NTNU

Norwegian University of Science and Technology

Modeling and simulation of a membrane module

Michal Galis

Chemical Engineering and Biotechnology

Submission date: May 2017

Supervisor: Heinz A. Preisig, IKP

Norwegian University of Science and Technology

Department of Chemical Engineering

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-5433-61027

Modeling and Simulation of a Membrane Module

Diploma thesis

2016/2017 Bc. Michal Galis

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-5433-61027

Modeling and Simulation of a Membrane Module

Diploma thesis

Study programme: Chemical Engineering

Study field: Chemical Engineering

Training workplace: NTNU, Department of Chemical Engineering

Thesis supervisor: Ing. Juraj Labovský, PhD.

Consultant: Prof. Heinz A. Preisig

2016/2017

Bc. Michal Galis

Slovak University of Technology in Bratislava
Institute of Chemical and Environmental Engineering

Faculty of Chemical and Food Technology
Academic year: 2016/2017
Reg. No.: FCHPT-5433-61027

MASTER THESIS TOPIC

Student:	**Bc. Michal Galis**
Student's ID:	61027
Study programme:	Chemical Engineering
Study field:	5.2.17. Chemical Engineering
Thesis supervisor:	Ing. Juraj Labovský, PhD.
Consultant:	prof. Heinz A Preisig
Workplace:	Norwegian University of Science and Technology, Department of Chemical Engineering

Topic: **Modeling and Simulation of a Membrane Module**

Language of thesis: English

Specification of Assignment:

There has been a rise to the adoption of Computational Fluid Dynamics (CFD) in recent years. It is mostly due to the reliability and efficiency of these modern numerical methods. This work will focus on modeling and simulation of a membrane module using OpenFoam software. The mentioned module is used for separation of organic acids dissolved in water.

Length of thesis: 43

Selected bibliography:

1. Dojčanský, J. – Longauer, J. *Chemické inžinierstvo II.* Bratislava : Malé centrum, 2000. 383 s. ISBN 80-967064-8-9.

Assignment procedure from: 13. 02. 2017

Date of thesis submission: 21. 05. 2017

Bc. Michal Galis
Student

prof. Ing. Milan Polakovič, CSc.
Head of department

prof. Ing. Milan Polakovič, CSc.
Study programme supervisor

Acknowledgements

I would like to thank my supervisor, Professor Heinz A. Preisig for his guidance and advise during the work on my thesis.

I would also like to express my gratitude to Sigve Karolius and Abel Mekonnen. Both of them have been a great help in understanding and figuring out the ways of OpenFOAM software.

Abstract

This work presents a way of simulating the operation of an anion exchange membrane module. A CFD simulation of this module was made in OpenFOAM software. The membrane module contains an ion exchange membrane and is used for separation of organic acids. It consists of an dilute and a concentrate compartments, separated by the mentioned membrane. In the model, only acetic acid was taken into account. A movement of acetic acid ions from the dilute compartment, trough the membrane and to the concentrate compartment under the effect of an electric field was assumed. Electrochemical reactions were not taken into account. First, a mesh describing the membrane module geometry is build. Two meshes have been built, a simplified one and a more detailed one. Then, pre-process utilities that have been used in this work are explained followed by boundary conditions. After that, the solver of this model is explained along with post-processing tools. The results present a simple, functioning membrane module model. Lastly, a number of future development possibilities are discussed.

Keywords: CFD, simulation, model, membrane module

Abstrakt

Predložená práca sa zaoberá vytvorením CFD modelu a simuláciami mebránového modulu. Všetky výpočty a simulácie boli uskutočnené v programe OpenFOAM. Tento membránový modul sa skladá z dvoch nádob oddelených iónovou membránou a slúži na separáciu zmesi organických kyselín. Model pracuje s kyselinou octovou. Konkrétne sa jedná o prestup záporne nabitých iónov, ktoré vznikli jej disociáciou, cez iónovú membŕanu pod vplyvom elektrického poľa. Model pracuje s koncentráciou octových aniónov, elektrochemické reakcie nie sú brané do úvahy. Základom CFD simulácií je vytvorenie mriežky. V práci sú opísané dve mriežky membránového modulu, zjednodušená a presná varianta. Nasleduje opis použitých utilít, ktoré zahŕňajú vytvorenie mriežky a priradenie okrajových podmienok. Potom je opísaný proces vytvorenia solvera a samotný výpočet. Výsledky potvrdzujú vytvorenie jednoduchého modelu membránového modulu. Následne sú opísane možné vylepšenia v budúcnosti.

Kľúčové slová: CFD, simulácia, model, membránový modul

Contents

Nomenclature

Greek Symbols

$\bar{\eta}$	electro-chemical potential in membrane [J.mol^{-1}]
η	electro-chemical potential [J.mol^{-1}]
Λ	molar conductivity [S.m^{2}.mol^{-1}]
μ	chemical potential [J.mol^{-1}]
ν	kinematic viscosity [m^{2}.s^{-1}]
ϕ	electric potential [V]
ρ	fluid density [kg.m^{-3}]
σ_{dc}	dc conductivity [S.m^{-1}]

Roman Symbols

A	membrane surface [m^{2}]
a	activity [$-$]
C	Courant number [$-$]
c	concentration [mol.m^{-3}]
D	diffusion coefficient [m^{2}.s^{-1}]
E	electric field strength [V.m^{-1}]
F	Faraday constant [C.mol^{-1}]

h	compartment height [m]
I	electric current [A]
k	Boltzmann constant [J.K^{-1}]
P	pressure [Pa]
R	electrical resistance [Ω]
R_g	gas constant [J.mol^{-1}.K^{-1}]
T	temperature [K]
t	time [s]
u	fluid velocity [m.s^{-1}]
V	electric potential [V]
V_{mol}	partial molar volume [m^3.mol]
z	charge number [$-$]
j	electrical current density [A.m^{-2}]

Superscripts

0	standard

Subscripts

aa	acetic anion
$anode$	anode
c	concentrate compartment
$comp$	compartment
d	dilute compartment
i	component
m	membrane

Chapter 1

Introduction

1.1 Computational fluid dynamics

Computational fluid dynamics (CFD) is an analysis of a system involving fluid flow and other associated phenomena such as heat transfer or chemical reactions by computer simulation. It is a very powerful process which spans a wide range from industrial to non-industrial applications. Examples include:

- aerodynamics of vehicles and aircraft
- flows inside rotating passages in various engines
- distribution of pollutants and effluents
- hydrodynamics of ships
- cooling of electrical equipment or microcircuits
- flows in rivers or oceans
- weather prediction
- blood flow trough veins and arteries

The first CFD techniques have been integrated into designs in 1960s by the aerospace industry in manufacture of aircraft and jet engines. The usage of these methods have more recently been broadened to the design of combustion chambers of gas turbines and combustion engines. Nowadays, it is a common practice of many motor vehicle manufactures to predict under-bonnet air flows or drag forces with CFD tools. CFD is becoming one of the vital components in designing of industrial products and processes.

