AQUAPORIN

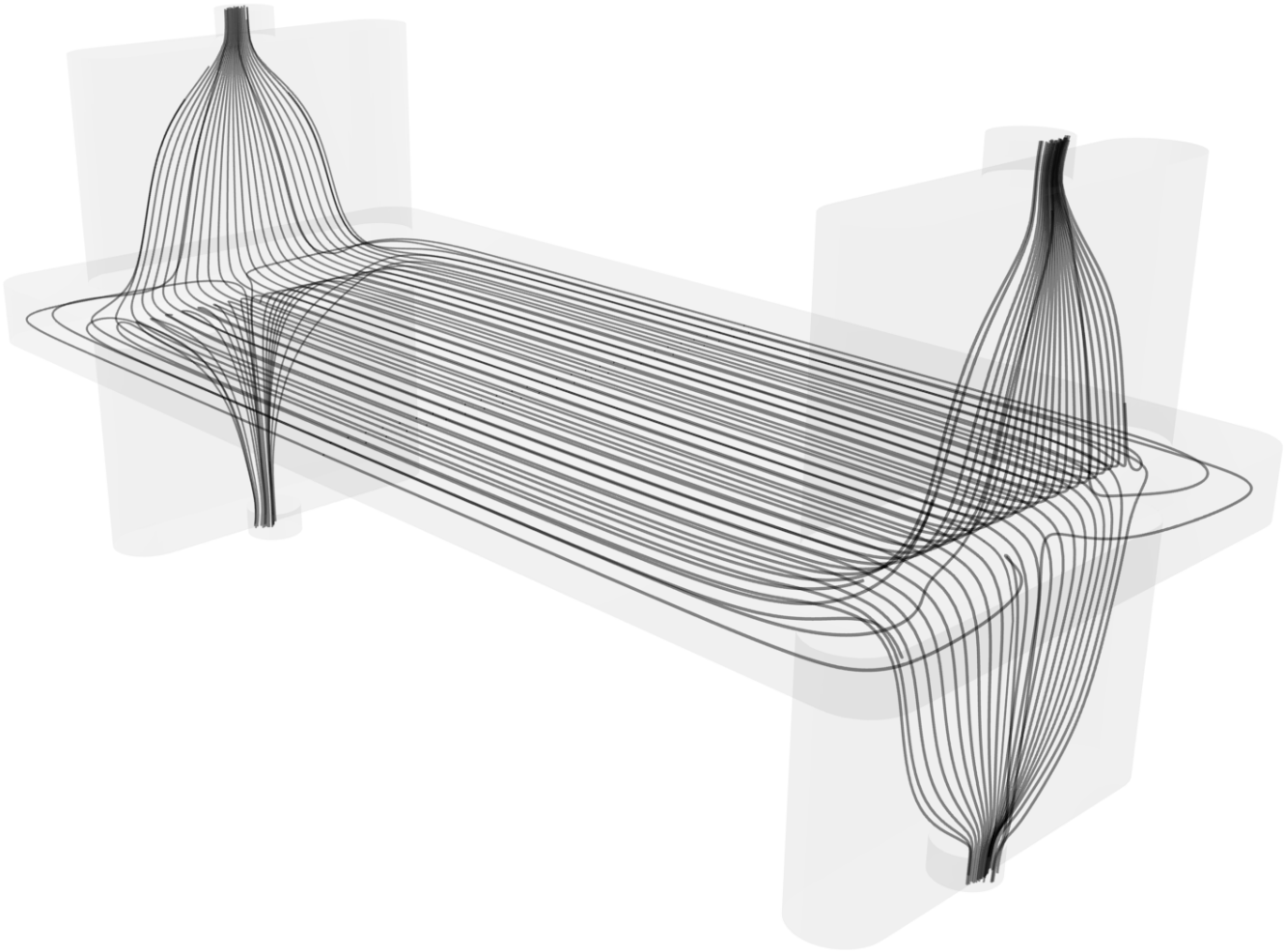
Technical University of Denmark

DTU

Computational Fluid Dynamics Simulations of Forward Osmosis Membrane Modules

Bachelor Thesis

Ulf Aslak



Supervisors:
Claus Hélix-Nielsen & Mathias Felix Gruber

Student 103072

January 15, 2014

Preface

This thesis is submitted as a final requirement for obtaining the degree of Bachelor in Physics & Nanotechnology at the Technical University of Denmark. It was carried out from september 2013 to january 2014 in collaboration with DTU and Aquaporin A/S. Simulations were done using the DTU High Performance Computing center, while my physical appearance was mostly at Aquaporin A/S. The project was supervised by Claus Hélix-Nielsen (main supervisor, DTU Physics & Aquaporin A/S) and Mathias Felix Gruber (co-supervisor, DTU Physics).

I want to thank the kind people at Aquaporin A/S for creating an inspiring and professional working environment. I thank Claus Hélix-Nielsen for making time in his busy weeks to engage actively with my work, and Mathias F. Gruber for his invaluable guidance along the way. I furthermore thank Sylvie Braekevelt and Søren Mentzel for their experimental work, and Niada Bajraktari, Mads F. Andersen and everyone else at Aquaporin that I shared my long work days with, for making this a very worthwhile experience.

Finally, I thank Vivi Lai for having tremendous amounts of patience with me through this project. You mean the world to me.

Ulf Aslak
January 15th, 2014

Abstract

The work documented in this thesis was performed with two goals in mind. The first goal was to develop and thoroughly document a generic working process for simulating flow in complex three-dimensional (3D) geometries with very high mesh resolution in specific regions. The second goal was to acquire a better understanding of how parameter setting and geometry can be optimized, to promote higher trans-membrane water fluxes in Forward Osmosis (FO) filtration.

FO is an osmotically driven filtration process, that relies on a large osmotic pressure differential across a semi-permeable membrane, to transport solvent through the membrane. It is an emerging technology with a great number of potential applications, and unlike the broadly applied Reverse Osmosis (RO) filtration process, it requires no external pressurization, which makes it potentially more energy-effective.

In its field of research, this work stands out because it analyses the FO flow problem in (3D), whereas previous efforts have mainly been focused on one- and two-dimensional analysis. This is done by means of Computational Fluid Dynamics (CFD) simulations using the open source software package, OpenFOAM. A previously validated CFD model specifically developed for simulation of FO filtration in OpenFOAM was used. Four series of simulations were carried out for FO filtration in the commercial Sterlitech CF042 membrane module, to determine how variation of key parameters would impact the flow conditions in the module.

Valuable knowledge about the flow patterns in the module was obtained. It was shown how the flux reducing phenomenon, Concentration Polarization (CP), occurred in the vicinity of the membrane surface, and discussed what measures could be taken to reduce its severity. Comparison of simulation results with analytical data provided further validation of the CFD model used. It was finally argued that certain aspects of the module design could be reconsidered such as to improve flow conditions and promote higher trans-membrane water fluxes.

Resumé

I arbejdet med dette projekt, har jeg haft to særlige mål for øje. Det første, at eftervise en generel måde hvorpå numeriske fluiddynamik-simuleringer (CFD-simuleringer) kan udføres for osmotiske membranmoduler, hvor områder af den virtuelt gengivne geometri har meget høj opløsning. Den anden målsætning har været at opnå en bedre forståelse for hvordan forskellige parametre i Direkte Osmose (DO) filtrering kan justeres for at skabe optimale strømningsbetingelser inde i et modul.

DO er en osmotisk dreven filtreringsproces, som afhænger af en stor osmotisk trykgradient på tværs af en semipermeabel membran, for at kunne transportere vand igennem den. Det er en teknologi under stor udvikling, som potentielt kan anvendes i mange forskellige henseender. DO skiller sig ud fra konventionel filtreringsteknologi, så som Omvendt Osmose (RO), fordi der ikke skal påføres noget hydraulisk tryk for at filtrering kan finde sted, hvorfor DO potentielt set er mere energieffektivt.

Dette projekt skiller sig ud, fordi det analyserer strømning i DO moduler ved brug af tredimensionelle (3D) CFD-simuleringer, hvor det øvrige videnskabelige arbejde i dette felt har fokuseret primært på en- og todimensionel analyse. Det gratis og åbent-kodede CFD-simuleringsprogram OpenFOAM er brugt til dette formål. En valideret CFD model, specifikt udviklet til simulering af strømning i DO systemer, blev brugt. I alt fire serier af simuleringer blev udført for DO filtrering i det populære Sterlitech CF042 membran modul, hver af dem for at undersøge hvordan variation af en enkelt parameter ville påvirke strømningen.

Værdifuld indsigt i strømningsmønstrene under forskellige betingelser blev opnået. Det blev vist hvordan fænomenet Koncentrations Polarisering (CP) havde negativ indflydelse på vandstrømning igennem membranen, og ydermere undersøgt hvilke tiltag der kunne gøres for at reducere dens effekt. Sammenligning mellem resultater fra simuleringerne med analytiske resultater bidrog med yderligere validering af den brugte CFD model. Det blev til sidst diskuteret hvilke aspekter af moduldesignet der kunne genovervejes, for at fremme vandstrømning gennem membranen.

Table of Contents

List of Abbreviations

Abbreviation	Interpretation
2D	Two-Dimensional
3D	Three-Dimensional
AL-DS	Active Layer facing Draw Solution
AL-FS	Active Layer facing Feed Solution
BC	Boundary Condition
CFD	Computational Fluid Dynamics
CP	Concentration Polarization
CV	Control Volume
DO	Direkte Osmose
ECP	External Concentration Polarization
FO	Forward Osmosis
FVM	Finite Volume Method
GAMG	Geometric-Algebraic Multi-Grid
GCI	Grid Convergence Index
GUI	Graphic User Interface
ICP	Internal Concentration Polarization
PISO	Pressure Implicit Splitting of Operators
RO	Reverse Osmosis

List of Symbols

Latin	Description	Unit
A	pure water permeability coefficient	$\mathrm{m\,(s\,Pa)^{-1}}$
B	solute permeation coefficient	$\mathrm{m\,s^{-1}}$
c	solute concentration	$\mathrm{kg\,m^{-3}}$
D	general diffusion coefficient	$\mathrm{m^2\,s^{-1}}$
D_{AB}	binary solute diffusion coefficient	$\mathrm{m^2\,s^{-1}}$
e	relative error	-
$\mathbf{g}$	gravitational acceleration vector	$\mathrm{m\,s^{-2}}$
h	height of `letShell` patch	m
H	height of `shell` patch	m
J_s	trans-membrane solute flux	$\mathrm{g\ m^{-2}\ h^{-1}}$
J_w	trans-membrane water flux	$\mathrm{L\ m^{-2}\ h^{-1}}$
$\mathbf{J}_s$	trans-membrane solute flux vector	$\mathrm{g\ m^{-2}\ h^{-1}}$
$\mathbf{J}_w$	trans-membrane water flux vector	$\mathrm{L\ m^{-2}\ h^{-1}}$
k_f	mass transfer coefficient in the flow	$\mathrm{m\,s^{-1}}$
K	diffusion resistivity / mass transfer coefficient	$\mathrm{s\,m^{-1}}$
l	length from edge of `shell` to edge of `letShell`	m
L	length of `shell` patch	m
m_A	solute mass fraction	$\mathrm{kg\,kg^{-1}}$
$\hat{\mathbf{n}}_w$	unit vector in pure water flow direction	-
p	pressure	Pa
p_{rgh}	hydrostatic-free pressure	Pa
Q	cross flow rate	$\mathrm{mL\ min^{-1}}$
R_g	universal gas constant	$\mathrm{J\,(K\,mol)^{-1}}$
$\mathbb{R}$	cell ratio between fine mesh and coarse mesh	-
S	membrane structural parameter	m
t	time	s
t_s	thickness of porous support layer	m
T	temperature	K
u	x-component of velocity	$\mathrm{m\,s^{-1}}$
U	magnitude of flow velocity	$\mathrm{m\,s^{-1}}$

$\mathbf{U}$	fluid velocity vector	$\mathrm{m\,s^{-1}}$
$\bar{U}$	mean cross-flow velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{U}_{\mathrm{slip}}$	surface slip velocity	$\mathrm{m\,s^{-1}}$

Greek	Description	Unit
α	slip coefficient	-
δ	unstirred boundary layer	m
Δ	difference between values of	-
κ	permeability	$\mathrm{m^2}$
μ	viscosity of fluid	$\mathrm{Pa\,s}$
π	osmotic pressure	Pa
ρ	fluid density	$\mathrm{kg\,m^{-3}}$
τ	membrane tortuosity	-

Subscript	Description
A	value for solute
B	value for solvent
bulk	bulk value
D	draw side of membrane
ECP	value for ECP
eff	effective value
F	feed side of membrane
f	value in flow
m	membrane surface value
S	value in porous support

1 Introduction

With the steady increase in energy prices and the high demand for potable water from an increasing world population, it has become of high priority to find ways to reduce the energy consumption of water desalination processes. As of today almost 15.000 desalination plants are in operation world-wide with a total output of over 50 million cubic meters of fresh water *every day*. If these numbers are to rise in the coming years as expected, the pressing need for better technology is undeniable [1].

In recent years, Forward Osmosis (FO) technology has shown great potential to help solve this problem. FO makes use of the naturally occurring osmotic pressure difference between solutions of different solute concentration, to establish a flow of clean water from a contaminated *feed solution* into a *draw solution*. Compared to conventional pressure-driven filtration techniques, such as Reverse Osmosis (RO), FO is potentially more cost-effective, since it requires no external pressurizing.

One of the main challenges in developing this technology is to design optimal filtration modules, both for industrial operation and research applications. So far, a number of experimental methods, analytical models and two-dimensional simulation approaches have been applied to promote a scientific understanding of FO technology [2–7].

With this challenge in mind, an important goal of this thesis is to develop and thoroughly document a generic working process for simulating flow in complex three-dimensional (3D) geometries with very high mesh resolution in specific regions. Simulations are documented for flow in the Sterlitech CF042 membrane module, using the open source CFD software package OpenFOAM®, with a CFD model developed by M. F. Gruber et. al. [8].

A second goal of this thesis is to acquire a better understanding of how parameter setting and geometry can be optimized, to promote higher transmembrane water fluxes in FO filtration. To meet this goal, results from four series of simulations for different varying parameters are analyzed and discussed.

At present, the methods applied in this work form the sole basis for simulating 3D flow in FO membrane modules. It is therefore the hope of this author that this thesis may serve as a guiding tool for future students, who wish to perform CFD simulations in complex 3D geometries, possibly for FO filtration.

2 Theory

The underlying theory that enables one to fully understand membrane technology and Computational Fluid Dynamics (CFD) methods extends much further than what is presented in the following chapter. I present here only the most necessary parts of the theory that the reader should be acquainted with in order to interpret my results and the conclusions that I make. This chapter may thus be considered a brief compilation of the most relevant theory for the subjects that I include in this thesis.

2.1 Membrane Filtration

Membrane filtration is a widely applied technique that is used for removal of solvent molecules (often water) from a solution containing one or more solute species (such as salt). In this text *water* and *salt* are used interchangeably with *solvent* and *solute*. The way filtration works is quite simple. Like coffee through a filter, a semipermeable membrane works to allow passage for certain particles or chemicals, while rejecting others. In scientific language the unfiltered coffee would then be the *feed solution* while the fresh coffee would be the *permeate*. The force that drives this process can be e.g. a concentration gradient, a hydraulic pressure difference, a temperature difference or an electrical potential.

