
154 for i in range(n_objects):
155     Obj1=bpy.data.objects[naming[b1[i]]]
156     Obj2=bpy.data.objects[naming[b2[i]]]
157     MeshMods(Obj1, Obj2, mDist, DeformFactor, DisplayProxy)
158     MeshMods(Obj2, Obj1, mDist, DeformFactor, DisplayProxy)
159     tracking+=[naming[b1[i]] +' ' + naming[b2[i]] +' ' + str(m[b1[i],b2[i]])]
160 TotalEnd=time.time()
161 print(*tracking, sep="\n")
162 #write time to modify file
163 timefile=RelFileNaming(fileDir,"timefile.txt")
164 fo = open(timefile, "w+")
165 fo.write(str("Took %f sec" % ((TotalEnd-TotalStart))))
166 fo.close()
167 bpy.ops.wm.save_as_mainfile(filepath =
168 ↪ RelFileNaming(fileDir,'Relative_MeshMod_bed_X.blend'))
169 bpy.ops.export_mesh.stl(filepath =
170 ↪ RelFileNaming(fileDir,'Relative_MeshMod_bed_X.stl'), check_existing=True,
171 ↪ axis_forward='Y', axis_up='Z', filter_glob="*.stl", use_selection=False,
172 ↪ global_scale=1, use_scene_unit=False, ascii=False, use_mesh_modifiers=True,
173 ↪ batch_mode='OFF')
174
175 # import bpy
176
177 def delThisObj(obj):
178     #bpy.data.collection[0].objects.unlink(obj)
179     bpy.data.objects.remove(obj, do_unlink=True)
180     return
181
182 def FluentGeomPrep():
183     itm=bpy.data.objects.values()

```

```

178     for obj in itm:
179         delThisObj(obj)
180
181     bpy.ops.import_mesh.stl(filepath=
182         ↪ RelFileNaming(fileDir, 'Relative_MeshMod_bed_X.stl'), axis_forward='Y',
183         ↪ axis_up='Z', filter_glob="*.stl", global_scale=1.0, use_scene_unit=True,
184         ↪ use_facet_normal=False)
185
186     for ob in bpy.context.scene.objects:
187         if ob.type == 'MESH':
188             ob.select_set(True)
189             bpy.context.view_layer.objects.active = ob
190
191     for obj in bpy.data.objects:
192         obj.name = 'PackedBed'
193         obj = bpy.data.objects["PackedBed"]
194         bpy.ops.object.origin_set(type='ORIGIN_GEOMETRY')
195         obj.location=0,0,0
196         u = bpy.context.object.dimensions
197         cyl_depth= 13*particleD + u[2]
198         z_loc = -3*particleD
199
200         #bpy.ops.mesh.primitive_cylinder_add(location = (0,0,z_loc),
201         #                                     vertices = 100,
202         #                                     radius = 2.5,
203         #                                     depth = cyl_depth)
204         bpy.ops.wm.save_as_mainfile(filepath =
205             ↪ RelFileNaming(fileDir, 'FluentGeomPrep_MeshMod_bed_X.blend'))
206
207     bpy.ops.export_mesh.stl(filepath = RelFileNaming(fileDir, 'F_MeshMod_bed_X.stl'),
208         ↪ check_existing=True, axis_forward='Y', axis_up='Z', filter_glob="*.stl",
209         ↪ use_selection=False, global_scale=1, use_scene_unit=False, ascii=False,
210         ↪ use_mesh_modifiers=True, batch_mode='OBJECT')
211
212     for area in bpy.context.screen.areas:
213         if area.type == 'VIEW_3D':
214             ctx = bpy.context.copy()
215             ctx['area'] = area
216             ctx['region'] = area.regions[-1]
217             bpy.ops.view3d.view_selected(ctx)
218
219     return

```

211 `FluentGeomPrep()`

Appendix C

UDF Codes

C.1 Interpolate

```
1  #include "udf.h"
2
3
4  DEFINE_EXECUTE_AT_END(particle_info)
5  {
6      Injection *I, *Ilist=Get_dpm_injections();
7      Particle *p;
8      FILE *fp;
9
10     fp = par_fopen("particle-location.dpm","w+",2,2);
11
12     par_fprintf_head(fp,"Here are the particle locations\n");
13
14     loop(I,Ilist)
15     {
16         loop(p,I->p)
17         {
```

```

18         par_fprintf(fp,"%d %d ((%g %g %g %g %g %g %g %g %g %g)
        ↪ my_injection_%d), %g, %g, %g \n", P_INJ_ID(P_INJECTION(p)),
        ↪ p->part_id,P_POS(p)[0], P_POS(p)[1], P_POS(p)[2], P_VEL(p)[0],
        ↪ P_VEL(p)[1], P_VEL(p)[2], P_DIAM(p), P_T(p), P_FLOW_RATE(p),
        ↪ p->stream_index, p->time_of_birth, PP_TIME(p),
        ↪ P_USER_REAL(p,4));
19     }
20 }
21 par_fclose(fp);
22 }

```

C.2 Shear Integral UDF

```

1  #include "udf.h"
2
3  /*****
4  /* Shear rate integral along the path of the particle
5  /* AnsysCustomerPortal/.../Knowledge+Resources/Solutions/FLUENT/2060920 */
6  *****/
7  DEFINE_DPM_SCALAR_UPDATE(srate_integral,cell,thread,initialize,tp)
8  {
9      if (initialize)
10     {
11         TP_USER_REAL(tp,0) = 0.0;
12         TP_USER_REAL(tp,1) = 0.0;
13         TP_USER_REAL(tp,2) = 0.0;
14         TP_USER_REAL(tp,3) = 0.0;
15     }
16
17     else
18     {
19         /*Strain Rate*/
20         TP_USER_REAL(tp,0) += TP_DT(tp) * (C_STRAIN_RATE_MAG(cell,thread) +
        ↪ TP_USER_REAL(tp,1)) / 2.0;
21         TP_USER_REAL(tp,1) = C_STRAIN_RATE_MAG(cell,thread);
22     }

```