The ultimate aim of CFD development field is to provide an option that is comparable to other computer-aided engineering tools. It is true that CFD has lagged behind and the reason for that is tremendous complexity of the case behaviour. There has been an increase to affordable high-performance computing hardware in the recent 20 years, which together with introduction of user-friendly interfaces led to recent rise of interest.

CFD also provides multiple unique advantages over common experiment based approaches for fluid system designs:

- significant decrease of lead times as well as costs of new designs
- ability to work with systems where experiments are difficult or impossible to perform (large systems)
- ability to work with systems under dangerous conditions at or beyond normal performance limits (safety and accident studies)
- unlimited level of result details

In terms of facility and personnel costs, the cost of an experiment is roughly proportional to the number of data or configurations tested. On the contrary, CFD techniques are able to provide extremely large quantities of results at no extra expense. It is also very cheap to optimize equipment performance. [1]

1.2 Goals of this work

The first goal of this work was to acquire a basis of CFD knowledge and practices. This also includes basic know-how about mesh generation, conversion and usage. The second goal was to explore ways of CFD implementation to create a model of an anion exchange membrane module, which is used for organic acid separation. Here, a few simplifications had to be made since this was the first mapping of the mentioned device and related processes. With further development, this model could be validated with experimental data in the future and possibly predict the behavior of similar systems.

1.3 Membrane module

The membrane module setup comprises a two-chambered electrochemical cell with an anode and a cathode on each side of the device. The frames are bolted together with stainless steel bolts between two square endplates. Inlet and outlet for liquid recirculation are provided in each compartment. An anion exchange membrane is placed between the cathode and the anode.

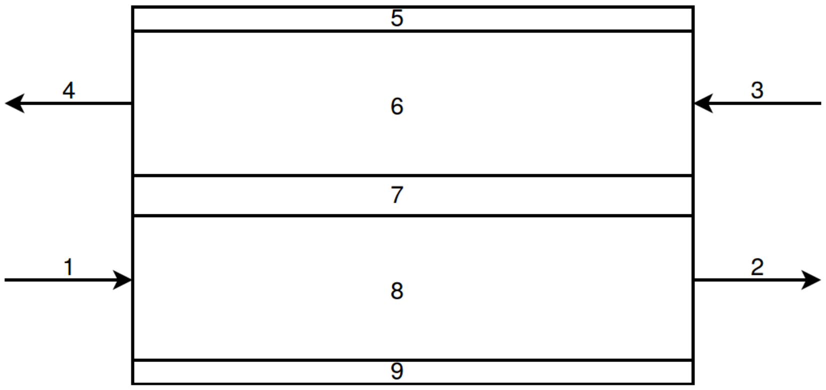


Figure 1.1: Membrane module schematic drawing

In the figure 1.1, 1 is dilute compartment inlet, 2 is dilute compartment outlet, 3 is concentrate compartment inlet, 4 is concentrate compartment outlet, 5 is anode, 6 is concentrate compartment, 7 is membrane, 8 is dilute compartment and 9 is cathode.

The whole membrane module setup consists of a membrane unit, two peristaltic pumps, a power source and feed and product tanks. This module is a part of a larger setup. The aim is to develop a new integrated biotechnological production process for butyl butyrate which is a promising substitute for diesel or jet engine fuels. The membrane module is shown on figure 1.2.

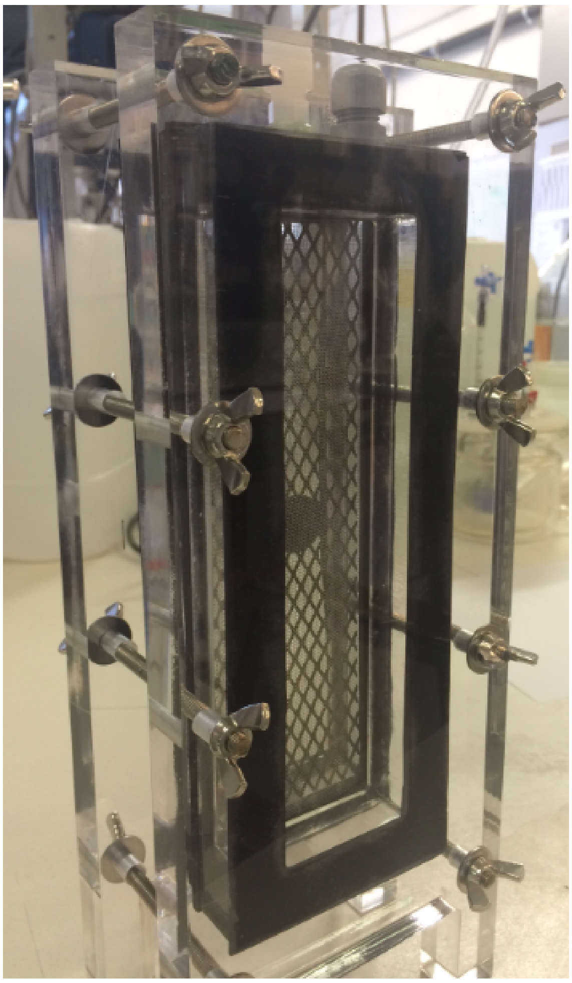


Figure 1.2: Anion exchange membrane module

1.4 Description of the case

The following conditions and simplifications were assumed in the model. The simulations were preformed in isothermal state with laminar flow regime. Although there is a mixture of acids in the experimental setup, only acetic acid is considered in the model. Mentioned acid is dissolved and the acetic anions then migrate trough the membrane into the other compartment. The model operates with a concentration of these dissolved species. The driving force of anions across the membrane is a difference in chemical potential and an electric field.

1.5 Software used

Geometry, meshing and all calculations were performed in OpenFOAM 4.1 software. OpenFOAM (Open source Field Operation And Manipulation) is a C++ toolbox used for the development of numerical solvers, pre- and post-processing utilities for continuum mechanics problem solving, including CFD. A number of OpenFOAM utilities and practices used in the membrane module development are described in this work.