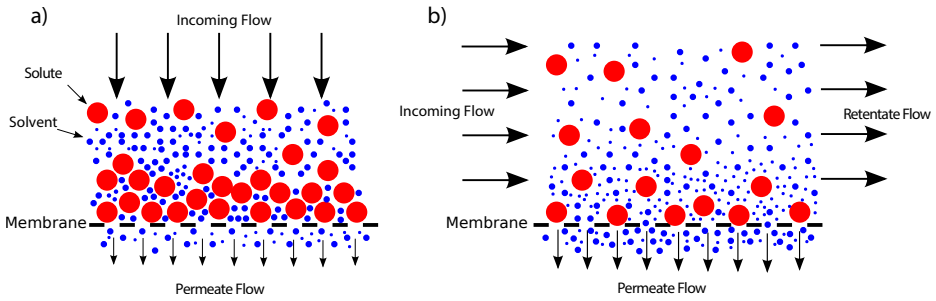


Figure 2.1: *Sketches showing the difference between* a) *dead-end filtration where the feed flow is forced directly into the membrane which accumulates solute molecules, and* b) *cross-flow filtration where the feed flow travels tangential to the membrane, reducing the accumulation of solute.*

The two main modes of filtration are *dead-end filtration* and *cross-flow filtration*. As Fig. 2.1 illustrates, the main difference between the two filtration types is the direction of the incoming feed flow. Dead-end filtration can be considered a "brute force" filtration mode, where the feed solution is forced through the membrane, while cross-flow filtration is a more "gentle" process that allows the solvent to either pass through the membrane and into the permeate, or to continue in the *retentate* stream, as it flows by tangential to the membrane. Choice

of method depends on the given application, however the cross-flow filtration mode is more popular, because it reduces the need to address problems with particle accumulation at the membrane, causing pore plugging i.e. *membrane fouling*. Furthermore it can be run continuously as opposed to the batch-wise operation restriction that lies on dead-end filtration mode.

Membrane filtration processes undergo further sectioning by characteristics such as what phases they separate (gas or liquid), the size of the particles that are rejected by the membrane, and the choice of driving force, all of which forms the basis for a large variety of different processes.

2.1.1 Forward Osmosis Separation

Forward Osmosis (FO) is a separation process that uses the concept of osmosis to extract solvent from the feed solution. Osmosis is a natural phenomenon commonly associated with water transport through cell membranes in live organisms. It is defined as the net movement of solvent molecules through a semipermeable membrane from one solution into another of higher solute concentration, in order to equalize the chemical potentials of the two solutions. It is quantified by the amount external pressure that must be applied to the more concentrated solution for all net movement of solvent through the membrane to stop. This quantity is the *osmotic pressure* and can be calculated for dilute solutions using the van't Hoff equation [9]:

$$\pi \approx c_A R_g T \tag{2.1}$$

where c_A is the molar concentration of an ideal solute, R_g is the universal gas constant and T is the absolute temperature.

In FO, an osmotic pressure gradient across the membrane is used as the driving force. This is done by introduction of a highly concentrated *draw solution*. The membrane chosen for this process commonly has an asymmetric design with a thin, dense active separation layer on one side and a porous support layer on the other. Depending on whether the membrane is oriented such that the active layer is facing the feed side or the draw side, it is labeled AL-FS or AL-FD, respectively. The chosen orientation depends on the given application, however, for FO it is common to use the AL-FS orientation, in order to avoid plugging of the support pores by feed contamination [3].

FO has emerged as popular filtration method because it works almost without the need for external pressurizing, contrasting the conventional and broadly applied process *Reverse Osmosis* (RO). In RO the feed solution is pressurized and thus forced through an asymmetric membrane. The natural occurrence of an osmotic pressure gradient must then be counteracted by the applied hy-

draulic pressure and it is therefore a very energy consuming separation process. Though RO is still the filtration process of choice in huge market areas such as water desalination, many have proposed FO as a serious alternative, due to its potentially high cost-effectiveness. There are, however, still many challenges to address, such as membrane development, post-filtration treatment of draw solutions and filtration module design.

2.1.2 Concentration Polarization

In a process such as FO that is engineered to promote transport of only solvent through the membrane, it is inherent that there will be an accumulation of solute concentration at one side of the membrane, and a dilution at the other. This effect is known as *Concentration Polarization* (CP). Fig. 2.2 illustrates the concept of CP.

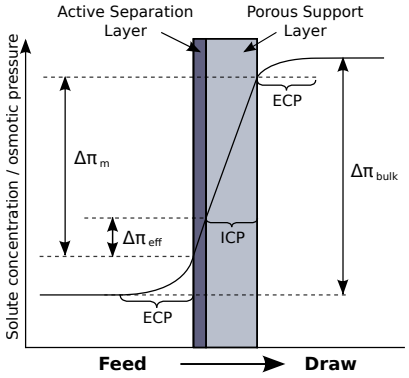


Figure 2.2: *Figure showing the concept of CP on a membrane in the AL-FS orientation. As solvent permeates the asymmetric membrane, ECP and ICP become pronounced outside and within the membrane respectively. $\Delta\pi_{bulk}$ is the bulk osmotic pressure difference between the feed and the draw, $\Delta\pi_m$ is the pressure difference across the membrane and $\Delta\pi_{eff}$ is the effective osmotic pressure difference that drives the flow of water.*

As the figure illustrates, there are two types of CP; namely *internal concentration polarization* (ICP) and *external concentration polarization* (ECP), both of which can be dilutive or concentrative depending on the membrane orientation. ICP, in this case dilutive, occurs due to the mixing of pure water permeating the membrane with the concentrated draw solution present in the porous support layer. ECP on the draw side occurs due convection of diluted draw solution out of the porous support and diffusion of salt back into the porous support, while it on the feed side occurs as a consequence of the removal of water from the weakly concentrated feed solution. CP has a diminishing effect on the water flux through the membrane, because it effectively reduced the osmotic pressure gradient that drives the flow [3]. In practice, ECP can be dealt with by bettering the flow conditions around the membrane, unlike ICP which occurs inside the membrane and is thus largely unaffected by the flow conditions. Reducing the effect of ICP is hence a matter of narrowing the porous support layer, which in turn makes the membrane more fragile.

2.1.3 Model for Water & Salt Flux in Forward Osmosis

Water and salt fluxes for any osmotic separation process are in principle governed by the following equations [10]:

$$\mathbf{J}_w = A|\Delta p_{eff} - \Delta\pi_{eff}|\hat{\mathbf{n}}_w \qquad (2.2) \qquad\qquad \mathbf{J}_s = -B|\Delta c_{eff}|\hat{\mathbf{n}}_w \qquad (2.3)$$

where A and B are the pure water permeability and the solute permeation coefficient of the membrane, Δp_{eff}, $\Delta\pi_{eff}$ and Δc_{eff} are differences in hydraulic pressure, osmotic pressure and concentration, respectively, across the membrane active layer, and $\hat{\mathbf{n}}_w$ is a normal vector pointing in the direction of the water flow. For an RO process Eqs. (2.2) and (2.3) accurately describe the fluxes through the membrane, however for FO, where $\Delta p_{eff} \approx 0$, the effects of CP become pronounced to a degree that needs to be accounted for in a model. Several analytical models describing water and solvent transport in FO filtration has been published [4–6], however, to my knowledge, none that fully account for CP effects in all domains of the feed-membrane-draw system. A revised version of the model derived by A. Tiraferri et. al. [5] is thus proposed for a membrane in the AL-FS orientation:

$$\mathbf{J}_w = A\,\frac{\pi_D \exp\left(-J_w\left[\frac{1}{k_m}+\frac{1}{k_f}\right]\right) - \pi_F \exp\left(\frac{J_w}{k_f}\right)}{1+\frac{B}{J_w}\left[\exp\left(\frac{J_w}{k_f}\right)-\exp\left(-\frac{J_w}{k_m}\right)\right]}\,\hat{\mathbf{n}}_w \qquad (2.4)$$

$$\mathbf{J}_s = -B\,\frac{c_D \exp\left(-J_w\left[\frac{1}{k_m}+\frac{1}{k_f}\right]\right) - c_F \exp\left(\frac{J_w}{k_f}\right)}{1+\frac{B}{J_w}\left[\exp\left(\frac{J_w}{k_f}\right)-\exp\left(-\frac{J_w}{k_m}\right)\right]}\,\hat{\mathbf{n}}_w \qquad (2.5)$$

where k are mass transfer coefficients (also inverse solute resistivities) and their subscripts denote the regions in which they are present: inside the membrane porous layer m, in the flow f, in the bulk of the draw D and the bulk of the feed F. Analytical approximations of k_m and k_f are given [4, 11]:

$$k_m = \frac{D_s}{S} \qquad (2.6) \qquad\qquad k_f = 1.62\left(\frac{QD_s^2}{2Wh^2L}\right)^{1/3} \qquad (2.7)$$

where D_s is the diffusion coefficient of the solute in the support layer and S is the structure parameter of the membrane support layer, Q is the flow rate and W, H and L are the width, height and length dimensions of a simple rectangular box approximating the flow channel. It is noted that the only change made to the model proposed by Tiraferri is the multiplication of a dilute ECP modulus $\exp\left(-\frac{J_w}{k_f}\right)$ with the draw osmotic pressure π_D and concentration c_D in (2.4) and (2.5), because it is assumed that the equations should account for ECP on

especially the draw side, at low flow rates. The reader is referred to Appendix A for a derivation of Eqs. (2.4) and (2.5).

For simulating flow of FO filtration, the model requires the membrane parameters A, B and k_m^{-1} as arguments. Determination of these is done experimentally by measuring J_w and J_s for different draw and feed concentrations, c_D and c_F, and solving the flux equations (2.4) and (2.5) by the numerical method of least squares. A MATLAB® script with graphic user interface (GUI) was developed for this very purpose by Tiraferri, making the parameter determination very straight forward [5]. In this work, changes to the script were made in accordance with Eqs. (2.4) and (2.5).

2.2 Computational Fluid Dynamics & Simulation

In short, *Computational Fluid Dynamics* or CFD as it is commonly referred to, is a branch of fluid mechanics that uses numerical methods and algorithms to analyze fluid flows. Motivated by limitations in the analytical approach offered by traditional fluid dynamics, CFD has proven to be an extremely powerful discipline allowing researchers to acquire detailed information about mass transport in and around complex geometries by means of simulation. It offers a way of gaining insight about system behavior that is expensive or even impossible to physically monitor, and it is applied in a broad range of industrial processes, from aerospace design optimization to simulation of blood flow in arteries and veins. Its earliest uses were reported around the advent of the first digital computer in the 50ies, and has matured in proportion with the increasing availability of computing power ever since[12].

CFD has emerged as a wide ranging discipline, and it is the purpose of this section to offer the reader an opportunity to understand the CFD that is used in this work. First the basic concepts and methodology of CFD is introduced. The mathematics that govern the flow of fluids in the case of this study is then presented. An introduction to the open source simulation software OpenFOAM® is given, and it is described how it can be used to carry out CFD simulations. This also includes a subsection describing the specific model that was used in this work. Finally I discuss some important sources of error and how they are best avoided.

2.2.1 Basics & Methodology

In a nutshell, the broad strategy of CFD is to take a continuous domain representing a known problem and replace it with a discrete domain to create a finite amount of grid locations. It is then possible to solve the equations that govern fluid flow throughout the whole domain, only at selected points, which can be interpolated to give values at any location. Fig. 2.3 sketches this general idea for a one-dimensional (1D) case.

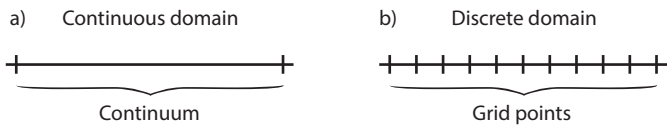


Figure 2.3: *Illustration of the difference between* a) *a continuous domain and* b) *a discrete domain for a 1D case.*

Discretization yields a huge set of algebraic equations that is solved repetitively in a computer, until a steady state is reached. The resulting data can then

be analyzed in different ways in order to gain insight about the simulated phenomenon. This general approach can be applied to any given problem involving fluid flow, thus making it an extremely powerful tool e.g. for analysis of physically inaccessible systems.

For any given case there are three main steps to a simulation process: pre-processing, solving and post-processing. These are described in the following.

Pre-processing In this initial step the case is defined such that it can be processed by a solver. In outline the typical procedure is:

1. Defining the surface of the problem domain.
2. Generating a grid/mesh of cells within the volume of the geometry.
3. Setting fluid and flow parameters.
4. Specifying appropriate boundary conditions (BC) for the different surfaces of the domain.

The quality of a flow problem solution is governed by the quality of the mesh. In general the finer the grid is, the better the solution. It is, however, not wise to make the grid equally fine at all locations as this would demand excessive computing power to solve. The optimal mesh is therefore designed such that it is sufficiently fine in places where detailed flow information is desired, and coarse elsewhere. It is noted that, at times, pre-processing can be the most time consuming step of the three, mainly due to difficulties with mesh generation.

Solving There is a variety of solution techniques in CFD each approaching problem solving differently. In this work a solver developed by M. F. Gruber et. al. utilizing the *finite volume method* (FVM) was used [8]. The numerical algorithm of the FVM takes the following approach:

1. Formal integration of the equations governing fluid flow over all of the control volumes (CV) in the mesh.
2. Discretization of the governing equations. This converts the equations into algebraic equations that can be put into system.
3. Solution of the system of algebraic equations by iterative method.

With an appropriate solver at hand this step demands little attention from the user. However, execution times can become very long depending on the fineness of the mesh. The solver used in this work was specifically developed for flow in FO membrane modules. It makes use of a so-called PISO-loop algorithm, which is explained in Subsec. 2.2.3.

Post-processing In this final step, the data from the solving step is visualized graphically for analysis and presentation purposes. In OpenFOAM, the program ParaView is the common tool for post-processing which is described in further detail in Subsec. 3.2.3.

2.2.2 Governing Equations

There are three fundamental concepts in all fluid dynamics, namely the concepts of *mass*, *momentum* and *energy* conservation. Each of these principles inherently lead to equations that govern the flow of fluids, and in CFD this is the same. However, to describe the transport of solute due to diffusion and convection a fourth governing equation is needed. Together these equations form a solid theoretical foundation that can be applied to solve practically any flow problem. In CFD problem solving, an important factor is solution time, and for a given case the governing equations can be readily simplified using appropriate assumptions. The model applied in this work was developed using a *weakly compressible* formulation of the governing equations, meaning that density ρ, viscosity μ and solute diffusion D were assumed to be functions of solute mass fraction m_A. In the following I will thus present the necessary governing equations in their relevant forms, such that the reader can acquire an understanding of the foundation for the model used in the simulations. The energy equation derived from the concept of energy conservation is not presented, since it is assumed in all simulations that the flow is incompressible and isothermal, i.e. that no temperature gradients are present in the fluid. Note that it is outside the scope of this thesis to provide thorough mathematical insight through derivation; for this the reader is referred to the work of J. D. Anderson[13].

The Continuity Equation By applying the principle of mass conservation to a fluid element one can derive the continuity equation. Taking as an example a constant finite CV, conservation of mass dictates that the net mass flow out of the CV must be equal to the time-rate of mass decrease inside the CV. The continuity equation is thus obtained as[13]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{2.8}$$

where ρ is fluid density, t is time and $\mathbf{U}$ is a flow velocity vector at a given location. In the weakly compressible formulation $\partial\rho/\partial t$ is not eliminated, as it would have been in the more common incompressible formulation, because ρ is a function of m_A which may vary in time. Therefore, Eq. (2.8) is not simplified for used in the model.