```

23      /*Shear Stress*/
24      TP_USER_REAL(tp,2) += TP_DT(tp) *
      ↪ ((C_STRAIN_RATE_MAG(cell,thread)*C_MU_L(cell, thread)) +
      ↪ TP_USER_REAL(tp,3)) / 2.0;
25      TP_USER_REAL(tp,3) = C_STRAIN_RATE_MAG(cell,thread) * C_MU_L(cell, thread);
26
27  }
28 }
29
30 /*****
31 /* 2.5.8.4. Example 2 - Source Code Template modified for Shear rate*/
32 /* https://ansyshelp.ansys.com/account/secured?returnurl=/Views/Secured/corp/v195/flu_
      ↪ _udf/flu_udf_sec_define_dpm_output.html%23flu_udf_dpm_output_ex2
      ↪ */
33 /*****
34
35 #define REMOVE_PARTICLES FALSE
36
37 DEFINE_DPM_OUTPUT(my_dpm_out, header, fp, tp, thread, plane)
38 {
39     if (header)
40     {
41         char *sort_name;
42         char sort_fn[4096];
43
44         if (NNLLP(thread))
45             sort_name = THREAD_HEAD(thread)->dpm_summary.sort_file_name;
46         else if ( ! NNLLP(plane))
47             sort_name = plane->sort_file_name;
48         else /* This is not expected to happen for regular particle sampling.. */
49         {
50             if (dpm_par.unsteady_tracking)
51                 sort_name = "parcels";
52             else
53                 sort_name = "tracks";
54         }
55
56         /* sort_name may contain "/" (Linux)
57          * or ":" and "\" (Windows) --
58          * replace them all by "_":

```

```

59      */
60      strcpy(sort_fn, sort_name);
61      replace_path_chars_in_string(sort_fn);
62      /* prints header of region at top of file*/
63      if (dpm_par.unsteady_tracking)
64          par_fprintf_head(fp, "(%s %d)\n", sort_fn, 13);
65      else
66          par_fprintf_head(fp, "(%s %d)\n", sort_fn, 12);
67
68      #if RP_2D
69          if (rp_axi_swirl)
70              par_fprintf_head(fp, "(x      r      theta      u      v      w");
71          else
72      #endif
73          par_fprintf_head(fp, "(x      y      z      u      v      w");
74
75      if (dpm_par.unsteady_tracking)
76          par_fprintf_head(fp, "    diameter      t      parcel-mass "
77                          "    mass      n-in-parcel      residence_time      flow-time
78                          ↪      name),
79                          Integral_Strain_Rate, Integral_Shear_Stress,
80                          ↪      Injection_Time[s], user_tp_id\n");
81      else
82          par_fprintf_head(fp, "    diameter      t      mass-flow "
83                          "    mass      frequency      time      name)\n");
84  }
85  else if ( ! NULLP(tp))
86  {
87      /* Do some preparatory calculations for later use:
88      */
89      real flow_rate = 0.;
90      real V_vel = TP_VEL(tp)[1];
91      real W_vel = TP_VEL(tp)[2];
92      real Y = TP_POS(tp)[1];
93      real Z = TP_POS(tp)[2];
94      real strength = 0.;
95      real mass = 0.;
96
97      if (TP_INJECTION(tp)->type != DPM_TYPE_MASSLESS)

```

```

96     {
97         mass = TP_MASS(tp);
98
99         if (dpm_par.unsteady_tracking)
100             strength = TP_N(tp);
101         else
102         {
103             strength = TP_FLOW_RATE(tp) / TP_INIT_MASS(tp);
104             if (TP_STOCHASTIC(tp))
105                 strength /= (real)TP_STOCHASTIC_NTRIES(tp);
106         }
107
108         flow_rate = strength * mass;
109     }
110
111     #if RP_2D
112     if (rp_axi_swirl)
113     {
114         Y = MAX(sqrt(TP_POS(tp)[1] * TP_POS(tp)[1] + TP_POS(tp)[2] * TP_POS(tp)[2]),
115             ↪ DPM_SMALL);
116         V_vel = (TP_VEL(tp)[1] * TP_POS(tp)[1] + TP_VEL(tp)[2] * TP_POS(tp)[2]) / Y;
117         W_vel = (TP_VEL(tp)[2] * TP_POS(tp)[1] - TP_VEL(tp)[1] * TP_POS(tp)[2]) / Y;
118         if (Y > 1.e-20) Z = LIMIT_ACOS(TP_POS(tp)[1] / Y);
119     }
120     #endif
121
122     if ( ! dpm_par.unsteady_tracking)
123         par_fprintf(fp,
124             "%d %d ((%e %e %e %e %e %e "
125             " %e %e %e %e %e %e %e) %s:%" int64_fmt " ), %e, %e, %e, %e\n",
126             P_INJ_ID(TP_INJECTION(tp)), TP_ID(tp),
127             TP_POS(tp)[0],
128             Y,
129             Z,
130             TP_VEL(tp)[0],
131             V_vel,
132             W_vel,
133             TP_DIAM(tp),
134             TP_T(tp),
135             flow_rate,

```