Chapter 2

Theoretical part

2.1 Ion exchange membranes

Ion exchange technology has been receiving growing attention in various industrial sectors for several decades. This technology is used to purify various solutions by removal of the dissolved ions using electrostatic sorption. Ions are absorbed into ion exchange materials which come in various physical shapes and forms. The absorbed ions are replaced by an equivalent amount of different ions with the same charge in the solution. The use of ion exchange membrane modules allows either a complete removal of all ions from a solution or a selective separation of a particular ions.

There are numerous application of ion exchange membranes ranging from industrial ones to households. Although they mainly cover purification purposes, their usage also covers extraction and separation of valuable substances. Another significant application is deionization of water and water softening.

The principle of ion exchange process is the exchange of ions between electrolyte solution in aqueous phase and ions that are immobilized in solid phase, the ion exchange material. A stoichiometric reversible ion exchange reaction takes place here. Mentioned ion exchange materials are the main part of the whole process. They fall into various categories as mineral and polymeric or anionic and cationic. [2]

There is a difference in the chemical potential across the ion exchange membrane. It can be caused by an applied current or by permeability difference. Only a certain types of ions are able to pass trough the membrane depending on their size, mobility and charge. The ion exchange membrane has one compartment on each of its sides. The dilute is the compartment the ions are being removed from while the concentrate is the compartment the ions are transported to. [3]

2.1.1 Donnan equilibrium

Donnan, or Gibbs-Donnan, equilibrium describes the equilibrium between two solutions separated by a membrane. The principle of the membrane is that is allows certain charged components in solution to pass trough. But the membrane does not allow all the ions present in solution to pass through which makes it a selectively permeable membrane.

The membrane selectivity is typically related to the particular ion size. The pores of the membrane can be too small to let an anion or cation pass through. The concentration of ions that can pass through the membrane should be the same on each side. The total number of ions should be equal on either side of the membrane as well.

A selective permeability membrane consequence is the formation of an electrical potential difference between the two membrane sides. The two solution also differ in osmotic pressure because one solution usually has more ions of a certain type than the other one. [4]

For compomemt i in a solution, the electro-chemical potential is defined

$$\eta_i = \mu_i^0 + R_g T ln(a_i) + (P - P^0)V_{mol,i} + z_i F \phi \tag{2.1}$$

where μ_i^0 is the standard chemical potential, R_g is the gas constant, T is the absolute temperature, a_i is the activity, P is the pressure, P^0 is the standard pressure, $V_{mol,i}$ is the partial molar volume, z_i is the charge number, F is the Faraday constant, and ϕ is the electric potential.

Equation 2.2 represents the Donnan equilibrium state. It implies that the electro-chemical potential of component i, η_i, in a solution and the electro-chemical potential of the same component in membrane are equivalent, $\bar{\eta}_i$. [5]

$$\eta_i = \bar{\eta}_i \tag{2.2}$$

2.1.2 Electric field diffusion

When an external driving forse is applied, diffusing particles experience drift motion in addition to standard diffusion. A most common example of an external force is an electric field.

The electrical conductivity is a result of the transport of ions rather than electrons for many ionic solids. This is different from metals or semiconductors.

When ions are the charge carriers in an electronically insulating material, the ionic motion under the influence of an electric field is described by the ionic conductivity. The dc (direct charge) conductivity, σ_{dc}, related with the electrical current density, j, and electric field strength, E, via Ohm's law. [6]

$$j = \sigma_{dc} E \tag{2.3}$$

2.1.3 Teorell, Meyer and Sievers Theory

Teorell, Meyer and Sievers Theory (TMS theory) can also be generally applicable in understanding the mechanisms of transport phenomena. This theory is based on the Nernst-Planck equation and the Donnan equilibrium theory. The TMS theory dicusses membrane phenomena in electrolyte solutions and reveals the mechanisms of characteristics such as diffusion coefficient, membrane potential, electric conductiviy, etc. The difference is that the ionic mobility and activity coefficient are taken as constants. Despite the differences, the results produced by the TMS theory are essentially equivalent to results produced by Donnan equilibrium theory. [5]

2.1.4 Electrochemical reactions

A closed circuit with an anode and a cathode is necessary to apply an electrical current to the system. The ions will then carry the charge across the system. Depending on the system setup or conditions the electrical potential varies. If the electrical potential is high enough, hydrogen and oxygen gases will be produced at the electrodes. [7]

Acetic acid is a weak acid and the dissociation in water will occur.

$$CH_3COOH_{(aq)} + H_2O_{(l)} \leftrightarrows H_3O^+_{(aq)} + CH_3COO^-_{(aq)} \tag{2.4}$$

On the surface of cathode, a reduction will take place. Hydrogen ions will receive electrons and form hydrogen in gaseous state. This half reaction is:

$$2H^+_{(aq)} + 2e^- \longrightarrow H_{2(g)} \tag{2.5}$$

The negatively charged remnant from an acetic acid molecule travels trough the anion exchange membrane to the other side of the membrane module. There, the second half reaction on the surface of the cathode occurs. This is an oxidation reaction where gaseous oxygen is formed after surplus electrons are taken away. The reaction is: [8]

$$2H_2O_{(l)} \longrightarrow O_{2(g)} + 4H^+_{(aq)} + 4e^- \tag{2.6}$$

The negatively charged remnant from acetic acid then reacts with the protone, forming acetic acid:

$$H^+_{(aq)} + CH_3COO^-_{(aq)} \longrightarrow CH_3COOH_{(aq)} \tag{2.7}$$

2.1.5 Water transport trough the membrane

Water transport across ion exchange membranes accompanies this process. This water migration occurs when a charged membrane under the influence of an electrical potential difference is used, and it has been termed electro-osmosis. It may consist of water transport corresponding to primary hydration of the ions or, also, as an additional quantity. The total water transport caused by the current is generally referred to as electro-osmosis. Water transport by osmosis is also a natural phenomenon in electrodialysis and the transport takes place in the same direction as the mass transport. Both electro-osmosis and osmosis are unavoidable side effects and they limit the usefulness of electrodialysis as a method of concetrating electrolyte solutions. The transport of ions with water trough a membrane is caused by pressure and osmotic forces. [9]

Two models of solver transport were distinguished:

- osmotic flow which may give rise to a streaming potential
- electro-osmotic flow which accompanies the migration of ions trough the membrane [10]

The main two methods for investigating water transport are based on weight and volume changes. The former one is suitable for flexible membranes and it requires meticulous care in transferring the solutions as well as rinsing the apparatus. The latter is much more easier and simpler. [9]

2.2 Equations used

All the CFD calculations were performed in the OpenFOAM software. The following subsection will discuss the applied equations in each part of the membrane module.