The Momentum Equation The momentum equation, commonly referred to as the *Navier-Stokes equation*, is derived by applying Newton's second law to a fluid CV. It is known to have many forms, depending on the application. For a weakly compressible formulation the sum of all surface stresses and body forces on the CV, for each direction in an orthogonal coordinate system are equated with the CV density times its respective acceleration, expressed as a substantial derivative of velocity [13]. It is then assumed that gravity is the only body force acting on the fluid and that viscosity is not constant throughout, which gathered in vector form yields the equation [14]:

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) = -\nabla p + \nabla \cdot \left[\mu \left(\nabla \mathbf{U} + (\nabla \mathbf{U})^{\mathrm{T}}\right)\right] + \rho \mathbf{g} \tag{2.9}$$

where p is fluid pressure and $\mathbf{g}$ is the gravity vector. By solving the Navier-Stokes equation, an expression for the velocity field within a domain can be obtained. Note that in much literature the full set of governing equations are commonly referred to as *Navier-Stokes equations*, however, in this thesis I will only use this term for the momentum equation.

The Diffusion-Convection Equation The continuity equation and the Navier Stokes equation only govern fluid flow. They do not account for mass transport of solute throughout the problem domain, and therefore it is necessary to introduce the *diffusion-convection equation*, which quite obviously describes the transport of solute due to two processes: diffusion and convection. The equation states that the change in concentration of a specie in a given CV equals the net flow into or out of the CV and the generation or consumption of specie within, e.g. due to chemical reaction. It is assumed that all flux originates from the difference between diffusive and convective terms, and furthermore, in the current formulation, it is assumed that there is no consumption or generation of specie. This yields the form [8]:

$$\frac{\partial \rho m_A}{\partial t} + \nabla \cdot (\rho \mathrm{U} m_A) - \nabla \cdot (\rho D_{\mathrm{AB}} \nabla m_A) = 0 \tag{2.10}$$

where concentration is expressed as the product of density ρ and solute mass fraction m_A and the subscript AB with the diffusion coefficient refers to the fact that it is only valid for two-constituent-solutions, e.g. saltwater. The second term of Eq. (2.10) is the convective flux and the third term is the diffusive flux given by Fick's law, which states that flux equals the negative concentration gradient with the diffusion coefficient as a proportionality constant [9].

2.2.3 Simulation in OpenFOAM

CFD simulations can be performed using a variety of software packages. Most of them are, however, only commercially available and hard coded to apply to a fixed range of problems. In this work the software package OpenFOAM® was used. OpenFOAM, short for *Open Field Operation and Manipulation*, is an open source C++ toolbox of pre-written solvers and utilities in a framework that is completely open, readily extensible and can be customized as needed. It is therefore a very powerful platform for performing any kind of numerical simulation in continuum mechanics. Since the OpenFOAM default library provides no model capable of including mass transport of solute in the flow, a model developed by Gruber et. al. was used in this work[8]. In the following I will provide a basic introduction to OpenFOAM as it was used in this work. For descriptions on how to use OpenFOAM, the reader is referred to the official user guide provided online[15], and the *Getting Started* guide, written by the author, provided in Appendix C.

OpenFOAM Basics The first thing new users should know about OpenFOAM is that it comes stripped of any graphic user interface whatsoever and users will thus have to get acquainted with the use of `Terminal` on either a Mac or Linux machine (workaround methods also allow Windows operation). Figure 2.4.a shows the general structure of OpenFOAM. The library is primarily used to create *applications* which are executables that fall into two categories: *utilities* that perform pre- and post-processing tasks, involving data manipulation and algebraic calculations, and *solvers* that can be executed on fully pre-processed cases to solve specific problems, i.e. run simulations.

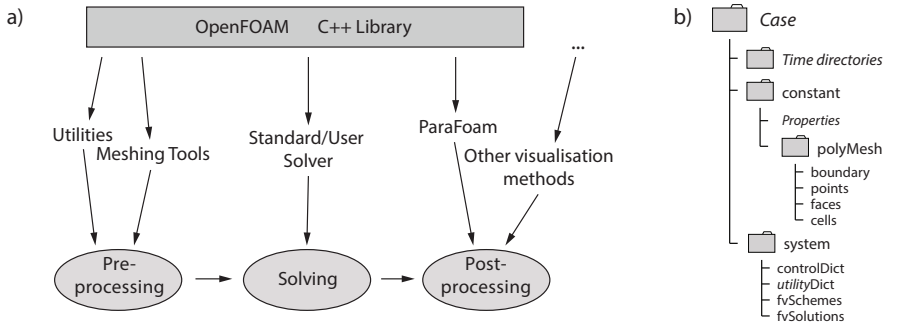


Figure 2.4: a) *An overview of the OpenFOAM structure.* b) *Example of a case directory. Italic font in the figure indicates type/class of folders of files, and not file name.*

When working with a specific case in OpenFOAM the user will typically navigate in a systematic network of folders containing the files that define the problem at

hand, such as it is illustrated in Fig. 2.4.b. The necessary files for a case to run is shown in the figure. The `system`-directory must contain a `controlDict` which is a dictionary of time-specific user defined information such as end time, time step size, write interval, etc. Furthermore it must contain the two files `fvSchemes` and `fvSolutions` which specify the mathematical methods used by the solver. In the `constant`-directory physical properties such as gravitation or, as in this work, fluid and membrane transport properties are specified. The `constant/polyMesh`-directory contains geometric information about the mesh in a `points` file that specify point locations in the mesh, and a `faces` file that use the points to define the faces of the mesh. Furthermore, in OpenFOAM, since each face is shared by two CVs, one of them is denoted "owner" and the other "neighbour", with a vector pointing from the face of the neighbour into the volume of the owner. The two files `neighbour` and `owner` are therefore also included in the `polyMesh` directory to account for this. All mesh data is specified in array form such that specific points, faces and cells have array numbers. By face number referencing, the `boundary` file defines the problem domain boundaries. Finally it is necessary to specify a `0` time directory, which contains initial conditions at boundaries of the problem domain. For the model used, it was required to have files `U`, `p` and `m_A` specifying the initial conditions for velocity, pressure and solute mass fraction, respectively. However, in general this folder will include any variable that changes with time.
In overview, simulation in OpenFOAM can be performed in the following steps:

1. Setting up the case directory with necessary folders and files. A common way to do this is by copying and altering files from similar tutorial cases given in the OpenFOAM library.
2. Creating a mesh using the `blockMesh`, `snappyHexMesh` or other relevant utilities.
3. Executing utilities on the case, e.g. to create specific boundaries, set initial fields, refine the mesh in certain regions, etc.
4. Decomposing the case into separate domains using the `decomposePar` utility such that a solver can be run in parallel on multiple processors.
5. **Executing the solver**.
6. Reconstructing the decomposed case using the `reconstructPar` utility.
7. Visualizing the results by executing `paraFoam` which launches the third-party visualization software ParaView and loads the time directories.

Note that step 4 and 6 are not necessary for the simulation to run, they just offer a neat way of shortening solution time - at the cost of computational power. Using decomposition methods to perform tasks faster can also be applied in step 2 when creating the mesh and in step 3 for faster execution of certain utilities.

FO Membrane Model The OpenFOAM standard library contains no solvers or BCs appropriate for simulating fluid flow in an FO membrane module, and thus a model developed by Gruber et. al. was used in this work[8]. The model was developed for simulation of both FO and RO operation and for FO it is referred to as the *OF-FO model*, where *OF* refers to the fact that it was implemented with OpenFOAM. Reversely the model for the RO setting is known as the *OF-RO model*, but since it is not relevant for this work it will not be discussed here.

The OF-FO model is composed of a solver and a membrane BC that are implemented separately in OpenFOAM. The solver, which is referred to as the `pisoSaltTransport` solver, uses the weakly compressible formulations of the governing equations discussed in Subsec. 2.2.2 to solve for U, p and m_A in a correction loop using the *PISO algorithm*[16]. In Fig. 2.5 the basic idea of the PISO algorithm is illustrated and explained.

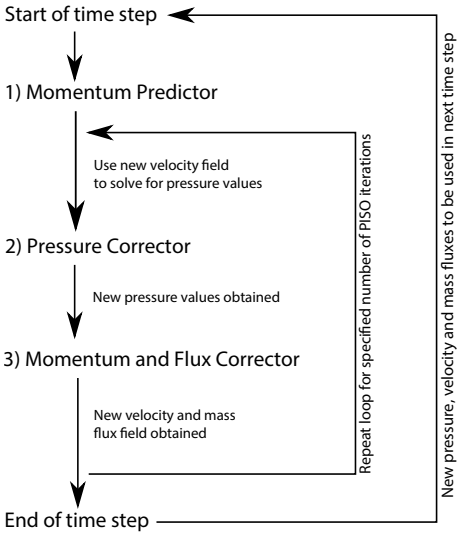


Figure 2.5: *A flow chart illustrating the PISO algorithm.* 1) *The Navier-Stokes equation is solved using an initial guess for pressure or a pressure distribution from the previous time step, to give a prediction of the velocity field.* 2) *The explicit pressure equation, derived from the momentum equation[16], uses the predicted velocity to produce a corresponding pressure field.* 3) *With the new pressure field both the velocity field and mass fluxes can be calculated using the momentum equation. Step* 2) *and* 3) *are repeated for a number of PISO corrector iterations, before pressure, velocity and mass fluxes are used in the next time step.*

The PISO-algorithm represents an efficient method for solving the governing equations for fluid flow in cases where it is necessary to resolve the pressure-velocity coupling through time.

The membrane BC for FO is implemented in OpenFOAM based on forms of the water and salt flux model equations (2.4) and (2.5) that neglect ECP and only account for ICP effects. It is applied to both sides of the membrane which is modeled as a 2D plane (no thickness). The membrane is discretized into segments that have two faces; one for each CV on either side of it. In OpenFOAM such an internal boundary face is known as a `baffle`, and because

the BC requires information from both sides of the membrane, the baffle faces are paired using a looping algorithm [17].

In this model a no-slip BC is generally assumed. This may however not accurately describe the conditions on the surface of the potentially rough porous support. The tangential slip velocity vector $\mathbf{U}_{\text{slip}}$ on the membrane is proportional to the shear rate $\partial\mathbf{U}/\partial n$, and can be expressed as [18]:

$$\mathbf{U}_{\text{slip}} = -\frac{\sqrt{\kappa}}{\alpha}\frac{\partial \mathbf{U}}{\partial n} \tag{2.11}$$

where α is the slip coefficient, κ is the permeability and n is the surface normal direction on the membrane. The model allows the user to specify the $\sqrt{\kappa}/\alpha$ proportionality constant.

2.2.4 Accuracy and Reliability

When validating numerical results by comparison to experimental results, it is important to be aware of common error sources in CFD. Below are listed two of the most important ones and how they are avoided.

- **Insufficient temporal resolution** When a time step becomes too large the solution will diverge and the simulation has to stop. To avoid this from happening in the case of this study, the time step must obey the Courant-Friedrichs-Lewy condition which, for an 1D case, is defined as [13]

$$\frac{u\Delta t}{\Delta x} < 1 \tag{2.12}$$

  where u is the velocity in the x-direction, Δt is the size of the time step and Δx is the grid size. What this equation basically states is that for one time unit, a particle in the velocity field must not displace itself further than the length of a grid length unit.

- **Insufficient spatial resolution** An important feature of the OF-FO model is its capability to capture CP effects in the vicinity of the membrane. In order to utilize this feature properly the spatial resolution in the areas where CP occurs must therefore have a certain level of fineness, such that the obtained results are *grid independent*. The Grid Convergence Index (GCI) estimates the percentage-wise error associated with one grid when compared to another of equal shape but different fineness, and as such it can be determined whether grid independence is achieved for a given discretized problem domain. The GCI in 3D for a coarse grid

compared to a finer grid can be found by [14]:

$$\mathrm{GCI}_{\mathrm{coarse}} = \frac{3|e|\mathbb{R}^3}{\mathbb{R}^3 - 1} \tag{2.13}$$

Reversely the GCI for a fine grid compared to a coarser grid is:

$$\mathrm{GCI}_{\mathrm{fine}} = \frac{3|e|}{\mathbb{R}^3 - 1} \tag{2.14}$$

where e is the relative error between integral functions (such as J_w) of the two grids, and $\mathbb{R}$ is the ratio of cells in the fine and the coarse grid. Validation is carried out by requiring that the GCI is below a certain value, typically 1%.

3 Simulation & Experimental Procedure

The task at hand was to investigate fluid flow under various conditions in the Sterlitech CF042 membrane module in FO operation, by means of 3D simulations in the open source CFD software OpenFOAM. Simulations were carried out using the *OF-FO model* implemented in OpenFOAM by Gruber et. al.[8], specifically developed to simulate flow in FO membrane modules. Four series of simulations were carries out for different flow rates, draw concentrations, membrane slip parameters and spacer densities. Membrane specific parameters of the Aquaporin Inside$^{\text{TM}}$ thin membrane (AQP Thin) was used in all series, while a single simulation for the thicker Aquaporin Inside membrane (AQP Thick) under standard conditions was performed. All simulations were carried out using the DTU High Performance Computing (HPC) clusters.

The purpose of this chapter is to document the work performed in this project, with a precision that allows future students of CFD to continue my work. It is written based on the assumption that the reader either has prior knowledge about CFD and `Terminal` usage, or is motivated to investigate unknown terms and concepts. The first section describes how membrane parameters were obtained through experiments and computations. The second section presents a generic working process for simulation of FO filtration in the Sterlitech CF042 membrane module, which can be applied for simulation of flow in any given complex membrane module geometry. A final section is presented, describing briefly the process of checking the solution for grid independence (see Subsec. 2.2.4). An overview of how to get started using OpenFOAM and the OF-FO model is furthermore offered in Appendix C. It is my hope that readers with the ambition to perform CFD simulations of FO filtration in membrane modules, will find valuable guidance in the chapter. Some parts may thus seem excessively elaborate, however, they are only presented so for the sake of clarity.

3.1 Obtaining Membrane Parameters

As mentioned in Subsec. 2.1.3, simulation of flow using the OF-FO model demands input specification of the three membrane parameters A, B, and k_m^{-1}. These parameters must be determined experimentally, otherwise the simulations will not reflect reality and their results are rendered meaningless. There are various methods that can be applied in order to determine these parameters and in this work the methodology proposed by A. Tiraferri et. al. has been used[5]. Tiraferri et. al. provides in their article a MATLAB script with GUI, which is used with the modifications to the flux equations (2.4) and (2.5) described in Subsec. 2.1.3. It should be noted that the modifications were made because

the equations were initially derived for much higher cross flow velocities such that ECP on the draw side was assumed negligible compared to dilutive ICP within the porous support layer. In this work, relatively slow cross flows were used such that ECP on the draw side would become pronounced in comparison to ICP, thus motivating the alterations made, as accounted for in Appendix A.

Obtaining membrane parameters for a membrane sample was fairly simple, as the methodology demands only a single FO experiment to be carried out, in order to obtain the necessary data. A membrane sample of the type in question was mounted into a membrane module which was connected to a feed and a draw reservoir through a peristaltic pump. The pump was switched on, and fluxes were monitored as the draw solute concentration was increased in four stages, simply by addition of solute to the draw reservoir. Each stage took up to an hour due to the transient nature of the solute concentration in the beginning of each stage. The approach is conceptually illustrated in Fig. 3.1.