```

135         mass,
136         strength,
137         (TP_TIME(tp) -TP_TIME_OF_BIRTH(tp)) /*Residence time*/,
138         TP_TIME(tp)/* flow time */,
139         TP_INJECTION(tp)->name,
140         /*tp->part_id*/ TP_ID(tp),
141         TP_USER_REAL(tp,0) /* Integral_Strain_Rate */,
142         TP_USER_REAL(tp,2) /* Integral_Shear_Stress */,
143         TP_TIME_OF_BIRTH(tp),
144         TP_USER_REAL(tp,4));
145     else
146         par_fprintf(fp, /* Note: The first two arguments to par_fprintf are */
147                     /* used internally and must not be changed, even */
148                     /* though they do not appear in the final output.
149
150                     /* The first two replacement variables in the format string */
151                     /* must be the particle injection ID and particle ID,
152                     ↪ respectively*/
153
154                     "%d %d ((%e %e %e %e %e %e "
155                     " %e %e %e %e %e %e %e) %s:%" int64_fmt " ), %e, %e, %e, %e\n",
156                     P_INJ_ID(TP_INJECTION(tp)), TP_ID(tp),
157                     TP_POS(tp)[0],
158                     Y,
159                     Z,
160                     TP_VEL(tp)[0],
161                     V_vel,
162                     W_vel,
163                     TP_DIAM(tp),
164                     TP_T(tp),
165                     flow_rate,
166                     mass,
167                     strength,
168                     (TP_TIME(tp) -TP_TIME_OF_BIRTH(tp)) /*Residence time*/,
169                     TP_TIME(tp)/* flow time */,
170                     TP_INJECTION(tp)->name,
171                     /*tp->part_id*/ TP_ID(tp),
172                     TP_USER_REAL(tp,0) /* Integral_Strain_Rate */,
173                     TP_USER_REAL(tp,2) /* Integral_Shear_Stress */,
174                     TP_TIME_OF_BIRTH(tp),
175                     TP_USER_REAL(tp,4));

```

```

174
175 #if REMOVE_PARTICLES
176     MARK_TP(tp, P_FL_REMOVED);
177 #endif
178 }
179 }

```

C.3 Monitor Points in Fluent UDF

```

1  #include "udf.h"
2  #include "surf.h"
3
4  #define MAXPOINTS 3000
5
6  real coords[MAXPOINTS][ND_ND];
7  int total_count;
8  cxboolean interpolation_initialized=FALSE;
9
10 #if !RP_HOST
11
12 struct interpolation_point{
13     cell_t c;
14     Thread* t;
15     real distance;
16     cxboolean active;
17     real center[ND_ND];
18 };
19
20 struct interpolation_point point_list[MAXPOINTS];
21
22 #endif
23 /******
24
25 DEFINE_ON_DEMAND(read_points)
26 {
27 #if !RP_HOST

```

```

28     Domain* d = Get_Domain(1);
29     Thread* t;
30     cell_t c;
31     int point, i;
32     real dx,dy,dz=0.0;
33     real xc[ND_ND];
34     real distance;
35     #if PARALLEL
36         real smallestDistance;
37     #endif
38     #endif
39     #if !RP_NODE
40         float x,y,z;
41         FILE* input,fp;
42         int n, m;
43     #endif
44
45     #if !RP_NODE
46
47         for(n=0; n<MAXPOINTS; n++)
48             for(m=0; m<ND_ND; m++)
49                 coords[n][m] = 0.0;
50
51         n = 0;
52
53         input = fopen("points.inp", "r");
54
55         while(!feof(input))
56         {
57             #if RP_3D
58                 fscanf(input,"%g %g %g\n", &x, &y, &z);
59                 coords[n][0]=x; coords[n][1]=y; coords[n][2]=z;
60                 Message("x=%g y=%g z=%g\n",x,y,z);
61             #else
62                 fscanf(input,"%g %g\n", &x, &y);
63                 coords[n][0]=x; coords[n][1]=y;
64                 Message("x=%g y=%g\n",x,y);
65             #endif
66
67             n++;

```

```

68         if (n == MAXPOINTS)
69         {
70             Message("\n\nWARNING: Number of points in input file has exceeded
              ↳ MAXPOINTS, which is set to %i\n",
71                   MAXPOINTS);
72             Message("      Recompile UDF with MAXPOINTS >= number of data
              ↳ points in input file.\n");
73             Message("      ... only %i points will be processed...\n\n",
              ↳ MAXPOINTS);
74             break;
75         }
76     }
77
78     total_count = n;
79     Message("\n\nThere are %i sets of coordinates read from input
              ↳ file.\n",total_count);
80
81     fclose(input);
82
83     #endif
84
85     /* Initialize coordinates on COMPUTE NODES */
86     host_to_node_int_1(total_count);
87     host_to_node_real(&coords[0][0],ND_ND*MAXPOINTS);
88
89     #if !RP_HOST
90     for(point=0; point<total_count; point++)
91     {
92         point_list[point].distance = 1.e+30;
93         point_list[point].active = TRUE;
94         for (i = 0; i < ND_ND; i++)
95         {
96             point_list[point].center[i] = 2.e+30;
97         }
98     }
99
100
101
102     /* Search over all cells */
103     thread_loop_c(t,d)

```



```

104     {
105         begin_c_loop_int(c,t)
106         {
107             C_CENTROID(xc,c,t);
108             for(point=0; point<total_count; point++)
109             {
110                 dx = xc[0]-coords[point][0];
111                 dy = xc[1]-coords[point][1];
112                 #if RP_3D
113                 dz = xc[2]-coords[point][2];
114                 #endif
115                 distance=dx*dx+dy*dy+dz*dz;
116                 if(point_list[point].distance > distance)
117                 {
118                     point_list[point].c = c;
119                     point_list[point].t = t;
120                     point_list[point].distance = distance;
121                     for (i = 0; i < 3; i++)
122                     {
123                         point_list[point].center[i] = xc[i];
124                     }
125                 }
126             }
127         }
128     }
129     end_c_loop_int(c,t);
130 }
131
132 #if PARALLEL
133     for(point=0; point<total_count; point++)
134     {
135         distance = point_list[point].distance;
136         smallestDistance = PRF_GRLow1(distance);
137         if(distance>smallestDistance)
138         {
139             point_list[point].active = FALSE;
140         }
141     }
142 }
143 #endif

```