2.2.1 Velocity

For the computation of velocity profiles in the system, Navier - Stokes equations are implemented. Since the flow is considered to be a laminar flow of an incompressible liquid, the equation takes the following form

$$\frac{\partial u}{\partial t} + (u.\nabla)u - \nu\nabla^2 u = -\nabla\left(\frac{p}{\rho}\right) \tag{2.8}$$

where u stands for fluid velocity, t is time, ν is kinematic viscosity, p is pressure and ρ is fluid density.

For the same set of assumptions, the continuity equation yields

$$\nabla \cdot u = 0 \tag{2.9}$$

2.2.2 Pressure

Pressure is solved together with velocity. The standard method in multiple OpenFOAM solvers is the PISO (Pressure Implicit with Splitting of Operators) method. This algorithm is an efficient method for solving the Navier-Stokes equations in unsteady state cases. It roughly consists of solving the momentum, mass and pressure equation followed by correction of these calculated values.

The pressure equation can by solved multiple times if the given geometry has a higher degree of non-orthogonality.

2.2.3 Concentration

Ions, the dissolved species in the liquid flow trough the membrane module, move trough the module via multiple mechanisms which include convection and diffusion. The latter consists of standard diffusion in liquid and diffusion caused by the electric field.

This movement is described by equation 2.10, where the first element on the right side stands for convection, the second one for standard diffusion and the last one stands for diffusion caused by the effect of the electric field. [11]

$$\frac{\partial c}{\partial t} = -\nabla(uc) + \nabla^2(Dc) - \nabla \cdot \left(\frac{DceE}{kT}\right) \tag{2.10}$$

In equation 2.10, c is concetration, D is diffusion coefficient, k is Boltzmann constant, e is elementary charge and E is electric field strength.

2.2.4 Electric field

Since the membrane module is under the effect of an electric field, it has to be calculated. First, a resistance of a compartment is calculated according to equation 2.11 [12]

$$R = \frac{h}{A\Lambda c_d} \tag{2.11}$$

where R is Electrical resistance, h is height of a membrane module compartment, A is membrane surface area, Λ is molar conductivity and c_d is concentration in the dilute compartment.

Taking into account the the electric current in system is stable, the electric potential drop across a compartment will be calculated with equation 2.12.

$$\triangle V = RI \tag{2.12}$$

$\triangle V$ is the electric potential drop in compartment or membrane and I is electric current.

Lastly, the strength of an electric field, E is calculated from the potential drop in the given compartment and the compartment height. [13]

$$E = \frac{\triangle V}{h} \tag{2.13}$$

In the case of the membrane itself, the electrical resistance is not calculated but taken from the membrane module provider web page [14]. The electrical potential drop as well as the electrical field are then calculated in the same way as previously.

Chapter 3

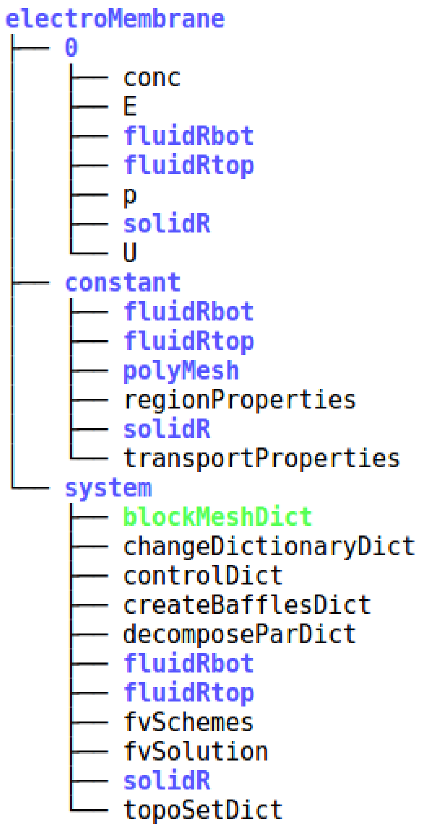
Practical part

3.1 OpenFOAM case file structure

There is a defined file structure in OpenFOAM software that needs to be followed in order to successfully run an application. The file structure of *electroMembrane* case is shown on figure 3.1. All the files can be also found in the appendix.

3.1.1 Constant directory

Constant directory contains full description of the case mesh in the *polyMesh* subdirectory. All the used constants are stored in the *transportProperties* file. The *regionProperties* file contains information on how to assign regions to categories. *FluidRbot* and *fluidRtop* are regions in fluid region category and *solidR* is a region in solid regions category. The dictionaries with the same names as the regions contain their own *polyMesh* dictionaries, where the geometry of each given region is stored. They also contain all the constants used in the separate regions.



```
electroMembrane
├── 0
│   ├── conc
│   ├── E
│   ├── fluidRbot
│   ├── fluidRtop
│   ├── p
│   ├── solidR
│   └── U
├── constant
│   ├── fluidRbot
│   ├── fluidRtop
│   ├── polyMesh
│   ├── regionProperties
│   ├── solidR
│   └── transportProperties
└── system
    ├── blockMeshDict
    ├── changeDictionaryDict
    ├── controlDict
    ├── createBafflesDict
    ├── decomposeParDict
    ├── fluidRbot
    ├── fluidRtop
    ├── fvSchemes
    ├── fvSolution
    ├── solidR
    └── topoSetDict
```

Figure 3.1: ElectroMembrane case file structure

3.1.2 System directory

The bare minimum of files that the *System* directory can contain is three and they are *controlDict*, *fvSchemes* and *fvSolution*. In the *controlDict* file, information that concern data output, run control as start time, end time or time step are stored. In the *fvSchemes* (finite volume schemes) file, discretisation schemes used in the solutions are selected. The *fvSolution* file contains a selection of solvers together with tolerances or other algorithm controls for all the used equations. All the remaining dictionaries store information on how each of the utilities, with the same name as the given dictionary, are set up to perform. Mentioned utilities are described in more detail in the following sections. Just as in the *Constant* directory, dictionaries sharing the names of separate regions contain same base files as the System directory apart from *controlDict*, as well as utility dictionaries used in each given region.

3.1.3 Time directories

Time directories contain individual data for particular fields for the given time. In the case of 0 time directory, initial values as well as boundary conditions describing the case are stored here. The rest of time directories contain results written to the files by OpenFOAM. Directories with specific region names contain 0 and different time directories as well. They too store initial values and boundary conditions as well as boundary condition between separate regions which describe how should a given field behave on a face between regions.