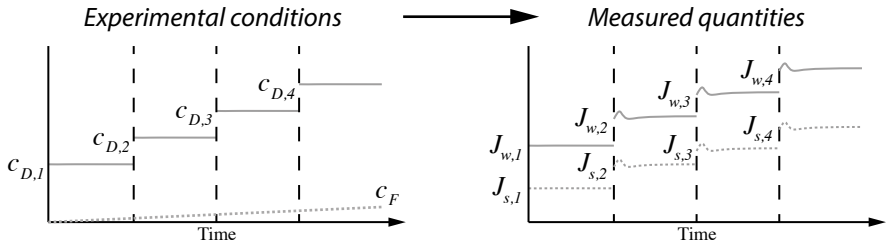


Figure 3.1: *A conceptual illustration of the experiment performed for determination of A, B and k_m values. The dashed vertical lines illustrate the transition between stages of the experiment, i.e. the addition of solute to the draw reservoir.*

For each stage this approach yields four data points: draw concentration c_D, feed concentration c_F, water flux J_w and reverse salt flux J_s. This data was simply loaded into the modified MATLAB script using the GUI, which demands temperature, solute diffusion coefficient, and initial guesses for the membrane parameters as arguments. The script automatically solved the flux equations (2.4) and (2.5) by the numerical method of least squares, and returned estimates of A, B and S (recalling Eq. (2.6)). This is illustrated in Fig. 3.2.

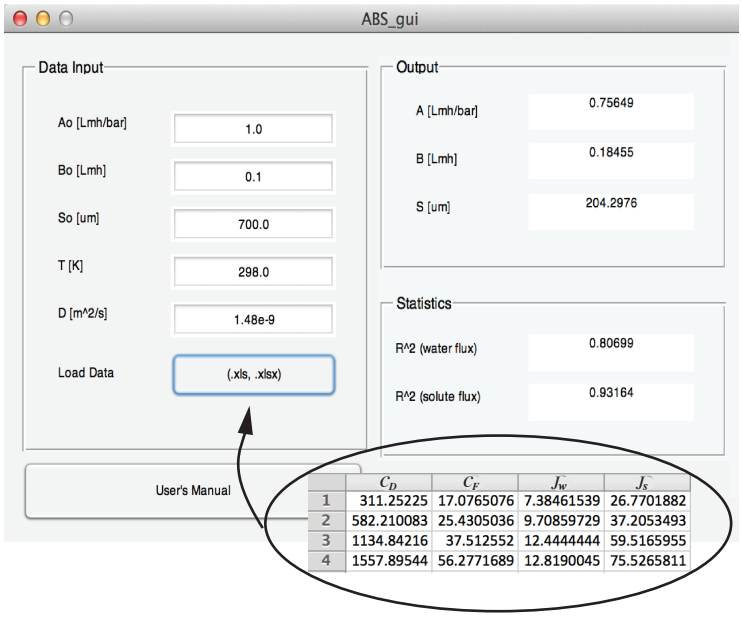


Figure 3.2: *Illustration of how the MATLAB script GUI was used. Starting MATLAB and navigating to the folder where the script files were located, the GUI was launched with the command* `ABS_gui` *executed from the main command window. The experimental data was stored, as illustrated, in Microsoft Excel® format, and was loaded by clicking the 'Load Data' button. The script then automatically returned values for A, B and S.*

Ideally, this is done for several samples from the same membrane in order to reduce uncertainty. In the case of this study three samples of the AQP Thick membrane and two samples of the AQP Thin membrane were tested. This was not a satisfying number of tests and due to this, some degree of uncertainty was introduced in the results obtained from simulation.

It should be stated that this author played no part in gathering experimental data, and that his participation here was purely that of deriving modified expressions for water and salt fluxes such that the script could be appropriately modified.

3.2 Performing Simulations

In the following section all the steps performed in carrying out a simulation for FO filtration in the Sterlitech CF042 membrane module are described. Membrane parameters of the AQP Thin membrane was used. The characteristic parameters of the case are given in Table 3.1. The method used for performing this particular simulation was general to the work performed in this project. In general, this simulation was carried out following the theory described in Sec. 2.2. This whole section is therefore structured in accordance with the method-

ology presented in Subsec. 2.2.1. I will start from the very bottom, describing geometry definition using the freely available 3D computer graphics software Blender® [19], and how the Blender addon SwiftSnap was used as a tool to prepare the case for meshing with the `snappyHexMesh` utility. I will then explain how the mesh was generated, manipulated and prepared for solving. Finally, I will describe the process of solving, followed by a brief description of how data was collected and visualized.

Flow param.	Value	**Mem. param.**	Value
Δc [M]	1	A [m(sPa)$^{-1}$]	2.23×10^{-12}
U_{flow} [mL/min]	50	B [ms^{-1}]	5.39×10^{-8}
U_{slip} [m/s]	0	k_m^{-1} [sm^{-1}]	1.39×10^{5}

Table 3.1: *Fluid flow and membrane parameters for the case. Δc is the difference in solute concentration between the draw and feed side of the membrane. U_{flow} is the flow velocity of the fluid in the module, and U_{slip} is the slip velocity on the membrane. A, B, and k_m^{-1} characterize the AQP Thin membrane and were obtained as discussed in Sec. 3.1.*

3.2.1 Pre-processing

In this simulation, as with all other simulations performed in this work, pre-processing was by far the most time consuming and difficult of the three main steps, generally due to errors in the meshing process described in further detail in the following. The overall work flow was that of 1) designing part of the module in Blender and 2) exporting it to a format accepted by OpenFOAM. 3) Generating the mesh using the `snappyHexMesh` utility and checking for meshing errors. 4) Mirroring the meshed part of the module to the full geometry, and defining the membrane surface using the `createBaffles` utility and 5) setting fluid and membrane parameters for the case. How this was done is explained in the following.

1 Module Design in Blender Blender is a free and open source 3D computer graphics software, which can be used for creating everything from 3D printing models to video games. Blender comes with an enormous amount of functionality, and therefore the following includes only a crude rendering of how the software was used to define the surface geometry of the module draw chamber. The reader is referred to the user guide written by I. Løvaas for a very elaborate description on how a membrane module is designed in Blender. The guide is provided online in Appendix D. The end product of this step is illustrated as rendered images in Fig. 3.3. Only half of the module is designed in Blender because it is symmetric and can therefore be mirrored in OpenFOAM using the `mirrorMesh` utility.

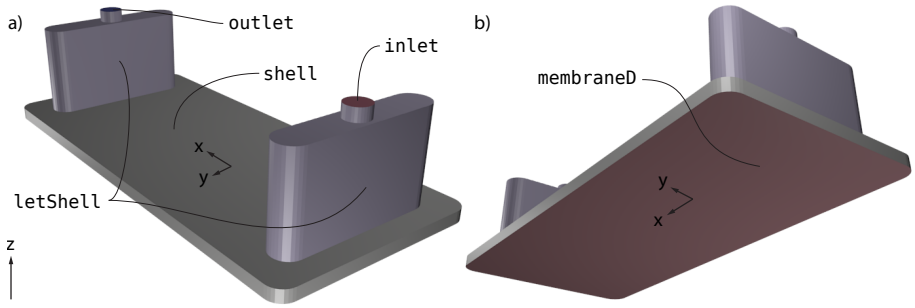


Figure 3.3: *Rendered versions of the membrane module draw chamber as it was designed in Blender,* a) *seen from the top and* b) *seen from below. Each color on the surface of the design represent a different surface patch, the names of which are given on the figure.*

First, I specified the problem domain. For a membrane module, what needs to be defined in a computer model is *the space that confines the fluid.* To do this, I started by simply taking a caliper and measuring out every dimension of the physical membrane module draw chamber. These measurements are given in Appendix B. The overall strategy of "building" the module draw chamber, was to construct half of it, and then mirror it to the full surface geometry, such that fewer design steps would have to be taken, and that symmetry was ensured. The design shown in Fig. 3.3 was obtained through the following steps:

1. Blender was launched and the work-space was cleared (all default objects deleted).
2. A plane was created and given the dimensions of the membrane surface cut in half about the y-axis. The center of the plane was moved to the origin and then translated in the negative x-direction by an amount that exactly aligned one of its sides with the y-axis.
3. The two farthest corners from the y-axis were rounded off, by insertion of circles of the measured radius.
4. The surface was extruded in the z-direction defining half the `shell` surface.
5. The outline of the `letShell` surface was then defined. This again was done by insertion of circles to define the rounding. The surface within the outline to be extruded was removed, such that there would be no wall separating the volumes within the `shell` and `letShell` surfaces.
6. The `letShell` outline was extruded in the z-direction by the measured amount.
7. The `inlet` outline was created as a circle centered in the empty top plane of the `letShell` surface. The surface was then closed around the `inlet` outline which was left open.

8. The `inlet` outline was extruded by the measured amount and closed.
9. Finally, the entire geometry was mirrored with respects to the origin, defining the full draw chamber.

Inspection for unwanted surfaces and double points was necessary in order to obtain good results in the meshing stage. Since the OpenFOAM meshing utility `snappyHexMesh` is very sensitive to irregularities in the surface files, it proved best to take the simplest possible design approach.

2 Exporting with SwiftSnap In order for the `snappyHexMesh` utility to define a mesh, the following must be included in the case directory: a `blockMeshDict` defining the bounding box of the domain, a `controlDict`, a `snappyHexMeshDict` and a `triSurface` folder containing the surface files in either `.stl` or `.obj` format. Fig. 3.4 gives an overview the "mesh-ready" folder system, for this particular case.

The Blender addon SwiftSnap was used in the process of preparing the case for meshing [20]. SwiftSnap is a powerful tool that enables the user to write the case with the files distributes to their right locations, such that adding a `controlDict` to the system directory, is the only further input needed. The user can mark surfaces as individual patches with specific *refinement levels*, and assign the whole geometry with a resolution in terms of the cell side length of a cell with refinement level 0. Refinement level 1 then corresponds to cubic cell side lengths of half the given resolution, level 2 is 1/4, level 3 is 1/8, etc. It can also be used to mark features (edges, corners) with refinement levels. Important to this work, SwiftSnap also allows the user to define a region inside which the geometry volume is meshed to a certain refinement level. Many other features exist which is also described in Løvaas' user guide (Appendix D).

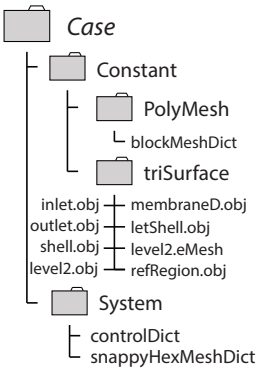


Figure 3.4: *The "mesh-ready" folder system of the case, in overview.*

Feature	Level
`inlet`	3
`outlet`	3
`letShell`	2
`shell`	1
`membraneD`	0
90° edges	2
Ref. region	4

Table 3.2: *Table showing the refinement levels of different features of the surface geometry.*

To prepare the case for meshing, SwiftSnap was installed and enabled for the surface geometry object. Patches of the surface was marked in accordance with Fig. 3.3, and the resolution was set to `0.0020`, i.e. cell side length in level 0 region of 2 mm. A refinement region of level 4 was defined to encapsulate the volume close to the `membraneD` patch. All 90° angled edges

were marked as level 2. Table 3.2 gives an overview of the refinement levels set for the case. Using the addon's `write` function and setting the destination to `case/system`, the surface files in `.obj` format, a `snappyHexMeshDict` and a `blockMeshDict` were written and distributed as illustrated in Fig. 3.4. A `controlDict` was additionally imported to the `system` folder. Changes were made to the `snappyHexMeshDict` to obtain a better mesh, as can be seen from the version of the `snappyHexMeshDict` included in the case example provided in the online material (Appendix D).

3 Generating the Mesh Comparing the case directory of Fig. 3.4 with that of Fig. 2.4, it is apparent that the one in question contains no information about the mesh of the problem domain, since the geometry-defining `polyMesh` folder contains only a dictionary for the `blockMesh` utility. The case does however contain all the necessary geometric information needed to generate the mesh. This is done by way of "computational sculpting", using the surface files exported from Blender as the "template". The `snappyHexMesh` utility is a powerful tool for mesh generation that iteratively "sculpts" the mesh. It starts with a simple block mesh which is reduced to fit the outline of the surface geometry. It then "snaps" the fit; in other words, smooths the fit surface until it resembles that of the geometry to a certain level of tolerance, yielding a meshed rendering of the surface geometry [15]. The block mesh was decomposed into 16 domains ($4 \times 4 \times 1$) to reduce the meshing time. `snappyHexMesh` was run in parallel using a *Message Passing Interface* (MPI). All of this was carried out on the HPC. The workflow was designed as follows:

```
blockMesh && decomposePar && mpirun -np 16
snappyHexMesh -parallel && reconstructParMesh -mergeTol
1e-06 -latestTime && checkMesh -allGeometry &&
paraview
```

where **`&&`** is a command separating operator that only runs the next command if the previous one executed successfully (marked in bold only to promote that it's a separator). The `reconstructParMesh` utility reconstructs the decomposed mesh, given merge tolerance and the time step to reconstruct as arguments, `-mergeTol 1e-06` and `-latestTime` respectively. It creates a new time directory in the `case` directory which contains the geometric information of the generated mesh in a `polyMesh` folder. This folder was copied to the constant directory, overwriting the existing `polyMesh` directory, such as to update the case mesh. The `checkMesh -allGeometry` utility was executed for analytical mesh analysis. The utility prints a full report on the mesh, and points out critical errors such as overtly skewed or non-orthogonal cells, critically small cell

volumes, etc. In other words, it gives a quantitative first indication of the mesh quality. Furthermore the mesh was visually inspected in ParaView. A first view of the draw chamber after meshing is given in Fig. 3.5.a. Fig. 3.5.b shows a close look from the side, such as to illustrate the effect of adding a refinement region.

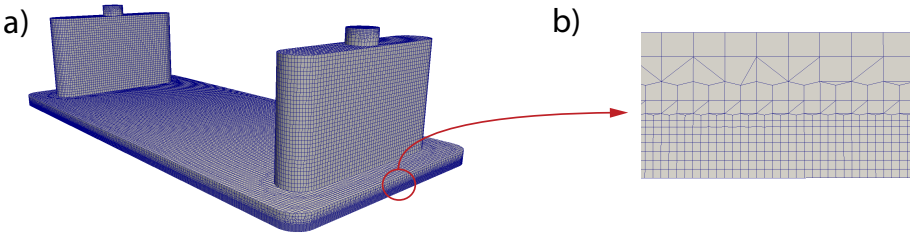


Figure 3.5: *Visualization of the mesh in ParaView.* a) *A first look of the module draw chamber after meshing with the* `snappyHexMesh` *utility.* b) *A close look from the side which shows how the cells are much smaller close to the membrane than in the rest of the domain.*

It proved most important to inspect the `membraneD` patch for irregularities, as the subsequent mirroring step required it to be completely flat. The `inlet` and `outlet` patches were, however, also required to be flat, as grooves would introduce uncertainty in the surface areas of the patches, which were required to be known in order to specify flow velocities through the patches. Creating a decent mesh was a process of trial and error, where meshing was tried for variety of different parameter settings in the `snappyHexMeshDict` until a satisfying mesh was obtained.