```

144
145 #endif
146     interpolation_initialized = TRUE;
147 }
148
149 /*****
150
151 DEFINE_ON_DEMAND(get_cell)
152 {
153     Domain *d = Get_Domain(1);
154     Thread *t = NULL; /* Lookup_Thread(d, 2); */
155     cell_t c=0;
156
157     CX_Cell_Id cx_cell;
158     real x[ND_ND]={-0.0020176121, 0.0054349047, 0.030792074}; /* coordinates of
159     ↪ the point of interest*/
160
161     DPM_Init_Oct_Tree_Search(); /* this takes time -- only do once! */
162     /*start loop for multiple points here */
163
164     cx_cell.ct.c = c;
165     cx_cell.ct.t = t;
166     /*****/
167     DPM_Locate_Point(x, &cx_cell,0,0,0);
168     c = RP_CELL(&cx_cell);
169     t = RP_THREAD(&cx_cell);
170     Message("Start get_cell %s, %d \n x=%g, y=%g, z=%g", ( NNULLP(cx_cell.ct.t))
171     ↪ ? "true" : "false",RP_THREAD(&cx_cell),x[0],x[1],x[2]);
172     if ( NNULLP(cx_cell.ct.t))
173     {
174         c = RP_CELL(&cx_cell);
175         t = RP_THREAD(&cx_cell);
176         Message("\n");
177         Message("Thread_ID = %d\n",THREAD_ID(t));
178
179     #if PARALLEL
180         Message("Cell_ID = %d\n",C_ID(c,t));
181
182     #endif
183
184     Message("cell = %d\n",c);
185     Message("c centre = %f, %f, %f\n", C_CENTROID_CACHE(c, t)[0],

```

```

182             C_CENTROID_CACHE(c, t)[1],
183             C_CENTROID_CACHE(c, t)[2]);
184         Message("My ID = %d\n",myid);
185     }
186
187     DPM_End_Oct_Tree_Search(); /* Free memory, only after all points have been
        ↪ processed. */
188 }
189 /*****
190
191 DEFINE_ON_DEMAND(monitor)
192 {
193     /* serial process or node processes calculate the values
194        serial or node0 process write the values
195        https://ansyshelp.ansys.com/account/secured?returnurl=/Views/Secured/corp/v_
        ↪ 195/flu_ug/flu_ug_sec_discrete_file_props.html?q=start%20flowtime%20in%20file
        ↪ */
196     #if !RP_HOST
197         Thread* t;
198         cell_t c;
199         int point, i, n;
200         real values[MAXPOINTS][7],flowtime;
201     #endif
202
203     if(!interpolation_initialized) read_points();
204
205     #if !RP_HOST
206         /* Initialize values */
207         for(i=0; i<MAXPOINTS; i++)
208             for(n=0; n<7; n++)
209                 values[i][n] = 0.0;
210
211         for(point=0; point<total_count; point++)
212         {
213             if(point_list[point].active)
214             {
215                 c = point_list[point].c;
216                 t = point_list[point].t;
217                 /* x, y, z values stored */
218                 values[point][0] = point_list[point].center[0];

```

```

219         values[point][1] = point_list[point].center[1];
220         values[point][2] = point_list[point].center[2];
221         /*distance value stored */
222         values[point][3] = point_list[point].distance;
223         /*u, v, w values stored */
224         values[point][4] = C_U(c,t);
225         values[point][5] = C_V(c,t);
226         values[point][6] = C_W(c,t);
227     }
228     values[point][0] = PRF_GRSUM1(values[point][0]);
229     values[point][1] = PRF_GRSUM1(values[point][1]);
230     values[point][2] = PRF_GRSUM1(values[point][2]);
231     values[point][3] = PRF_GRSUM1(values[point][3]);
232     values[point][4] = PRF_GRSUM1(values[point][4]);
233     values[point][5] = PRF_GRSUM1(values[point][5]);
234     values[point][6] = PRF_GRSUM1(values[point][6]);
235 }
236
237 #if PARALLEL
238     if (I_AM_NODE_ZERO_P)
239 #endif
240     {
241         FILE* output;
242         output = fopen("points.out","w");
243         if(!output)
244         {
245             Message("\n\nERROR: Could not open interpolation output file.\n");
246             return;
247         }
248
249         flowtime = RP_Get_Real("flow-time");
250         fprintf(output, "%g\n", flowtime);
251         fprintf(output, "x_c y_c z_c u v w, distance, x y z \n", flowtime);
252         for(point=0; point<total_count; point++)
253         {
254             fprintf(output,"%g %g %g %g %g %g, %g, %g %g %g \n",
255                 ↪ values[point][0], values[point][1], values[point][2],
256                 values[point][4], values[point][5], values[point][6],
257                 values[point][3],
258                 coords[point][0], coords[point][1], coords[point][2]);

```