3.2 Mesh

In order to perform a simulation, a proper geometry has to be built. This is achieved by building a mesh, which is an integral part of a numerical solution. A mesh that satisfies certain criteria ensures a valid and accurate solution. In OpenFOAM, mesh is by default made of arbitrary polyhedral 3D cells.

3.2.1 CFD meshes

Fluid flow or heat transfer problems are generally governed by partial differential equations. Only very simple cases lead to analytical solutions and that is why in order to analyze a fluid flow, the geometry needs to be split into smaller domains. The mentioned equation are discretized and then solved in each of these smaller domains. Three typical methods used for solving these equations: finite volumes, finite elements and finite differences. Mesh generation is a process of obtaining an appropriate mesh and it has been long considered to be a bottleneck in the CFD analysis due to the lack of fully automatic mesh generation procedures.

3.2.2 Mesh components

A mesh is build by specifying points. A point represents a location in 3D space and it is defined by a vector in meters. The points are then compiled into a list, while each one is referred to by a numeric label that represents its position in the list. The numbering starts from zero.

The points are then ordered into faces, which are created by calling the point labels in a way that each two neighbouring points are connected by an edge. Faces can also be compiled into a list and are referred to by labels. There are two different types of a face. Internal faces connect exactly two faces and cannot hold boundary conditions. Boundary faces belong only to one cell, they coincide with the boundary of the domain. A boundary face is addressed by a boundary patch.

The faces are then built into cells. Cells must completely cover given computational domain and must not overlap each other. Also, every cell has to be closed. [15]

3.2.3 Mesh types

Most common 3D mesh cell shapes are tetrahedron and hexahedron. A tetrahedron consists of 4 vertices, 6 adges bound by 4 triangular faces. Tetrahedral meshes can be generated automatically in most cases but they do not provide the best accuracy of solutions. Hexahedron, or simply a brick, has 8 vertices and 12 edges bounded by 6 quadrilateral faces. The accuracy of a solution with a hexahedral mesh is the highest for the same cell amount.

Two types of mesh grids include structured and unstructured grids. Structured grid is identified by regular connectivity. This grid model is highly space efficient since the neighborhood cell relationships are defined by storage arrangement. Also, this type of grid provides better convergence and higher resolution. The unstructured grid is defined by irregular connectivity and compared to structured grid, it is highly space inefficient. [16]

3.2.4 Meshes used in this work

Two meshes were build for simulations of the membrane module process. Both of the meshes consist of main three blocks, two for the compartments and one for the membrane. The first mesh takes a simpler approach which means that the whole sides of the two compartment blocks are used as inlet and outlet. This can be seen on figure 3.2.

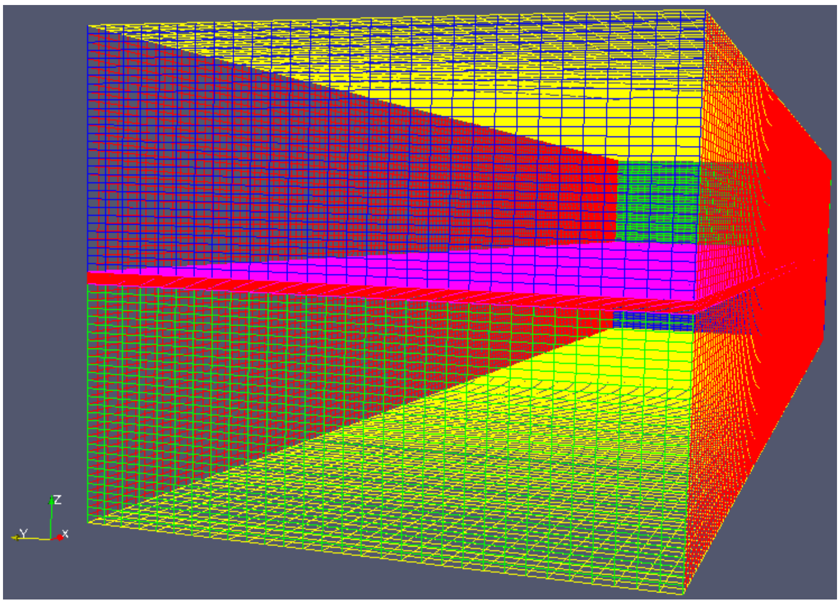


Figure 3.2: Simple version of membrane module mesh

The second mesh is more detailed. It takes into account, that inlets and outlets of the compartments are circular holes and not the whole faces as can be seen on figure 3.3. This means that there is a cylinder along the whole compartment, connecting the circle inlet and outlet. Naturally, the geometry around this cylinder has to be adapted to the shape. This brings a good deal of nonorthogonality to the mesh.

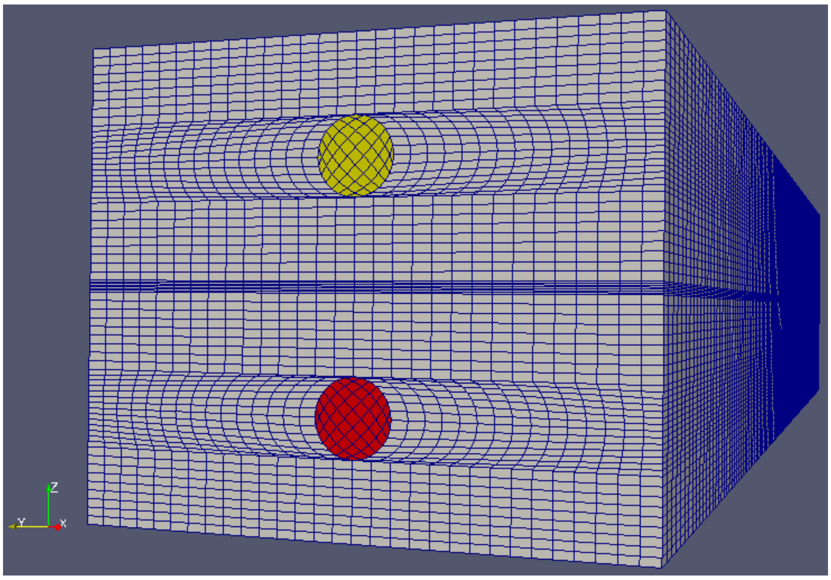


Figure 3.3: More detailed version of membrane module mesh

In both mesh cases, the same dimensions were used. A schematic membrane module with dimensions can be seem on figure 3.4.