4 Mirroring & Membrane Definition Having successfully meshed the module draw chamber, the full geometry could be generated simply by mirroring the draw side with respects to a point on the `membraneD` patch. The `mirrorMesh` utility was used, which required a `mirrorMeshDict` to be imported to the `system` directory, specifying the reflection point (`base point`) and a normal vector for the mirroring direction. The reflection point was specified as the origin and the normal vector, the negative z-direction. The mesh was therefore translated to a location where the `membraneD` patch would coincide with the origin, using the `transformPoints` utility with the argument `-translate "(x y z)"`, where `x`, `y` and `z` controlled the translation. `mirrorMesh` was then executed. This created a new time directory for the full mesh with the `membraneD` patch removed, and the amount of faces for all other patches doubled, most importantly the `inlet` and `outlet` patches. As in Step *3) Generating the Mesh* the `polyMesh` folder of the new time directory was copied to the `constant` directory, overwriting the current `polyMesh` folder to

update the case mesh. The `boundary` file in the `constant/polyMesh` directory was then manually edited, such that the duplicated `inlet` and `outlet` patches were split into four patches, namely `inletD`, `outletD`, `inletF` and `outletF`. This was necessary in order for independent boundary conditions to be specified for each opening of the chamber. The mesh was then inspected in ParaView, most importantly to ensure full removal of the `membraneD` patch. This was important since the membrane was to be redefined with new boundary conditions as just `membrane`, using the `createBaffles` utility.

Having successfully constructed the full module mesh by mirroring, the last mesh manipulation step was to define the membrane, as an internal boundary. This was done by first creating a `set`, i.e. a selection, of only the membrane boundary faces using the `topoSet` utility. It requires a `topoSetDict` in the `system` directory to specify the given `name` of the set (`f0`), the `type` (`faceSet`) and two points that define a box which encapsulates only the faces of the set (bottom corner and top opposite corner). Having set the values in the dictionary, a `createBafflesDict` was imported into the `system` directory, such that the behavior of the later executed `createBaffles` utility could be controlled. Recalling Subsec. 2.2.3, a `baffle` is an internal boundary that has a face defined for each CV that it confines. `createBaffles` allows the user to define such an internal boundary if the boundary faces are written to a `set` and converted to a face zone. The `createBafflesDict` requires specification of `type`, `set name` and the names of the two baffle sides, in this case `membrane` for both. To define the `membrane` patch the following commands were executed:

```
topoSet && setsToZones -noFlipMap && createBaffles
```

The `setsToZones` utility converts sets into zones that can be read by the `createBaffles` utility, and the `-noFlipMap` argument tells the utility to ignore the orientation of the set. Again, inspection in ParaView was necessary to ensure that the mesh was good. In Fig. 3.6 the mesh is presented as seen in inspection of a) the mesh after mirroring, and b) the `membrane` patch.

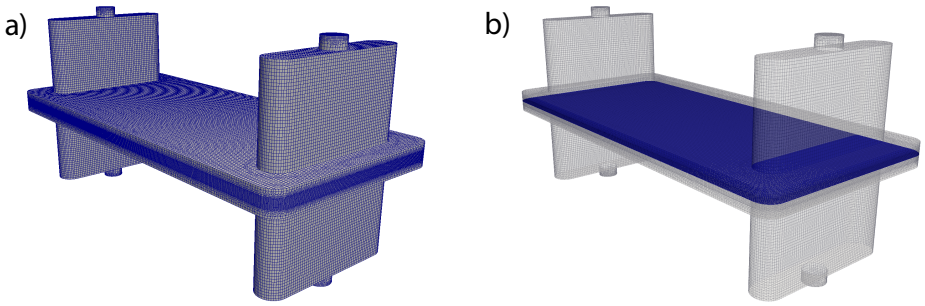


Figure 3.6: *The mesh as visualized upon inspection in ParaView.* a) *shows the mesh after mirroring, and* b) *gives a look of the* `membrane` *patch after it has been created with the* `createBaffles` *utility.*

Here it was very important that no other faces than those of the membrane boundary had been labeled as `membrane`. Such error would mean that a narrower selection box in the `topoSetDict` should have been defined, or worse that the mesh quality was unsatisfying. In the latter of these scenarios it would, as a minimum, have been required to go back one step and generate the mesh again.

5 Setting Fluid & Membrane Parameters

With a successfully generated mesh, the last step necessary before solving, was to include files in a `0` directory, defining the initial flow conditions, and in the `constant` directory defining parameters that characterize the membrane and the fluid. Files for solute mass fraction, pressure and velocity - `m_A`, `p` and `U` respectively - were added to the `0` directory, with the initial condition values specified in Table 3.1. The file `transportProperties` was added to the `constant` directory with the A, B and k_m^{-1} values obtained in the experiment described in Sec. 3.1, and parameters needed for calculating the solute dependent viscosity of the fluid. Furthermore the file `g` defining gravity was included in `constant` as the solver took gravity as an argument. Gravity was set to 0 as it was assumed to be negligible. The files `fvSchemes` and `fvSolutions` were added to the `system` directory to specify the mathematical methods to be applied by the solver. Finally the `setFields` utility was executed on the case in order to set the initial fields for m_A such that the simulation

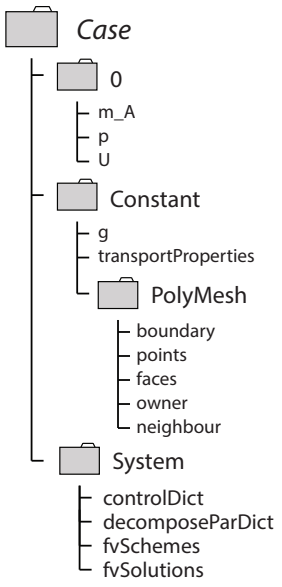


Figure 3.7: *The folder system of the fully pre processed case. Available in Appendix D.*

would reach steady state in less time. It required a `setFieldsDict` specifying bounding boxes and scalar field values for the domains where it was to operate.

An overview of the fully pre-processed case is given in Fig. 3.7. The case files are furthermore made available in Appendix D which provides a link for a website where they can be downloaded, and used freely upon reference to this thesis.

3.2.2 Solving

The process of solving required simply the execution of the solver on the case. Due to the very high resolution of the mesh, it was however necessary to use decomposition methods to reduce the calculation time. It is noted that decomposition of the mesh is a process that inherently belongs in the *pre-processing* stage, however, since in this work finding a good decomposition was a matter of trial-and-error, it is included here. The process of reconstructing the mesh after the simulation had reached steady state is also described in the following.

1 Decomposing the Mesh The term "good decomposition" refers to a decomposition of the mesh that will allow the solver to process the case fast and without crashing. Because errors in the meshing stage, described in Subsec. 3.2.1-3, inherently occur due to the high resolution of the mesh, "bad" cells will lead the solver to crash if one is located on a decomposition boundary. Using logic, it is apparent that for a higher number of decompositions, the solution is more likely to crash. Furthermore since the speed of parallel computing relies heavily on the communication speed between the processors that the case is distributed to, a high number of processors means more communication which slows down solving speed relative to the number of processors used.

Through trial-and-error it was concluded that a decomposition into 16 domains - 4 sections along the x-axis and 4 sections along the y-axis - was good for the purpose of this simulation. Sectioning along the z-axis proved to always lead the solver to crash, likely do the internal BCs on the `membrane` patch.

2 Executing `pisoSaltTransport` To solve the case when it is decomposed into 16 domains, the Open-MPI library version 1.6.4 was used. The `controlDict` was set to write a time directory every 50th simulation second and upon finalization. The following was executed to start the simulation:

```
mpirun -np 16 pisoSaltTransport -parallel
```

Steady state was declared after 315 simulation seconds (s_{sim}) when the water

flux would change less than $10^{-6}\%$ per s_{sim}. This corresponded to approximately 5 days and 6 hours of real-time solving. In the `fvSolutions` file, GAMG (Geometric-Algebraic Multi-Grid) was chosen as the solver algorithm.

3 Reconstructing the Mesh The final step before post-processing was reconstruction of the mesh after steady state had been declared. This required execution of the `reconstructPar -latestTime` command, with the `-latestTime` argument added such as to only reconstruct the latest written time directory.

3.2.3 Post-processing

In OpenFOAM simulation, the most common tool for post-processing is the program ParaView. It allows the user to visualize the geometry at each written time step in the simulation, while it contains a large amount of functionality that can be applied to collect specific data. In this work, data was collected by first using the `calculator` tool to calculate cell data (not interpolated values) from the mesh, and then then applying the `plot over line` tool.

Values of the J_w and J_s were read from the log file written by the solver. The fluxes were notices to have a very slow transient nature in every simulation, and therefore solving until an *actual* steady state was reached would have taken much longer than the 5 days and 6 hours given in the previous example. Since steady state was declared when the fluxes would change less than $10^{-6}\%$ per s_{sim}, the data was collected at its minimum value. Fig. 3.8 illustrates this with an exaggerated minimum, only to promote understanding.

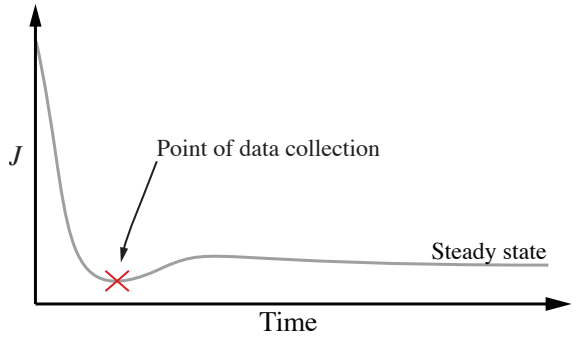


Figure 3.8: *The red cross on the plot illustrates the characteristic point at which data was collected from each simulation. It is obvious that collecting "true" steady state data would have required a significantly longer solving time. This figure is not to scale.*

This approach to data collection was taken for all simulations, and therefore the possible error associated with it is consistent throughout.

3.3 Determining Grid Independence

Due to the fact that the BCs, which the OF-FO model was based on, took no account of ECP phenomena, the accuracy of the results obtained from simulation relied heavily on the ability of the case to resolve ECP occurring close to the membrane. Investigations were made to determine what the mesh resolution should be in order to obtain grid independent results. A series of meshes with different resolutions were generated and compared using Eqs. (2.13) and (2.14). A maximum GCI threshold of 1% was set, and it was found that a mesh of resolution 2 mm yielded a GCI of 0.73 % when compared to a mesh of resolution 2.5 mm. This was satisfactory, and grid independence was declared for a mesh of 2 mm resolution.

4 Results & Discussion

In order to illustrate the impact of different varying parameters on the flow conditions in the module, the results obtained in the four series of simulations carried out, are presented and discussed in the following. Throughout the four series, membrane parameters from the AQP Thin membrane was used. A single simulation for the AQP Thick is also presented, solved for the standard flow parameters presented in the left part of Table 3.1. Unless otherwise stated, the flow parameters of the presented simulations are those of Table 3.1. A general overview of the simulation series performed is presented in Table 4.1.

Varied parameter	Symbol	Number of simulations	Range of variation
Draw concentration	c_D	5	0.3 – 2 [M]
Cross flow rate	Q	7	15 – 250 [mL min^{-1}]
Permeability & slip coeff.	$\sqrt{\kappa}/\alpha$	6	10^{-6} – 7.5×10^{-5} [m]
Spacer density	n_{spacers}	4	10 – 36 [-]

Table 4.1: *A schematic overview of the four series of simulations carried out in this work.*

4.1 Draw Concentration as a Variable

Flow conditions for different draw concentrations are of interest in FO research, because, while greater concentrations yield higher water fluxes, they are, if necessary, more difficult and expensive to post-process. It is therefore apparent that a certain cost-benefit balance must be achieved for good draw solutions. To show how J_w depends on c_D, Fig. 4.1.a illustrates this relationship, determined through simulation, experiments and mathematical analysis. In Fig. 4.1.b the concentration profiles are plotted across the membrane, such as to illustrate the occurrence of CP for different draw concentrations. For all simulations the concentration profile was recorded along the z-axis intersecting the geometric origin of the module.

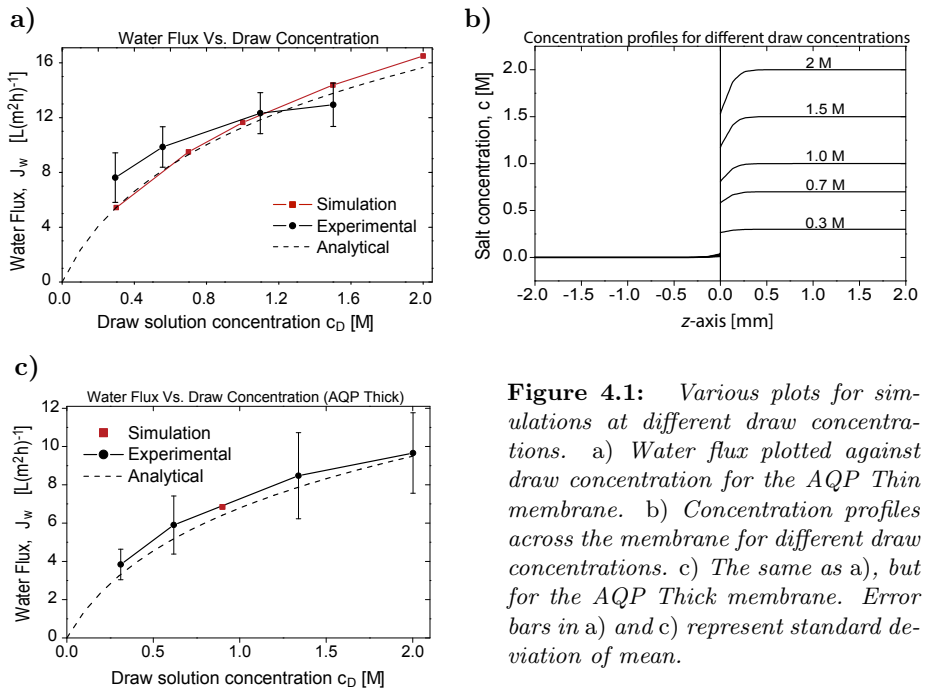


Figure 4.1: *Various plots for simulations at different draw concentrations.* a) *Water flux plotted against draw concentration for the AQP Thin membrane.* b) *Concentration profiles across the membrane for different draw concentrations.* c) *The same as* a)*, but for the AQP Thick membrane. Error bars in* a) *and* c) *represent standard deviation of mean.*

It is seen from Fig. 4.1.a that there is a good correlation between the results obtained from simulation and those obtained analytically, using Eq. (2.4). This was expected and provides further validation for the OF-FO model. J_w does not increase linearly with c_D, but follows an exponentially decaying trend. This is due to the increasing severity of ECP with increasing c_D as illustrated in Fig. 4.1.b. It is noted that the experimental data seems to follow a trend that is different from the two other sets, which was unexpected since the model has been previously validated in comparison to experiments[21]. Considering Fig. 4.1.b which plots J_w against c_D for the AQP Thick membrane, it is however noted that the analytical and experimental data compare quite well, with a small offset, and that the simulation data point lies even closer to the trend of the experimental data. Keeping in mind that the model is previously validated[21], and that Fig. 4.1.a shows the simulation and analytical results to follow roughly the same curve, it is safe to conclude that the model accurately predicts water fluxes for the AQP Thick membrane. Reasons as to why the experimental and simulated data for the AQP Thin membrane differs to such a degree, must be found in the physical properties of the membrane, and the way the experiment is conducted. It can be argued, that since the thicker membrane has a wider support layer, it is less prone to deform under high pressures, especially in the regions below the inlets. The peristaltic pump, furthermore, delivers a periodical flow, which works to deform the membrane in "bursts", and in the

case of the thinner and easily deformed membrane, the geometry dependent mass transfer coefficient in the flow, k_f, may thus begin to vary along the x-axis (see Fig. 4.2), and deviate from what is predicted by Eq. (2.7) by a critical amount. This would introduce a large degree of uncertainty, not accounted for in the model. It should also be noted that due to the transient nature of the concentration and measured water fluxes (see Sec. 3.1), measuring consistent values of the fluxes in repeated experiments was very difficult.