```
258         }
259         fprintf(output, "\n");
260         fclose(output);
261     }
262     #endif
263 }
```

Bibliography

- Afeyan, N. B. et al. (Oct. 1990). “Flow-through particles for the high-performance liquid chromatographic separation of biomolecules: perfusion chromatography”. In: *Journal of Chromatography A* 519.1, pp. 1–29. ISSN: 0021-9673. DOI: 10.1016/0021-9673(90)85132-F.
- ANSYS Inc. (2019a). “16.2.1. Equations of Motion for Particles”. In: *Fluent 2019 R3 - Fluent Theory Guide*. 2020-09-28 10:34:36.
- (2019b). *19.5, ANSYS Fluent Theory Guide*. Publication Title: Online Documentation. ANSYS, Inc.
- (2019c). *19.5, ANSYS Fluent User’s Guide*. Publication Title: Online Documentation. ANSYS, Inc.
- ANSYS, Inc (2019). *19.5, ANSYS SpaceClaim*. Version 19.5. Publication Title: Online Documentation.
- Bai, Hua et al. (2009). “A coupled DEM and CFD simulation of flow field and pressure drop in fixed bed reactor with randomly packed catalyst particles”. In: *Industrial and Engineering Chemistry Research* 48.8, pp. 4060–4074. DOI: 10.1021/ie801548h.

- Baker, Matthew John (2011). “CFD simulation of flow through packed beds using the finite volume technique”. Pages: 208 Publication Title: University of Exeter. PhD thesis. 208 pp.
- Baraff, David (1997). “An introduction to physically based modeling: Rigid body simulation II — Nonpenetration Constraints”. In: *SIGGRAPH '97 Course Notes*. Robotics Institute Carnegie Mellon University, pp. D31–D68. ISBN: 0-201-50933-4. DOI: 10.1145/97880.97881.
- Bender, Jan, Kenny Erleben, and Jeff Trinkle (Feb. 2014). “Interactive simulation of rigid body dynamics in computer graphics”. In: *Computer Graphics Forum* 33.1. Publisher: Blackwell Publishing Ltd, pp. 246–270. DOI: 10.1111/cgf.12272.
- Bender, Jan et al. (2012). “Interactive Simulation of Rigid Body Dynamics in Computer Graphics”. In: *Eurographics 2012 - State of the Art Reports*. Ed. by Marie-Paule Cani and Fabio Ganovelli. tex.ids: bender_interactive_2012. The Eurographics Association. DOI: 10.2312/conf/EG2012/stars/095-134.
- Benenati, R. F. and C. B. Brosilow (June 1962). “Void fraction distribution in beds of spheres”. In: *AIChE Journal* 8.3. Publisher: John Wiley & Sons, Ltd, pp. 359–361. DOI: 10.1002/aic.690080319.
- BioCentury Inc. (2019). *Product Profiles*. BCIQ. URL: <https://bciq.biocentury.com/products> (visited on 12/03/2019).

BioMarin Pharmaceutical Inc. (Dec. 23, 2019). *BioMarin Submits Biologics License Application to U.S. Food and Drug Administration for Valoctocogene Roxaparvovec to Treat Hemophilia A*. BioMarin InvestorRoom. URL: <https://investors.biomarin.com/2019-12-23-BioMarin-Submits-Biologics-License-Application-to-U-S-Food-and-Drug-Administration-for-Valoctocogene-Roxaparvovec-to-Treat-Hemophilia-A>.

Blender Developer Wiki (June 2019). *Reference/Release Notes/2.80/Python API*. URL: https://wiki.blender.org/wiki/Reference/Release_Notes/2.80/Python_API.

Blender Foundation (2019). *Blender - a 3D modelling and rendering package*. <http://www.blender.org>. Blender Institute, Amsterdam.

Cleary, Paul W. (1998). “Discrete element modelling of industrial granular flow applications”. In: *TASK. Quarterly-Scientific Bulletin* 2.3, pp. 385–416.

Coumans, Erwin (2005). *Bullet Physics Engine*. Open Source, <https://pybullet.org/wordpress/>.

— (2015). “Bullet-Physics Simulation”. In: *Course at Special Interest Group on Computer Graphics and Interactive Techniques (SIGGRAPH) Conference, August 11, 2015*. Los Angeles, CA, USA: ACM SIGGRAPH 2015 Conference.

Danckwerts, P.V. (Feb. 1953). “Continuous flow systems”. In: *Chemical Engineering Science* 2.1, pp. 1–13. ISSN: 00092509. DOI: 10.1016/0009-2509(53)80001-1.

- De Klerk, Arno (Aug. 2003). “Voidage variation in packed beds at small column to particle diameter ratio”. In: *AIChE Journal* 49.8. Publisher: John Wiley & Sons, Ltd, pp. 2022–2029. DOI: 10.1002/aic.690490812.
- De las Mercedes Segura, María, Amine Kamen, and Alain Garnier (May 1, 2006). “Downstream processing of oncoretroviral and lentiviral gene therapy vectors”. In: *Biotechnology Advances* 24.3, pp. 321–337. ISSN: 0734-9750. DOI: 10.1016/j.biotechadv.2005.12.001.
- De Wilde, Daan et al. (Dec. 2009). “Modelling the relation between the species retention factor and the C-term band broadening in pressure-driven and electrically driven flows through perfectly ordered 2-D chromatographic media”. In: *Journal of Separation Science* 32.23. Publisher: Wiley-Blackwell, pp. 4077–4088. ISSN: 1615-9314. DOI: 10.1002/jssc.200900458.
- Decision Resources Group (Sept. 2019). *Top 10 players in the CAR T-cell therapy development field*. DRG. URL: <https://decisionresourcesgroup.com/downloads/car-t-cell-therapy-pipeline-forecast-snapshot/> (visited on 12/03/2020).