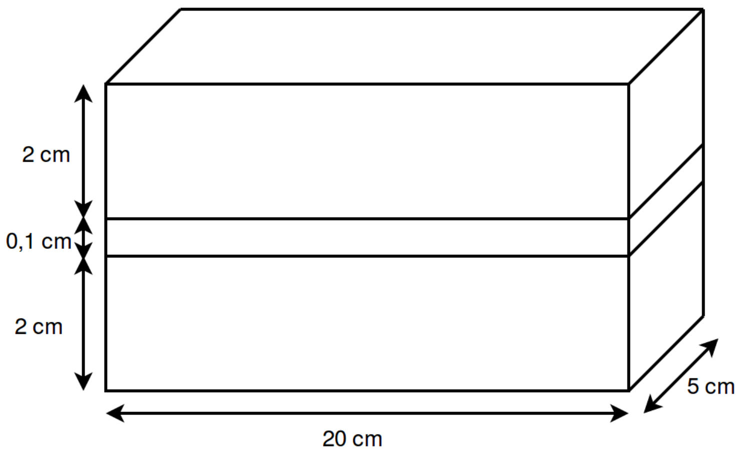


Figure 3.4: Drawing of the membrane module with dimensions

3.2.5 Courant number

Courant number provides a measure for the convergence. It is used in solving partial differential equations numerically. The condition states that given a certain space discretization, a time step should not be bigger than some a certain time, given by the equation 3.1. The typical C_{max} value is 1. [17]

$$C = \frac{u \bigtriangleup t}{\bigtriangleup x} \leq C_{max} \tag{3.1}$$

In equation 3.1, C is Courant number, u is fluid velocity, $\bigtriangleup\ t$ is time step and $\bigtriangleup\ x$ is spacing of the grid.

3.3 Pre-processing

Pre-processing consists of generating and modifying the mesh as well as fields and boundary conditions so the simulation may run. The whole pre-processing step along with explanation of used utilities is discussed in the following sections.

3.3.1 blockMesh

After all the points, edges, faces and blocks are declared in the appropriate dictionary (in this case *blockMeshDict*), the *blockMesh* utility is called. *BlockMesh* is a mesh generation utility used for creating parametric meshes with curved edges and grading. It decomposes the geometry of the domain into a set of 3D, hexahedral blocks. Each of these blocks is defined by 8 points, one at each corner of given block. The blocks can also have a label assigned, if desired.

3.3.2 topoSet

The mesh, created with *blockMesh* utility, contains two internal faces. One on each side of the membrane, connecting it to the dilute and concentrate compartments. It is desired that the internal faces would be converted into boundary faces because boundary faces can have various boundary conditions assigned. These two faces represent the contact of the membrane with the compartments.

Firstly, a set of faces has to be created using *topoSet* utility using the *boxToFace* source. All the specifications are given in the given dictionary, *topoSetDict*. The utility then takes every internal face that has its center inside the given box and assigns it into a *faceSet* with a specified name. The next step is converting the mentioned $faceSet$ into a labeled $faceZoneSet$. The created $faceZoneSet$, which contains the internal faces of the mesh, can be later used by other utilities.

3.3.3 splitMeshRegions

The next step is splitting the geometry into multiple regions. In this case, three regions are created, two for the compartments and one for the membrane. This is achieved by $splitMeshRegions$ utility, which is used with $-cellZones$ and $-overwrite$ options. The $-cellZones$ option uses previously created $faceZoneSets$ to created desired regions.

3.3.4 createBaffles

Then, the internal faces contained in the $faceZoneSet$ are converted into boundary ones. Internal faces are the ones connecting different regions in the geometry. This is achieved with $createBaffles$ utility. The utility will convert internal faces into boundary faces of types specified in the $createBafflesDict$ dictionary. The specific boundary condition types are discussed more in the following sections.

3.3.5 changeDictionary

After using utilities mentioned above, the boundary conditions are set to default and have to be changes to the desired ones. This is accomplished by the *changeDictionary* utility which simply edits the appropriate dictionaries and changes default boundary conditions to the specified ones.

3.3.6 decomposePar

There is an option to run OpenFOAM applications in parallel on distributed processors. This means that the geometry is broken down into pieces which are then assigned to separate processors for solving. The geometry is broken down according a set of parameters that are specified in the *decomposeParDict*. This includes the specification of how the geometry should be broken down (x,y and z axis) as well as method of decomposition.

3.3.7 Other approaches

There are many other ways how to approach a geometry of any given OpenFOAM case. One would be the usage of *snappyHexMesh* utility. This utility creates a mesh by approximate conformations to the surface. It chooses a starting mesh and then iteratively refines it to fit the given geometry.

Another option would be to use a different software to create a geometry and then convert it to an OpenFOAM format. A commonly used software includes Fluent, Salome or Gambit.

3.4 Boundary conditions

The boundary conditions play a very important role in every OpenFOAM application. The role of a boundary condition is not only as a geometric entity but also as integral part of the solution. The boundary faces also represent a "connection" between different regions. A list of boundary conditions on specific patches used in this word is presented in tables 3.1, 3.2 and 3.3. These boundary conditions are discussed in the following sections.

	p	conc	U	E
Inlet	zeroGradient	fixedValue	fixedValue	fixedValue
Outlet	fixedValue	zeroGradient	zeroGradient	fixedValue
Walls	zeroGradient	zeroGradient	noSlip	fixedValue
Cathode	zeroGradient	zeroGradient	noSlip	fixedValue

Table 3.1: Boundary conditions for the fluidRbot region

	p	conc	U	E
fluidRbot-solidR	zeroGradient	zeroGradient	noSlip	fixedValue
solidR-fluidRbot	zeroGradient	mappedFixedInternalValue	-	fixedValue
solidR-fluidRtop	zeroGradient	zeroGradient	-	fixedValue
fluidRtop-solidR	zeroGradient	mappedFixedInternalValue	noSlip	fixedValue
walls	zeroGradient	zeroGradient	-	fixedValue

Table 3.2: Boundary conditions for the solidR region

	p	conc	U	E
Inlet2	zeroGradient	outletMappedUniformInlet	fixedValue	fixedValue
Outlet2	fixedValue	zeroGradient	zeroGradient	fixedValue
Walls	zeroGradient	zeroGradient	noSlip	fixedValue
Cathode	zeroGradient	zeroGradient	noSlip	fixedValue

Table 3.3: Boundary conditions for the fluidRtop region

3.4.1 fixedValue

This boundary condition is one of the standard ones. As the name suggests it simply supplies a fixed value to the given boundary face. It is required to provide the value in appropriate, scalar or vector, form.