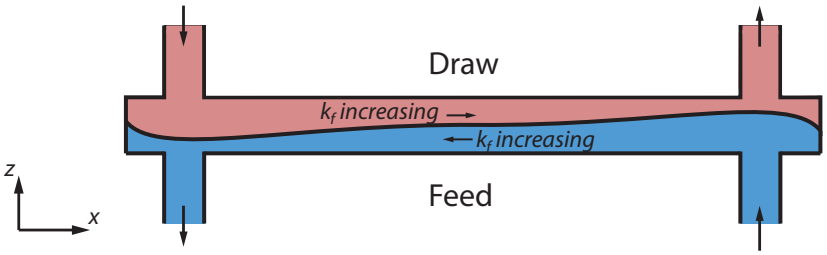


Figure 4.2: *A conceptual illustration of the argument that membrane deformation works to vary k_f across the chamber. An analytical expression for k_f is given in Eq.* (2.7).

4.2 Cross Flow Rate as a Variable

It is generally known that higher cross flow rates work to reduce the severity of ECP and thus promote higher water fluxes through the membrane[22]. To show this, simulations for flow rates of 15, 30, 50, 100, 150, 200 and 250 mL min^{-1} are presented below.

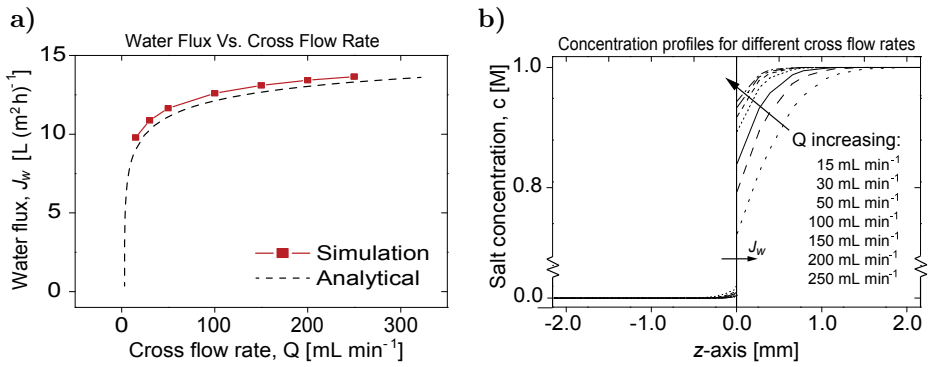


Figure 4.3: a) J_w *vs.* Q *for simulated results at steady state, and analytically determined values.* b) *Concentration profiles for the seven different simulations at different cross flow rates, plotted along the z-axis through the geometrical origin.*

Fig. 4.3.a shows good correlation between simulation results and analytical values calculated using Eq. (2.4). In 4.3.b, a significant dilutive ECP is observed

on the draw side for flow rates < 150 mL min^{-1}. This confirms the assumption, made in Subsec. 2.1.3, that ECP on the draw side cannot be neglected for low cross flow rates. On the feed side, ECP is observed to be insignificant, which is consistent with the results obtained in [17], where it was observed that concentrative ECP on the feed side, was significant only for cross flow velocities comparable to the membrane water flux velocity. In this case, the lowest cross flow rate yielded a cross flow velocity tangential to the membrane approximately 10^3 times larger than the trans-membrane water flux it promoted.

Since concentrative ECP on the draw side cannot be neglected, the direct impact on local concentration is investigated. Fig. 4.4 shows two plots that illustrate the relationship between impact and cross flow rate, as determined by simulation.

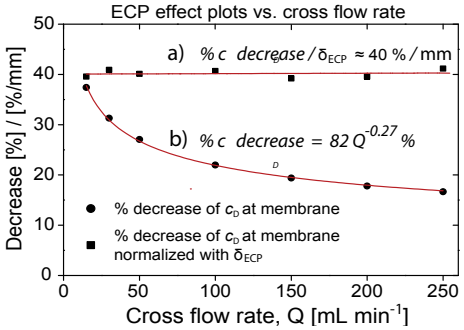


Figure 4.4: *Plots showing the relationships between the percentage-wise decrease of c_m compared to c_D. Plot* a) *is the data of plot* b)*, normalized with respects to the boundary layer thickness δ_{ECP}. The straight line indicates a direct proportionality between concentration decrease and δ_{ECP}.*

Plot b) in Fig. 4.4 shows the percentage of concentration decrease at the membrane in steady state i.e. $(c_D - c_m)/c_D$, for each simulation at different cross flow rate. It is shown that the data can be fitted to a power function with coefficient of determination $R^2 = 0.998$, which yields the empirical relation:

$$\%\ c_D \text{ decrease } = 82\, Q^{-0.27} \tag{4.1}$$

This specific relation is only valid for the membrane parameters and module geometry used in the current simulations, and new simulations would therefore have to be conducted to obtain a similar expression for a different membrane, module or both. Plot a) in Fig. 4.4 shows the values of plot b), normalized with respects to *boundary layer thicknesses* δ_{ECP}, which is determined as the distance from the membrane where the draw concentration has decreased by 10^{-3}%. It is observed that the data points found though simulation assume a horizontally linear trend, showing that there is a direct proportionality between the severity of ECP and the thickness of the boundary layer.

4.3 The Effect of a Slip Boundary Condition

In fluid dynamics theory, it is commonly assumed that fluid velocity at a boundary is zero, i.e. that there is no *slip velocity* present in the flow. This may however not be an accurate assumption in a case such as this, where it is in fact desirable to have a rough membrane surface that can create slip velocities which stir the boundary layer and reduce ECP. In Fig 4.5, results from six simulations for different membrane slip velocities, are presented. The slip parameters range from 1×10^{-6} m to 7.5×10^{-5} m. This corresponds to slip velocities between 1/5 and 1/3 of the mean cross flow velocity tangential to the membrane, $\overline{U} = 9.5$ mm s^{-1}, calculated by dividing the flow rate with the area it passes through above the membrane, $W \times H$ (see Appendix B) .

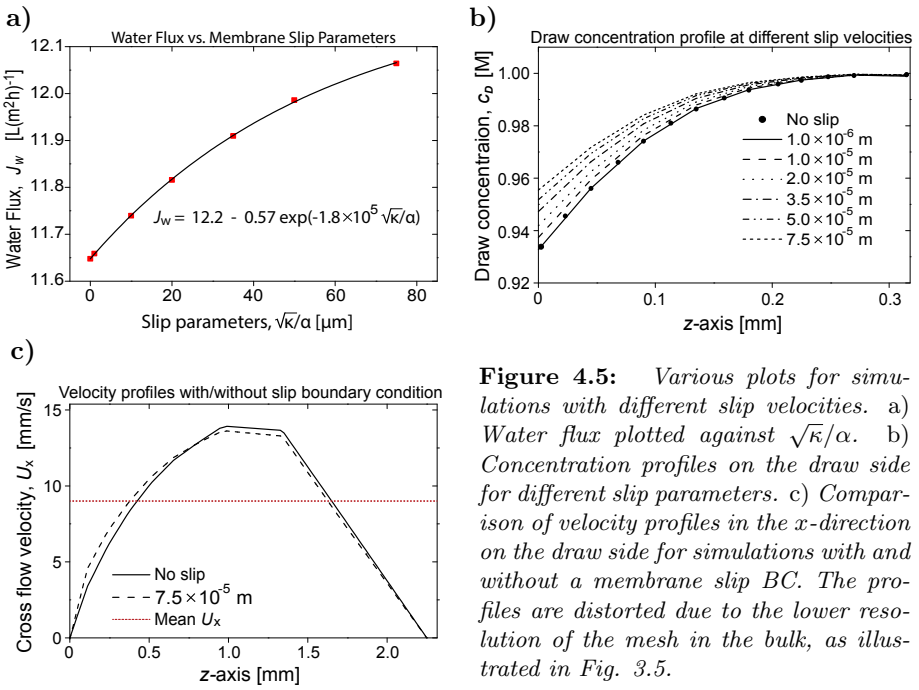


Figure 4.5: *Various plots for simulations with different slip velocities.* a) *Water flux plotted against* $\sqrt{\kappa}/\alpha$. b) *Concentration profiles on the draw side for different slip parameters.* c) *Comparison of velocity profiles in the* x*-direction on the draw side for simulations with and without a membrane slip BC. The profiles are distorted due to the lower resolution of the mesh in the bulk, as illustrated in Fig. 3.5.*

Fig. 4.5.a shows that there is an exponential proportionality between J_w and $\sqrt{\kappa}/\alpha$, expressed as a function $J_w = 12.2 - 0.57\exp\left(-1.8\times10^5\sqrt{\kappa}/\alpha\right)$, which fits the data with $R^2 = 0.9998$. In Fig. 4.5.b it is observed that there is a small degree of ECP reduction on the draw side which is what accounts for the small increase in J_w observed in Fig. 4.5.a. Fig. 4.5.c compares the velocity profiles, tangential to the membrane, of flows with slip and no slip BCs. It is observed that the dashed curve, representing the slip BC profile, displays a higher cross flow velocity close to the membrane, compared to the profile of the flow with a no slip BC. This was expected and corresponds well with the observations made in Fig. 4.5.a and Fig. 4.5.b.

4.4 Introduction of Simple Spacers

Studies of spacer impact on ECP in simplified 2D geometries is thoroughly documented in the literature, and several models have been proposed to describe this phenomenon [7, 23, 24]. A CFD study of spacer impact in complex 3D geometries taking into account membrane properties, such that trans-membrane water fluxes can be related to spacer geometry/density, has however not been documented so far. In the following I present results from four simulations where a different number of the same simple spacer geometry (a triangular prism) was embedded into the module `shell` patch parallel to the membrane plane (see Fig. 4.8). It is noted that better results were likely to have been obtained with more complex spacer geometries, however, the following is just to show that even very simple spacers have an observable and positive influence on J_w.

Fig. 4.6 below shows the z-component of the velocity field in the membrane plane for five different spacer densities.

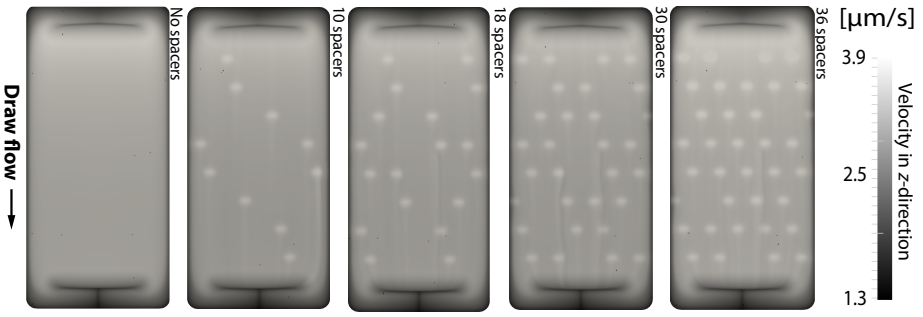


Figure 4.6: *Trans-membrane velocity fields for different spacer densities.*

Looking at the different trans-membrane velocity fields, it is apparent that spacers promote higher water fluxes in the regions directly below them, and furthermore creates a trace of turbulence in the flow direction. Comparing the four fields, the increasing brightness with spacer density, suggests that the overall J_w is increased by a higher number of spacers. This is confirmed by the plot shown in Fig. 4.7 that illustrates an approximately linear relationship ($R^2 = 0.97$) between J_w and spacer density.

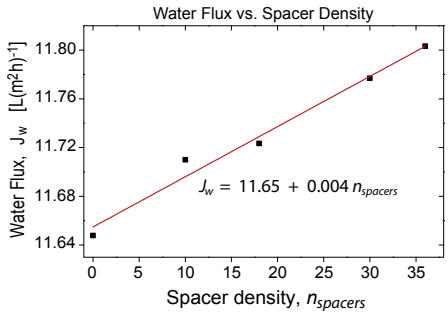


Figure 4.7: *Simulated values of J_w plottet against spacer density. The fit suggests an approximately linear relationship, with coefficient of determination $R^2 = 0.97$. As it may appear on the previous figure, the spacers were places such that they had little influence on one another.*

To investigate the local impact of each spacer, Fig. 4.8 shows plots of the concentration at the membrane, directly below the indicated spacer in directions perpendicular and parallel to the flow, i.e. the y and x axes, respectively.

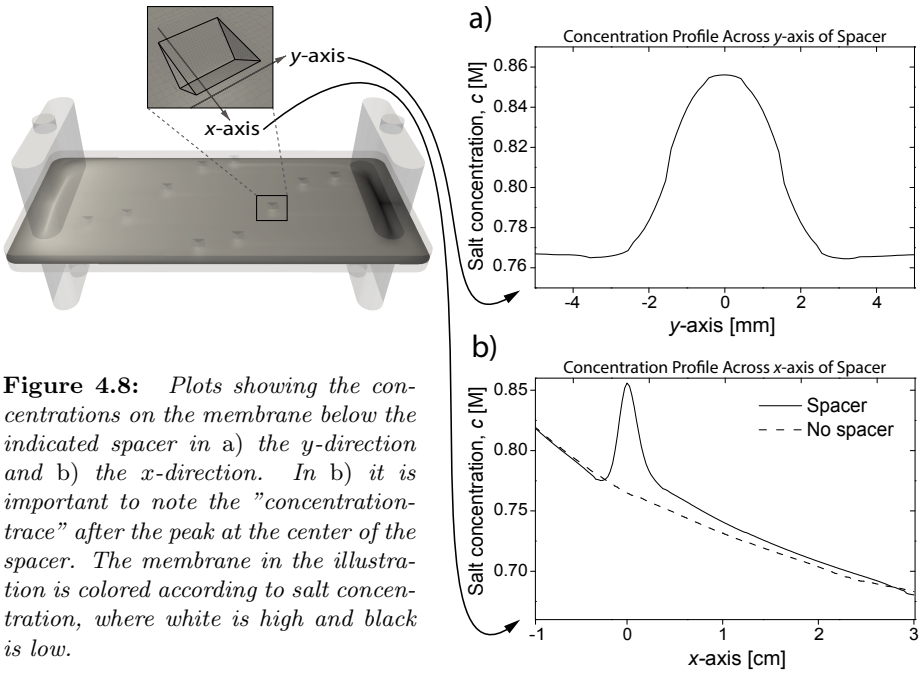


Figure 4.8: *Plots showing the concentrations on the membrane below the indicated spacer in* a) *the y-direction and* b) *the x-direction. In* b) *it is important to note the "concentration-trace" after the peak at the center of the spacer. The membrane in the illustration is colored according to salt concentration, where white is high and black is low.*

It is observed that the concentration at the membrane sees an increase of approximately 12% across the spacer.