- Di Felice, R. and L. G. Gibilaro (2004). “Wall effects for the pressure drop in fixed beds”. In: *Chemical Engineering Science* 59.14, pp. 3037–3040. DOI: 10.1016/j.ces.2004.03.030.

- Dixon, Anthony G., Michiel Nijemeisland, and E. Hugh Stitt (2013). “Systematic mesh development for 3D CFD simulation of fixed beds: Contact points study”. In: *Computers and Chemical Engineering* 48, pp. 135–153. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2012.08.011.
- Effio, Christopher Ladd and Jürgen Hubbuch (2015). “Next generation vaccines and vectors: Designing downstream processes for recombinant protein-based virus-like particles”. In: *Biotechnology Journal* 10.5, pp. 715–727. ISSN: 1860-7314. DOI: 10.1002/biot.201400392.
- Eppinger, T., K. Seidler, and M. Kraume (2011). “DEM-CFD simulations of fixed bed reactors with small tube to particle diameter ratios”. In: *Chemical Engineering Journal* 166.1, pp. 324–331. DOI: 10.1016/j.cej.2010.10.053.
- Flamm, Matthew H. (Aug. 20, 2019). “rtdpy: A python package for residence time distributions”. In: *Journal of Open Source Software* 4.40, p. 1621. ISSN: 2475-9066. DOI: 10.21105/joss.01621.
- Fogler, H. Scott (2006). *Elements of Chemical Reaction Engineering*. 4th. Hoboken, NJ: Prentice Hall PTR. 1080 pp. ISBN: 978-0-13-047394-3.
- Fuerstenau-Sharp, Maya et al. (2017). “Scalable Purification of Viral Vectors for Gene Therapy: An Appraisal of Downstream Processing Approaches”. In: *BioProcess International* 15.2, pp. 12–17.

- Ginn, Samantha L. et al. (May 2018). “Gene therapy clinical trials worldwide to 2017: An update”. In: *The Journal of Gene Medicine* 20.5. Publisher: John Wiley & Sons, Ltd, e3015–e3015. DOI: 10.1002/jgm.3015.
- Glueckauf, E. (Jan. 1, 1955). “Theory of chromatography. Part 9. The “theoretical plate” concept in column separations”. In: *Transactions of the Faraday Society* 51.0. tex.ids: glueckauf_theory_1955, glueckauf_theory_1955-1, glueckauf_theory_1955-2 publisher: The Royal Society of Chemistry, pp. 34–44. ISSN: 0014-7672. DOI: 10.1039/TF9555100034.
- Goodling, J. S. et al. (1983). “Radial porosity distribution in cylindrical beds packed with spheres”. In: *Powder Technology* 35.1, pp. 23–29. DOI: 10.1016/0032-5910(83)85022-0.
- Gosman, A D and E Loannides (Jan. 1983). “Aspects of Computer Simulation of Liquid-Fueled Combustors”. In: *Journal of Energy* 7.6. Place: Reston, Virigina Publisher: American Institute of Aeronautics and Astronautics, pp. 482–490. DOI: 10.2514/3.62687.
- Gostick, Jeff et al. (July 2016). “OpenPNM: A Pore Network Modeling Package”. In: *Computing in Science & Engineering* 18.4. tex.ids: gostick_openpnm_2016, pp. 60–74. DOI: 10.1109/MCSE.2016.49.

- Grein, Tanja A. et al. (Apr. 2019). “Aeration and Shear Stress Are Critical Process Parameters for the Production of Oncolytic Measles Virus”. In: *Frontiers in Bioengineering and Biotechnology* 7. DOI: 10.3389/fbioe.2019.00078.
- Hecker, Chris (Mar. 1997). “Physics, part 3: Collision Response”. In: *Game Developer Magazine*, pp. 11–18.
- Holdich, Richard G. (2002). “Fluid flow in porous media”. In: *Fundamentals of Particle Technology*. tex.ids: holdich_fluid_2002. Midland Information Technology & Publishing, pp. 21–28. ISBN: 0-9543881-0-0. DOI: 10.1039/df9480300061.
- Horner, M., S. Joshi, and Y. Waghmare (2017). “Process modeling in the biopharmaceutical industry”. In: *Predictive Modeling of Pharmaceutical Unit Operations*. Elsevier Ltd, pp. 383–425. ISBN: 978-0-08-100154-7. DOI: 10.1016/B978-0-08-100154-7.00014-4.
- Hummel, Johannes et al. (2012). “An evaluation of open source physics engines for use in virtual reality assembly simulations”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7432 LNCS. Issue: PART 2, pp. 346–357. ISBN: 978-3-642-33190-9. DOI: 10.1007/978-3-642-33191-6_34.
- Inc., ANSYS (2019a). 19.5, *ANSYS CFD-POST*. Publication Title: Online Doc.
- (2019b). 19.5, *ANSYS Fluent*. Publication Title: Online Doc.

- Kelley, Brian (2007). “Very Large Scale Monoclonal Antibody Purification: The Case for Conventional Unit Operations”. In: *Biotechnology Progress* 23.5, pp. 995–1008. ISSN: 1520-6033. DOI: 10.1021/bp070117s.
- Kodam, Madhusudhan et al. (2010). “Cylindrical object contact detection for use in discrete element method simulations. Part I – Contact detection algorithms”. In: *Chemical Engineering Science* 65.22. Publisher: Elsevier, pp. 5852–5862. DOI: 10.1016/j.ces.2010.08.006.
- Komiyama, Hiroshi and Hakuai Inoue (1974). “Effects of Intraparticle Flow on Catalytic Reactions”. In: *Journal of Chemical Engineering of Japan* 7.4, pp. 281–286. DOI: 10.1252/jcej.7.281.