3.4.2 zeroGradient

$ZeroGradient$ boundary condition is a OpenFOAM version of a Newmann condition, where the value of the derivate is given. In this case, the value is zero. This means that a point of a boundary face with this condition will be given the same value as the last point in geometry before the boundary face.

3.4.3 noSlip

$NoSlip$ is a boundary condition used with fluid velocity. It is a common type of idealized boundary condition found in the applications of fluid dynamics. The fluid is flowing along an impenetrable wall. The overall boundary for viscous fluid flowing along this impenetrable wall is that there is no motion between the wall and the fluid that is in immediate contact with the wall. [18]

3.4.4 mappedFixedInternalValue

Boundary condition $mappedFixedInternalValue$ maps the boundary values of a neighbor patch field to the boundary values of desired patch field. This boundary condition enables calculated fields to move from one region to another.

3.4.5 outletMappedUniformInlet

OutletMappedUniformInlet boundary condition averages the field values of the specified outlet patch and applies it as an uniform value to the field over this patch. This boundary in used with the inlet patch of the concentrate compartment. The reasoning behind this is that after concentrate flow leaves the membrane module, it will get perfectly mixed in the concentrate tank before entering the module again.

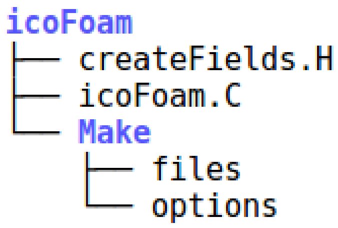
3.5 Solver

Solver with a name *membraneSolver* was used to preform the simulation. Custom solver in OpenFOAM are usually developed based on one of the default solvers. The same procedure was used in this work. *MembraneSolver* is based on a solver named *icoFoam*.

3.5.1 icoFoam

This is a transient solver for incompressible laminar flow of Newtonian fluids. The code requires an initial conditions and boundary conditions. The *icoFoam* solver can take mesh non-orthogonality into account with a number of non-orthogonality iterations. The number of PISO corrections and non-orthogonality corrections are controlled through user input.

OpenFOAM applications are organized using a standard convention. The source code of each application is placed in a directory that shares name with the application. The source code of *icoFoam* solver resides in a directory *icoFoam* as is shown on figure 3.5. The top level file also shares the name, *icoFoam.C*.



```
icoFoam
├── createFields.H
├── icoFoam.C
└── Make
    ├── files
    └── options
```

Figure 3.5: IcoFoam solver file structure

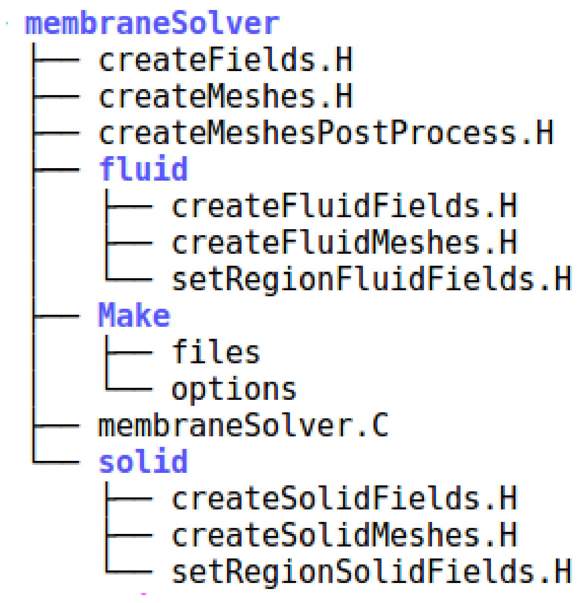
The *createFields.H* file consists all the information on created and used fields. Here, the constants, scalar and vector fields have to be stated via *IOobject* (input output object) classes. The information concerning reading and writing of the fields is also stated in this file.

An OpenFOAM solver also has to contain a Make directory in its main directory. There are files named *files* and *options* in this directory. The *files* file contains information on where the solver executables should be written. The *options* file contains the full directory paths to locate other files used in the solver.

3.5.2 membraneSolver

As was mentioned previously, *membraneSolver* solver is based on the *icoFoam* solver. That means that the pressure and velocity calculations are handled the same way, with the PISO algorithm. This solver also calculates the concentration of the dissolved ionic species as well as handles multiple regions of the geometry. Concentration is calculated as it is stated in the theory part of this work.

As multiple regions are used in *membraneSolver*, the file structure of this solver is a bit more complex than the file structure of *icoFoam* solver. This can be seen on figure 3.6.



```
membraneSolver
├── createFields.H
├── createMeshes.H
├── createMeshesPostProcess.H
├── fluid
│   ├── createFluidFields.H
│   ├── createFluidMeshes.H
│   └── setRegionFluidFields.H
├── Make
│   ├── files
│   └── options
├── membraneSolver.C
└── solid
    ├── createSolidFields.H
    ├── createSolidMeshes.H
    └── setRegionSolidFields.H
```

Figure 3.6: MembraneSolver solver file structure

All the regions are sorted into two categories, fluid and solid regions, each category has a dictionary with the same name. The fluid dictionary contains *createFluidFields.H*, *createFluidMeshes.H* and *setRegionFluidFields.H* files.

The workflow of this solver is as it follows. Firstly, the meshes are created. This is handled by *createFluidMeshes.H* file. The program will retrieve a list of all region names in the fluid region category. Then, a mesh of an appropriate name is created for each of the regions. Secondly, all fields are created with the contents of *createFluidFields.H* file. Here, a pointer list ,with the same size as given field, is assigned to each of the computed fields. The next step is a creation of *IOobjects* for each field in each region. Lastly, with the contents of *setRegionFluidFields.H* file, the values in the pointer lists are allocated into the appropriate fields. This means that a given computed fields will have the same label in every region of both region categories.

3.6 Post-processing

3.6.1 Mesh reconstruction

Cases that have been run in parallel can either be reconstructed or each of the geometry segments can be processed individually. Here, the former option was used. After all the calculations in parallel are finished, the mesh is rebuilt together again. This is achieved by *reconstructPa*r utility, which merges the time directories from each processor into single set of time directories.

3.6.2 paraFoam

ParaFoam is the main post-processing tool provided with OpenFOAM. It is an open-source, visualization application. This tool was used to create all visualizations of the geometry and meshes as well as graphs.

Another alternative is conversion of data into VTK format so it can be read not only by paraFoam but any VTK-based graphic tool.