The above results point out only the positive effects of spacers in membrane modules. It is stressed that since spacers inherently must obstruct the flow to promote turbulence, they will also increase the pressure within the module. This can have a number of negative effects on the performance of the membrane. In Fig. 4.9 it is shown that an increase in spacer density will increase the hydraulic pressure on the membrane, especially in the regions below the inlets.

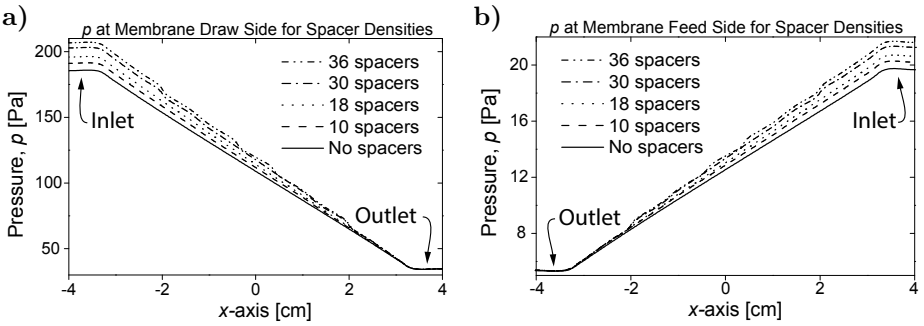


Figure 4.9: *Plots of pressure along the membrane x-axis in* a) *the draw compartment and* b) *the feed compartment of the module, for different spacer densities. The pressure on the membrane peaks in the region below the inlet and decreases linearly to its minimum value below the outlet.*

From the plots in Fig. 4.9 it is evident that pressure across the membrane and in particular below the inlets increase with spacer density. It is noted that the pressures are defined as the *hydrostatic-free pressure* $p_{rgh} = p - \rho(\mathbf{g} \cdot \mathbf{x})$, meaning the the pressure in the module is purely hydraulic. Comparing a module with no spacers to a module with 36 spacers, it is calculated that the pressure sees an increase of 7.6% below the draw side inlet, and 9% below the feed side inlet. The effect that this may have on the membrane is illustrated by Fig. 4.2, which shows a sheared membrane profile due to pressures one the inlet.

To further understand this phenomenon, a shear profile of the membrane can be calculated. Simulating the deformation of the membrane in OpenFOAM is also possible using the `solidDisplacementFoam` solver, and approximating the membrane to an impermeable surface with a Young's modulus. Recalling the discussion of Sec. 4.1, it would be beneficial in such a simulation, to set the inlet BCs to varying velocity fields, such as to better represent the periodicity of the flow rate delivered by a peristaltic pump. The limited amount of time given for this project did, however, not allow for such an investigation, and it is therefore added to the list of future perspectives.

4.5 Suggestions for Module Optimization

The Sterlitech CF042 membrane module is one of the most commonly used test modules in membrane research, and therefore an understanding of the flow conditions within the module could potentially be of interest to the scientific community. Analysis of the flow patterns in the draw compartment of the module are presented in the following, for a standard condition simulation (see Table 3.1). Fig. 4.10 show stream-lines close to the membrane, and Fig. 4.11 shows stream-lines in the bulk.

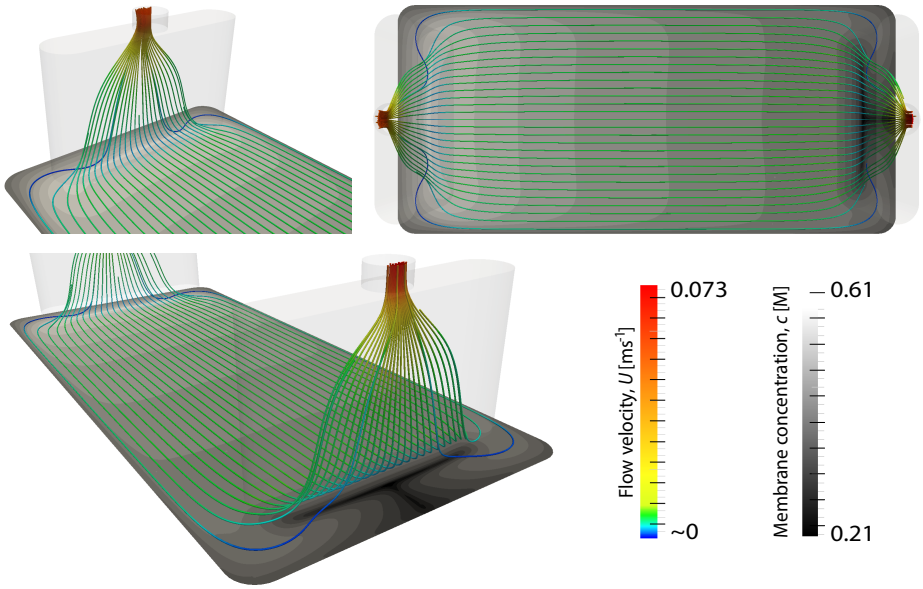


Figure 4.10: *Velocity stream-lines close to the membrane in the draw compartment of the module. The stream-lines are colored according to the magnitude of the velocity field, and the membrane is colored according to the salt concentration in a low-resolution grey scale only to guide the eye.*

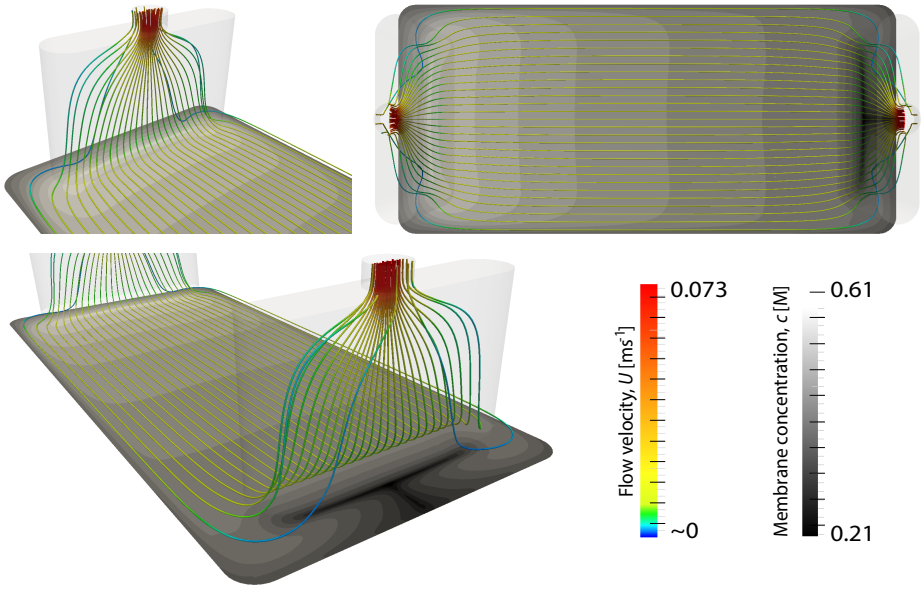


Figure 4.11: *Velocity stream-lines in the bulk of the module draw compartment. The stream-lines are colored according to the magnitude of the velocity field, and the membrane is colored according to the salt concentration.*

As expected, colors of the flow velocity stream-lines in Fig. 4.10 and Fig. 4.11, suggest a faster flow in the bulk region than in the vicinity of the membrane. Furthermore, in specific areas on the membrane, the salt concentrations are seen to be very low and so is the presence of stream-lines, i.e. ECP is highly pronounced. Fig 4.12 shows a plot of the concentration on the membrane draw side and trans-membrane water flux along the x-axial flow direction.

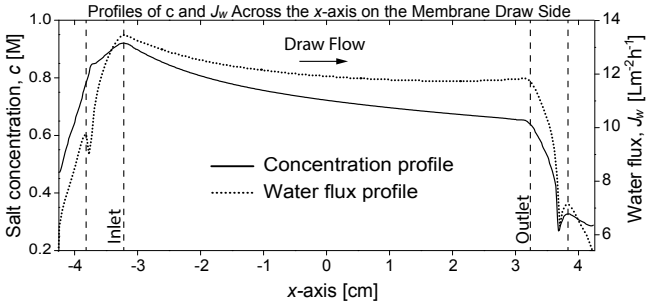


Figure 4.12: *Plots of the concentration on the membrane and trans-membrane water flux along the flow direction (x-axis). It is observed that very high differences in the concentration are present on the membrane, which is reflected by the trend of the trans-membrane water flux.*

It is observed from Fig. 4.12 that there is a very large difference in concentration on the membrane between the inlet and outlet regions. Comparing the minimum and maximum values of concentration, a variation of approximately 65% is calculated. As it is seen on the dotted plot in Fig. 4.12, this variation of solute concentration is reflected in the trans-membrane water flux. It is calculated that the variation between the maximum J_w below the inlet and the outlet is approximately 12%, which is an acceptable variation, however when comparing the global minimum value with the global maximum, the variation is approximately 60%.

This is a very large variation which inherently introduces a degree of uncertainty when testing membrane performance. For an ideal module, the J_w profile should not have such great lows, but instead assume a slowly decreasing profile across the entire cross-section, such as it does between the inlet and outlet.

One obvious design aspect which can be reconsidered in this respect, is the placement of the inlets and outlets in the module. It is suggested by Fig. 4.12 that moving the inlets and outlets to the end positions of the chamber, would eliminate the "dead areas" observed in the outer locations. It is speculated that this would give rise to a lower global variation of trans-membrane water flux, creating flow conditions that would promote a higher average water flux through the membrane. Furthermore, design aspects such as inlet and outlet direction would likely influence module performance. It can be argued that a

more tangential inlet/outlet direction would redistribute the pressure on the membrane to a larger area, reducing the shear of the membrane, which was possibly accountable for the unexpected deviations from experimental data for the AQP Thin membrane, observed in Sec. 4.1.

5 Conclusions

There were two main goals in this project. The first was to develop a generic working process for performing CFD simulations of FO filtration using OpenFOAM. The second was to run a number of simulations to investigate the flow conditions for different varying parameters. In summary, the work presented and the conclusions drawn in this thesis are:

- The theory relevant for osmotic membrane separation and mass transport through membranes in FO was presented. A fundamental introduction to CFD was given, outlining the methodology and governing equations for fluid and solute flow. The basics of OpenFOAM usage was presented along with a description of the OF-FO model used for simulation.
- A generic work process for setting up and performing simulations on complex membrane module geometries in OpenFOAM was developed, taking the Sterlitech CF042 membrane module as an example.
- The OF-FO model was applied for a number of different cases, proving a high level of accuracy in comparison with analytical results. It was concluded to consistently predict the impact of ECP to such a degree that a direct proportionality between percentage decrease of concentration and boundary layer thickness, could be established (Fig. 4.4). The error found in comparison to experimental results in Sec. 4.1 was conclusively assigned to experimental reasons, and the fact that only a small number of repeated experiments were performed.
- It was shown that parameter setting in all four simulation series could reduce the severity of ECP and enhance the trans-membrane water flux. Properties of the system that was shown to reduce ECP was: *lower* draw concentrations and *higher* cross flow rates, membrane slip velocities and spacer densities.
- It was concluded that while spacers had a positive effect on water flux through the membrane, they also increased the pressure within the module. It was argued that since the increased pressure was distributed primarily to the regions by the inlets, an increased shear of the membrane would occur for a higher number of spacers.
- The flow patterns within the CF042 module was presented as stream lines inside of the draw chamber of the module. Based on plots showing the values of local concentration and trans-membrane water flux along the flow direction, two propositions for module optimization were made, both

focusing on the position and orientation of the inlets and outlets of the module.

In general, the goals of this thesis are considered reached. What this thesis has to offer is thus a comprehensible introduction to a very powerful simulation tool, capable of simultaneously resolving the effects of different parameters in a given complex membrane module geometry. For any realistic FO process where it is essential that the flow chamber is fully optimized, the working process developed in this work can readily be applied to quantify the effects of modifications, made to the flow conditions and the chamber geometry.

5.1 Future Perspectives

The working process presented in this thesis provides a stepping stone for performing simulations of flow in complex chamber geometries under different flow conditions. While this has in fact been successfully done for a range of settings, further analysis can be made to derive shear-profiles for the membrane for different pressures and flow rates. The actual deformation of the membrane under different flow conditions can furthermore be simulated in OpenFOAM by characterizing the membrane with a Young's modulus and running the simulation using the `solidDisplacementFoam` solver.

Regarding the module geometry, a large number of possibilities within optimization remains to be investigated. For the Sterlitech CF042 module, it was evident that the introduction of spacers promoted higher trans-membrane water fluxes. Furthermore, results suggest that improvements could be made to the flow conditions, by changing the position and direction of the inlets and outlets. Since the CF042 module is widely used for research in FO technology, simulation based research for module optimization, would likely be of interest to the scientific community and the Sterlitech Corporation.

References

[1] Raphael Semiat and David Hasson. Water desalination. *Reviews in Chemical Engineering*, 28(1):43–60, 2012.

[2] Laura Chekli, Sherub Phuntsho, Ho Kyong Shon, Saravanamuthu Vigneswaran, Jaya Kandasamy, and Amit Chanan. A review of draw solutes in forward osmosis process and their use in modern applications. *Desalination and Water Treatment*, 43(1-3):167–184, 2012.

[3] Tzahi Y. Cath, Amy E. Childress, and Menachem Elimelech. Forward osmosis: Principles, applications, and recent developments. *Journal of Membrane Science*, 281(1-2):70–87, 2006.

[4] Chuyang Y. Tang, Qianhong She, Winson C. L. Lay, Rong Wang, and Anthony G. Fane. Coupled effects of internal concentration polarization and fouling on flux behavior of forward osmosis membranes during humic acid filtration. *Journal of Membrane Science*, 354(1-2):123–133, 2010.

[5] Alberto Tiraferri, Ngai Yin Yip, Anthony P. Straub, Santiago Romero-Vargas Castrillon, and Menachem Elimelech. A method for the simultaneous determination of transport and structural parameters of forward osmosis membranes. *Journal of Membrane Science*, 444:523, 2013.

[6] Jeffrey R. McCutcheon and Menachem Elimelech. Influence of concentrative and dilutive internal concentration polarization on flux behavior in forward osmosis. *Journal of Membrane Science*, 284(1-2):237–247, 2006.

[7] Minkyu Park and Joon Ha Kim. Numerical analysis of spacer impacts on forward osmosis membrane process using concentration polarization index. *Journal of Membrane Science*, 427:10–20, 2013.

[8] M. F. Gruber, C. J. Johnson, C. Y. Tang, M. H. Jensen, L. Yde, and C. Helix-Nielsen. Computational fluid dynamics simulations of flow and concentration polarization in forward osmosis membrane systems. *Journal of Membrane Science*, 379(1-2):488–495, 2011.

[9] *Atkins' Physical Chemistry*. Oxford University Press, eighth edition edition, 2006.

[10] JG Wijmans and RW Baker. The solution-diffusion model - a review. *Journal of Membrane Science*, 107(1-2):1–21, 1995.

[11] EMV Hoek, AS Kim, and M. Elimelech. Influence of crossflow membrane filter geometry and shear rate on colloidal fouling in reverse osmosis and

nanofiltration separations. *Environmental Engineering Science*, 19(6):357–372, 2002.

[12] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

[13] J. Anderson. *Computational Fluid Dynamics*. Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill Education, 1995.