- Levenspiel, Octave (Nov. 1999). “Chemical Reaction Engineering”. In: *Industrial & Engineering Chemistry Research* 38.11, pp. 4140–4143. ISSN: 0888-5885, 1520-5045. DOI: 10.1021/ie990488g.
- Levenspiel, Octave and W. K. Smith (Apr. 1, 1957). “Notes on the diffusion-type model for the longitudinal mixing of fluids in flow”. In: *Chemical Engineering Science* 6.4, pp. 227–235. ISSN: 0009-2509. DOI: 10.1016/0009-2509(57)85021-0.
- Li, Amy and Goodarz Ahmadi (Jan. 1992). “Dispersion and Deposition of Spherical Particles from Point Sources in a Turbulent Channel Flow”. In: *Aerosol Science and Technology* 16.4, pp. 209–226. DOI: 10.1080/02786829208959550.

- Lyddiatt, Andrew and Deirdre A O’Sullivan (Apr. 1, 1998). “Biochemical recovery and purification of gene therapy vectors”. In: *Current Opinion in Biotechnology* 9.2, pp. 177–185. ISSN: 0958-1669. DOI: 10.1016/S0958-1669(98)80112-2.
- MacMullin, R.B. and M. Weber Jr (1935). “Concept of residence time distribution”. In: *AIChE Journal* 31, p. 409.
- Mahdavi, M., M. Sharifpur, and J. P. Meyer (2018). “Discrete modelling of nanoparticles in mixed convection flows”. In: *Powder Technology* 338. Publisher: Elsevier B.V., pp. 243–252. DOI: 10.1016/j.powtec.2018.07.025.
- Merten, Otto-Wilhelm et al. (2014). “Manufacturing of viral vectors: part II. Downstream processing and safety aspects”. In: *Pharmaceutical Bioprocessing* 2.3, pp. 237–251. DOI: 10.4155/pbp.14.15.
- Millman, K. Jarrod and Michael Aivazis (Mar. 2011). “Python for Scientists and Engineers”. In: *Computing in Science & Engineering* 13.2, pp. 9–12. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.36.
- Moleirinho, Mafalda G. et al. (May 3, 2020). “Current challenges in biotherapeutic particles manufacturing”. In: *Expert Opinion on Biological Therapy* 20.5, pp. 451–465. ISSN: 1471-2598. DOI: 10.1080/14712598.2020.1693541.
- Morenweiser, R. (Oct. 2005). “Downstream processing of viral vectors and vaccines”. In: *Gene Therapy* 12.1. Number: 1 Publisher: Nature Publishing Group, S103–S110. ISSN: 1476-5462. DOI: 10.1038/sj.gt.3302624.

Nolte-‘t Hoen, Esther et al. (Aug. 16, 2016). “Extracellular vesicles and viruses: Are they close relatives?” In: *Proceedings of the National Academy of Sciences* 113.33, pp. 9155–9161. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1605146113.

Nvidia Corporation, Ageia, and NovodeX AG. *PhysX*. <https://developer.nvidia.com/gameworks-physx-overview>.

Ookawara, Shinichi et al. (Sept. 16, 2007). “High-fidelity DEM-CFD modeling of packed bed reactors for process intensification”. In: European Congress of Chemical Engineering (ECCE-6). Copenhagen, p. 11.

Ounis, Hadj, Goodarz Ahmadi, and John B McLaughlin (1991). *Brownian Diffusion of Submicrometer Particles in the Viscous Sublayer*.

Panda, Priyadarshi et al. (2008). “Stop-flow lithography to generate cell-laden microgel particles”. In: *Lab on a Chip* 8.7. Publisher: Royal Society of Chemistry, pp. 1056–1061. DOI: 10.1039/b804234a.

Partopour, Behnam and Anthony G. Dixon (2017). “An integrated workflow for resolved-particle packed bed models with complex particle shapes”. In: *Powder Technology* 322. Publisher: Elsevier B.V., pp. 258–272. DOI: 10.1016/j.powtec.2017.09.009.

Poletti, Valentina and Fulvio Mavilio (Mar. 2018). “Interactions between Retroviruses and the Host Cell Genome”. In: *Molecular Therapy - Methods and Clinical Development* 8. Publisher: Cell Press, pp. 31–41. DOI: 10.1016/j.omtm.2017.10.001.

Python (2018). *Python 3.7.0*. <https://www.python.org/>. [Online].

- Rathore, Anurag S (Nov. 1, 2014). “QbD/PAT for bioprocessing: moving from theory to implementation”. In: *Current Opinion in Chemical Engineering*. Biotechnology and bioprocess engineering • Process systems engineering 6, pp. 1–8. ISSN: 2211-3398. DOI: 10.1016/j.coche.2014.05.006.
- Rathore, Anurag S and Vijesh Kumar (Apr. 2017). “Mechanistic Modeling of Preparative Ion-Exchange Chromatography”. In: 30.4, pp. 41–45.
- Roblee, L. H.S., R. M. Baird, and J. W. Tierney (Dec. 1958). “Radial porosity variations in packed beds”. In: *AIChE Journal* 4.4. Publisher: John Wiley & Sons, Ltd, pp. 460–464. DOI: 10.1002/aic.690040415.
- Rodrigues, Alírio E. et al. (Jan. 24, 1992). “Importance of intraparticle convection in the performance of chromatographic processes”. In: *Journal of Chromatography A*. Eight International Symposium on Preparative Chromatography 590.1, pp. 93–100. ISSN: 0021-9673. DOI: 10.1016/0021-9673(92)87009-W.
- Roldão, António et al. (Oct. 1, 2010). “Virus-like particles in vaccine development”. In: *Expert Review of Vaccines* 9.10. tex.ids: roldao_virus-like_2010 publisher: Taylor & Francis, pp. 1149–1176. ISSN: 1476-0584. DOI: 10.1586/erv.10.115.