[14] G. A. Fimbres-Weihs and D. E. Wiley. Review of 3d cfd modeling of flow and mass transfer in narrow spacer-filled channels in membrane modules. *Chemical Engineering and Processing*, 49(7):759–781, 2010.

[15] Openfoam official website. http://www.openfoam.org.

[16] Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62(1):40–65, 1986.

[17] M. F. Gruber. Computational fluid dynamics simulations of osmotic membrane separation processes. Master thesis, Technical University of Denmark, 2011.

[18] S. Chellam, MR Wiesner, and C. Dawson. Slip at a uniformly porous boundary - effect on fluid-flow and mass transfer. *Journal of Engineering Mathematics*, 26(4):481–492, 1992.

[19] Blender official website. http://www.blender.org/.

[20] Installation/mac os/openfoam 2.1.x. http://openfoamwiki.net/.

[21] Claus Helix-Nielsen, Mogens H. Jensen, Carl J. Johnson, Chuyang Tang, Mathias F. Gruber, and Lars Yde. Validation and analysis of forward osmosis cfd model in complex 3d geometries. *Membranes*, 2(4):764–782, 2012.

[22] R. Baker. *Membrane Technology and Applications*. Wiley, 2004.

[23] Lawrence C. Murdoch, David A. Ladner, and Peng Xie. Hydrodynamics of sinusoidal spacers for improved reverse osmosis performance. *Journal of Membrane Science*, 453:92–99, 2014.

[24] Wenqiang Mi and Songlin Xu. A two-dimensional cfd simulation for different spacers of spiral wound moudles. *Advanced Materials Research*, 557-559(557-559):2249–2252, 2012.

[25] Virtualbox official website. https://www.virtualbox.org/.

[26] Parallels official website. http://www.parallels.com/.

[27] Cfd online discussion forums. http://www.cfd-online.com/.

[28] Appendix e: Complete source code. http://nanomathias.com/.

Appendices

A Deriving the Flux Equations

The flux equations for water and salt transport across an asymmetric semi-permeable membrane is derived in the following. Fig. A.13 explains the interfaces and regions of the feed-membrane-draw system which are referenced in the following.

The flux equation (2.4) is derived in three parts, by applying a diffusion-convection model to one interface at a time. This derivation starts from the left. Initially, however, it is established that the water flux across the active layerof the membrane is given by:

$$J_w = A(\Delta\pi) \tag{A.1}$$

where $\Delta\pi$ is the effective osmotic pressure that drives the flow. This expression assumes that there is 100 % solute rejection at the membrane, and is therefore only valid for very salt-rejecting membranes. Reversely, the salt flux in the opposite direction across the membrane is given by:

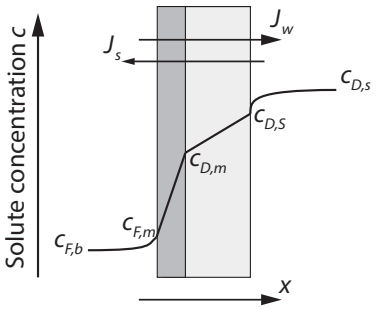


Figure A.13: *Illustration of the interfaces and regions of the assymetric membrane which are referenced in this appendix.*

$$-J_s = -B(c_{D,m} - c_{F,m}) \tag{A.2}$$

As water permeates the membrane, a small fraction of salt is rejected at the membrane which leads to a usually negligible concentrative ECP on the feed side. This ECP is in turn reduced slightly by diffusion of solute. The pure water permeate stream convectively dilutes the salt concentration in the porous support layer, resulting in dilutive ICP. The permeate then enters the draw side and dilutes the concentration in the unstirred boundary layer. Diffusion in the draw solution works to restore some of the lost concentration, resulting in a balance of fluxed that defines the dilute ECP profile on the draw side.

Feed-Membrane Interface The ECP within the instirred boundary layer on the feed side is governed by the convection of solution towards the draw, and diffusion of solute toward the feed, as expressed in the following

equation:

$$-J_s = -D\frac{dc(x)}{dz} + J_w c(x) \tag{A.3}$$

Eq. (A.3) is equated with (A.2) which yields the equation:

$$-\frac{dc(x)}{dx} + \frac{J_w}{D}c(x) = -\frac{B}{D}(c_{D,m} - c_{F,m}) \tag{A.4}$$

This equation is solved with respects to boundary conditions $c(-\delta) = c_{F,b}$ and $c(0) = c_{F,m}$, to obtain:

$$c_{F,m} = c_{F,m}\exp\left(\frac{J_w}{k}\right) + \frac{B}{J_w}(c_{D,m} - c_{F,m})\left(\exp\left(\frac{J_w}{k}\right) - 1\right) \tag{A.5}$$

were $k = D/\delta$.

Membrane-Support Interface For the interface where the concentration on Fig. A.13 is $c_{F,m}$ the reverse salt flux can be expressed as the sum of diffusive and convective contributions:

$$-J_s = -D_s\frac{dc(x)}{dx} + J_w c(x) \tag{A.6}$$

Eqs. (A.2) and (A.6) are equated, yielding the expression:

$$\frac{dc(x)}{dx} - \frac{J_w}{D_s}c(x) = \frac{B}{D_s}(c_{D,m} - cF, m) \tag{A.7}$$

Solving this equation with respects to the boundary conditions $c(0) = c_{D,S}$ and $c(-t_s) = c_{D,m}$, the following expression is obtained:

$$c_{D,m} = c_{D,S}\exp\left(-\frac{J_w S}{D}\right) + \frac{B}{J_w}(c_{D,m} - c_{F,m})\left(\exp\left(-\frac{J_w S}{D}\right) - 1\right) \tag{A.8}$$

where it has been used that $S = t_s\tau/\epsilon$.

Support-Draw Interface At this interface where the concentration is $c_{D,S}$ the diffusion-convection model gives the equation:

$$J_s = D\frac{dc(x)}{dx} - J_w c(x) = 0 \tag{A.9}$$

Solving this equation with respects to the boundary conditions $c(0) = c_{D,b}$ and $c(-\delta) = c_{D,S}$ yields:

$$c_{D,S} = c_{D,b}\exp\left(-\frac{J_w}{k}\right) \tag{A.10}$$

Combining the Equations Both $c_{D,m}$ and $c_{F,m}$ are interfacial concentrations on either side of the active layer, and are therefore not experimentally accessible. This is circumvented by subtracting one from the other such as to obtain an expression for the concentation gradient that drives the water flux across them membrane.

$$c_{D,m} - c_{F,m} = \frac{c_{D,S} \exp\left(-\frac{J_w S}{D}\right) - c_{F,b} \exp\left(\frac{J_w}{k}\right)}{1 + (B/J_w)\left[\exp\left(\frac{J_w}{k}\right) - \exp\left(-\frac{J_w S}{D}\right)\right]} \tag{A.11}$$

Eq. (A.10) is substituted into the equation to account for ECP on the draw side. The van't Hoff equation is applied to Eq. (A.11), assuming that concentration is linearly dependent on concentration difference, which yields expression for the effective osmotic pressure across the active layer. This is substituted into Eq. (A.1) and assigned a normal vector, yielding the expression for water flux presented in Subsec. 2.1.3:

$$\mathbf{J}_w = A \frac{\pi_D \exp\left(-J_w \left[\frac{1}{k_m} + \frac{1}{k_f}\right]\right) - \pi_F \exp\left(\frac{J_w}{k_f}\right)}{1 + \frac{B}{J_w}\left[\exp\left(\frac{J_w}{k_f}\right) - \exp\left(-\frac{J_w}{k_m}\right)\right]} \hat{\mathbf{n}}_w \tag{2.4}$$

An expression for the salt flux can be obtained by substitution of Eq. (A.11) into (A.2):

$$\mathbf{J}_s = -B \frac{c_D \exp\left(-J_w \left[\frac{1}{k_m} + \frac{1}{k_f}\right]\right) - c_F \exp\left(\frac{J_w}{k_f}\right)}{1 + \frac{B}{J_w}\left[\exp\left(\frac{J_w}{k_f}\right) - \exp\left(-\frac{J_w}{k_m}\right)\right]} \hat{\mathbf{n}}_w \tag{2.5}$$

B Dimensions of the Module Draw Chamber

The official Sterlitech website provides no information about the exact dimensions of their CF042 module. The dimensions given in this appendix are measured by the author, using a caliper with sub-mm precision.

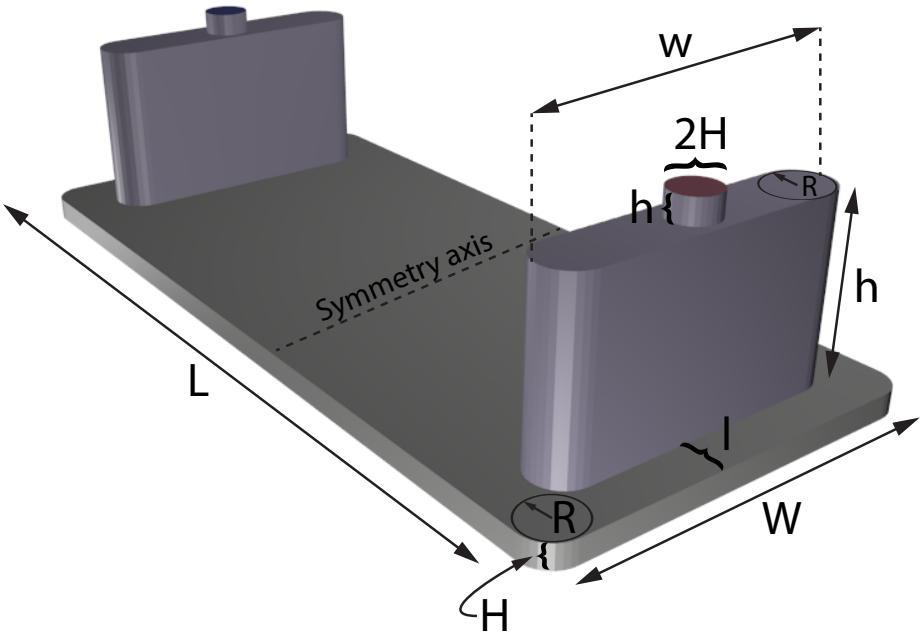


Figure B.14: *Dimensions of the CF042 membrane module, as measured with a caliper. Values are given in Tab. B.1*

Dimension	L	W	H	R	l	w	h
Value	8.5	3.9	0.225	0.3	0.425	2.95	1.7

Table B.1: *Values for the dimensions illustrated in Fig. B.14, given in units of* cm.

C Getting Started

This section serves as a brief introduction on how to get started using the OF-FO model in OpenFOAM. It is presented in two steps: Installation of OpenFOAM and ParaView on a UNIX-based operating system (e.g. Mac OS X®, Linux Ubuntu®), and compilation of the OF-FO model with OpenFOAM. A subsection on how to gain access and submit jobs to the DTU HPC as a DTU user is also presented in the following.

C.1 Installing OpenFOAM

OpenFOAM is a UNIX based software specifically developed for use in the Linux based operating system Ubuntu. Installing it in Ubuntu is very straight forward and can be done using the `apt` package management tool in `Terminal`. Instructions are provided on the official OpenFOAM website[15].

Running OpenFOAM on Mac OS X can be done either by using a virtualization software (such as *VirtualBox*® [25] or *Parallels Desktop*® [26]) to create a virtual Linux machine on which the software can be installed, or - slightly more complicated but much more rewarding - by installing it on the Mac OS X itself. The latter of these installations is described in detail online [15].

For new users that are not well acquainted with OpenFOAM, it is highly recommended to follow a few of the basic tutorials provided with the installation. Instructions are provided in the OpenFOAM Userguide[15] and furthermore the online CFD community *CFD Online* has discussion forums that can be consulted for help[27].

C.2 Model Compilation with OpenFOAM

As discussed in Subsec. 2.2.3, the OF-FO model was developed in two parts, namely as a solver and as a boundary condition. Therefore it must also be compiled with OpenFOAM as such. Compiling custom solvers and boundary conditions with OpenFOAM follows a standard method, which is also mentioned in the OpenFOAM Userguide[15].

The `pisoSaltTransport` solver and the FO boundary conditions were compiled following these steps:

- Downloading the `pisoSaltTransport` solver and the `FO_BC` folders online[28], and moving them to a fitting location.
- Launching `Terminal`.
- Navigating to the `pisoSaltTransport` folder and executing `wclean` and `wmake`. This should start compiling the solver.

- Navigating to the `FO_BC` and executing `wclean` and `wmake libso`, where the argument `libso` is added because `FO_BC` is a dynamically linked library. The boundary condition should start compilation.
- Having successfully compiled the solver and boundary condition, the last important thing is to add the argument `libs ("libDHIBoundaryConditions.so");` to the `controlDict` file in the cases where the boundary condition is used.

,

C.3 Using the DTU HPC

In this work, all computing jobs were processed by the DTU Compute High Performance Computing center (HPC), due to a great demand of processing power. DTU Compute offers all DTU users access to the HPC, with a standard maximum capacity of 64 processors used at once. The HPC is accessed from the `Terminal` with the command `ssh <username>@hpc-fe.gbar.dtu.dk`, which will prompt the user for a password. This will log the user on to a "front-end" computer where the command `linuxsh` is then executed in order to log on to the HPC.

The HPC already has OpenFOAM installed on it (currently v.2.2.0). Compiling the OF-FO model on the HPC follows exactly the same scheme as presented in Subsec. C.2. Submission of jobs is done by executing `qsub <scriptFileName>.sh` (see Appendix C). Executing `qstat` will produce a table showing submitted jobs and their respective status, run time, job-ID, etc. Simple tasks, such as execution of utilities, can be processed without having to submit them as jobs. To do this the relevant modules must be loaded, by executing:

```
module load gcc/4.7.2-cloog
module load mpi/gcc
module load OpenFOAM/2.2.0/gcc-4.7.2-openmpi
```

For post-processing it is possible to use ParaView through an X-window system. This requires the user to add an `-X` after `ssh` and after `linuxsh` when logging in. ParaView is launched simply by executing `paraview`, *not* `paraFoam` as this command does not work in the HPC. The `case.foam`-file must then be loaded manually which will in turn open the case. The main disadvantage to using an X-window system for visualization is that it is rather slow even on fast internet connections. Therefore it is preferable to do post-processing solely on the work computer. Files can be moved from the HPC to the work computer hard drive using the `rsync` file transfer program command executed from the work computer `Terminal`. As an example, if the file `file` within the folder

`fHPC` on the HPC is to be relocated to the folder `fPC` on the work computer, the following command needs to be executed from the parent directory to `fPC`:

```
rsync -av <username>@hpc-fe.gbar.dtu.dk:/fHPC/file fPC
```

Reversely, to upload `file`, located within the current working directory on the work computer, to `fHPC` on the HPC the command changes to:

```
rsync -av file <username>@hpc-fe.gbar.dtu.dk:/fHPC
```

In both cases the user will be prompted for a password from the DTU server, and the file transfer will start automatically. Note that using the HPC for simulation is not always necessary, as most simple tasks can be handled by the working computer processor. It is strongly recommended to only use the HPC for processing of large jobs, and initial testing to determine whether jobs can run without crashing.

D Online Material

A selected amount of material produced and referenced in this project is made available online, at `http://ulf-memsci.tumblr.com`. This material includes:

- The fully pre-processed and solved simulation case discussed in Sec. 3.2, which holds all necessary data for reproducing the results gathered from that particular case.
- The Blender design user guide written by I. Løvaas.
- Links of interest to this work.
- This very thesis.