- Roosendaal, Ton (Jan. 2019). *Blender’s 25th birthday!* Publication Title: blender.org. URL: <https://www.blender.org/press/blenders-25th-birthday/>.

- Rosenberg, Steven A. et al. (Aug. 1990). “Gene Transfer into Humans — Immunotherapy of Patients with Advanced Melanoma, Using Tumor-Infiltrating Lymphocytes Modified by Retroviral Gene Transduction”. In: *New England Journal of Medicine* 323.9, pp. 570–578. DOI: 10.1056/NEJM199008303230904.
- Ruthven, Douglas M (1984). *Principles of Adsorption and Adsorption Processes*. 1st. tex.ids: ruthven_principles_1984, ruthven_principles_1984-2 pages: 464. Ney York: John Wiley & Sons, Inc. 464 pp. ISBN: 978-0-471-86606-0 0-471-86606-7.
- Saffman, P G (June 1965). “The lift on a small sphere in a slow shear flow”. In: *Journal of Fluid Mechanics* 22.2, pp. 385–400. DOI: 10.1017/S0022112065000824.
- Seisenberger, Georg et al. (2001). “Real-Time Single-Molecule Imaging of the Infection Pathway of an Adeno-Associated Virus”. In: 294 (November). tex.ids: seisenberger_real-time_2001-1, pp. 1929–1932. ISSN: 0036-8075 (Print)\n0036-8075 (Linking). DOI: 10.1126/science.1064103.
- Shekhawat, Lalita K. and Anurag S. Rathore (July 2019). “An overview of mechanistic modeling of liquid chromatography”. In: *Preparative Biochemistry and Biotechnology* 49.6. Publisher: Taylor and Francis Inc., pp. 623–638. DOI: 10.1080/10826068.2019.1615504.
- Smith, Russ. *Open Dynamics Engine*. URL: <http://www.ode.org/>.

- Smits, Wim, Kazuki Nakanishi, and Gert Desmet (2016). “The chromatographic performance of flow-through particles: A computational fluid dynamics study”. In: *Journal of Chromatography A* 1429, pp. 166–174. DOI: 10.1016/j.chroma.2015.12.019.
- Suzuki, Motoyuki (1990). In: *Adsorption engineering*. Chemical engineering monographs vol. 25. 97-99. Tokyo : Amsterdam ; New York: Kodansha ; Elsevier, pp 97–99. ISBN: 978-0-444-98802-7.
- T. A. P. Ltd. *True Axis*. <https://trueaxis.com/>.
- Taylor, Ben (June 12, 2010). *Viral Tegument*. Publication Title: Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Viral_Tegument.svg.
- Van Dongen, Helena M. et al. (June 2016). “Extracellular Vesicles Exploit Viral Entry Routes for Cargo Delivery”. In: *Microbiology and Molecular Biology Reviews* 80.2, pp. 369–386. ISSN: 1092-2172, 1098-5557. DOI: 10.1128/MMBR.00063-15.
- Van Gelder, Klaas B. and K. Roel Westerterp (1990). “Residence time distribution and hold-up in a cocurrent upflow packed bed reactor at elevated pressure”. In: *Chemical Engineering & Technology* 13.1, pp. 27–40. DOI: 10.1002/ceat.270130106.
- Van Gumster, J. (2015). *Blender For Dummies*. For Dummies. Pages: 528. Wiley. 528 pp. ISBN: 978-1-119-03953-2.
- Virtanen, Pauli et al. (2020). “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature Methods* 17.3. tex.ids: 2020SciPy-NMeth tex.adsurl:

- <https://rdcu.be/b08Wh> number: 3 publisher: Nature Publishing Group, pp. 261–272.
ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2.
- Walker, John M. (2011). *Viral Vectors for Gene Therapy*. Ed. by Otto-Wilhelm Merten and Mohamed Al-Rubeai. Vol. 737. Methods in Molecular Biology. Issue: 10 Pages: 2015. Totowa, NJ: Humana Press. 2013 pp. ISBN: 978-1-61779-094-2. DOI: 10.1007/978-1-61779-095-9.
- Waskom, Michael and the seaborn development team (Sept. 2020). *mwaskom/seaborn*. Version latest. DOI: 10.5281/zenodo.592845.
- Wehinger, Gregor D, Carsten Fütterer, and Matthias Kraume (2017). “Contact modifications for CFD simulations of fixed-bed reactors: Cylindrical particles”. In: *Industrial and Engineering Chemistry Research* 56.1, pp. 87–99. DOI: 10.1021/acs.iecr.6b03596.
- Wen, Chin-Yung, Liang-tseng Fan, and others (1975). *Models for flow systems and chemical reactors*. M. Dekker.
- Williams, J R and R O’Connor (1999). *Archives of Computational Methods in Engineering Discrete Element Simulation and the Contact Problem*. Volume: 6, pp. 279–304.
- Wirth, Thomas, Nigel Parker, and Seppo Ylä-Herttuala (2013). “History of gene therapy”. In: *Gene* 525.2. Publisher: Elsevier B.V., pp. 162–169. ISSN: 1879-0038. DOI: 10.1016/j.gene.2013.03.137.

-
- Wittig, Kay, Andreas Richter, and Petr A. Nikrityuk (2012). “Numerical Study of Heat and Fluid Flow Past a Cubical Particle At Subcritical Reynolds Numbers”. In: *Computational Thermal Sciences* 4.4, pp. 283–296. DOI: 10.1615/ComputThermalScien.2012005098.
- Wrzosek, Katarzyna et al. (2013). “Modeling of equilibrium and kinetics of human polyclonal immunoglobulin G adsorption on a tentacle cation exchanger”. In: *Chemical Papers*, p. 11.
- Young, M. E., P. A. Carroad, and R. L. Bell (1980). “Estimation of diffusion coefficients of proteins”. In: *Biotechnology and Bioengineering* 22.5, pp. 947–955. ISSN: 1097-0290. DOI: 10.1002/bit.260220504.