Chapter 4

Results and discussion

The parameters of simulations can be seen in Table 4.1. The electric strength values were calculated outside of the model and set as a constant value.

Parameter	Value	Parameter	Value
c_d	10 mol.m^{-3}	ρ	998,2 kg.m^{-3}
T	293.15 K	D_c	$1,089.10^{-9}$m^2.s^{-1} [19]
E_c	61.125 V.m^{-1}	D_{m1}	$1,089.10^{-9}$m^2.s^{-1}
E_m	2500 V.m^{-1}	D_{m2}	$1,089.10^{-10}$m^2.s^{-1}
ν	$1,004.10^{-6}$m^2.s^{-1}	D_{m3}	$1,089.10^{-11}$m^2.s^{-1}
u	0.04 m.s^{-1}	e	$1,6022.10^{-19}$C
k	$1,3806.10^{-23}$m^2.kg.s^{-2}.K^{-1}	Λ_{aa}	$40,9.10^{-4}$S.m^2.mol^{-1} [19]
I	0,05 A	V_{anode}	5 V

Table 4.1: Input parameters sued in simulations

4.1 Simulation results

The diffusion coefficient of acetic anions in the membrane was not provided by the manufacturer. Multiple simulation were performed with different values of this diffusion coefficient. In first case, value of diffusion coefficient of acetic anions in membrane D_{m1}, was the same as the diffusion coefficient of acetic anions in water D_c. In the second and third simulations, D_{m2} and D_{m3} are 10 and 100 times smaller than D_c.

4.1.1 Pressure

The pressure profile of a cross section of the membrane module can be seen on figure 4.1. The values of p in the simulations represent pressure divided by fluid density. In both lower (dilute) and upper (concentrate) compartments, a pressure decrease is formed. A slight decrease of pressure is to be expected from flowing fluids.

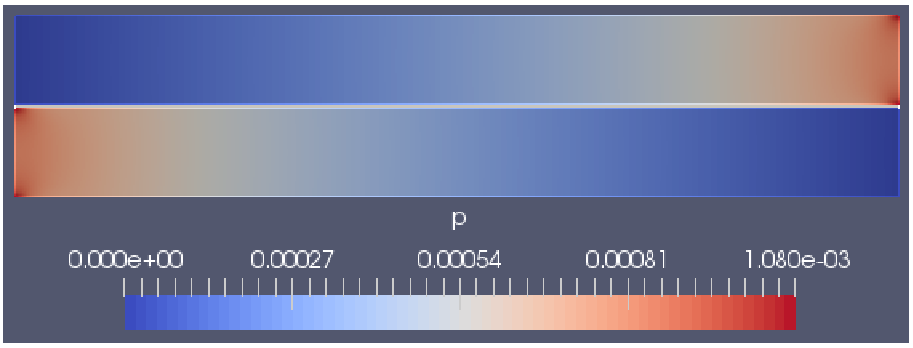


Figure 4.1: Cross section of the pressure profiles in the mesh with dilute compartment at the bottom and concentrate compartment at the top

The value of pressure was not set in the simulation. Instead, the inlet value was set as zero gradient and the outlet value as zero. Zero pressure represents a reference pressure, which can be for example standard pressure. Because inlet velocity is set, the inlet pressure calculation is based on that value.

The same pressure drop for dilute compartment as on figure 4.1 but in terms of plot is shown on figure 4.2. Here, the x represents the distance on x axis from the dilute compartment inlet.

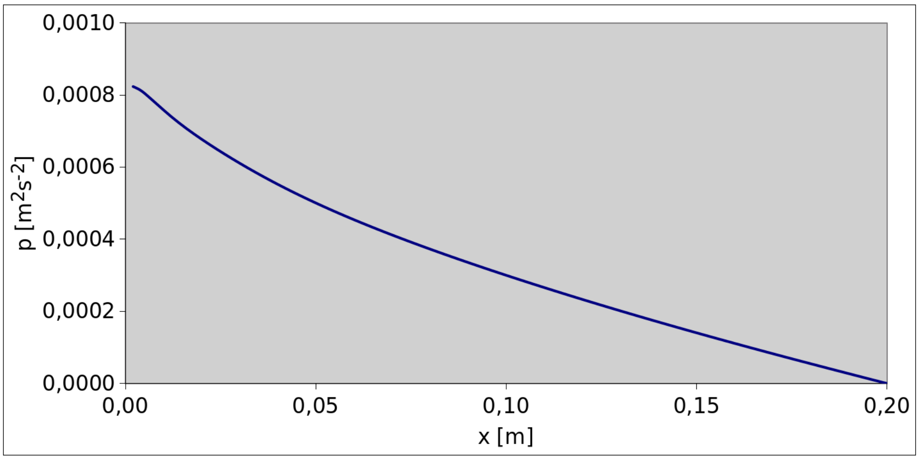


Figure 4.2: Pressure profile of dilute compartment

4.1.2 Velocity

Cross section trough the velocity profiles in the membrane module is shown on figure 4.3. In both dilute and concentrate compartment inlets, the velocity profile in not formed as the inlets are set to have uniform velocity distribution. It can be seen that the velocity near walls is zero which agrees to wall boundary conditions set for the simulation.

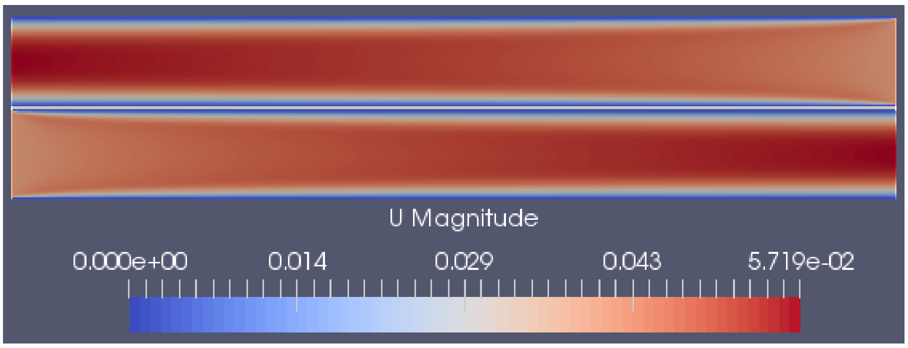


Figure 4.3: Cross section of the fluid velocity profiles in the mesh with dilute compartment at the bottom and concentrate compartment at the top

The figure 4.4 shows velocity profiles at dilute compartment at more detail. The lines represent cuts of the compartment in the direction of z axis. The cuts are made on three different positions on the x axis.

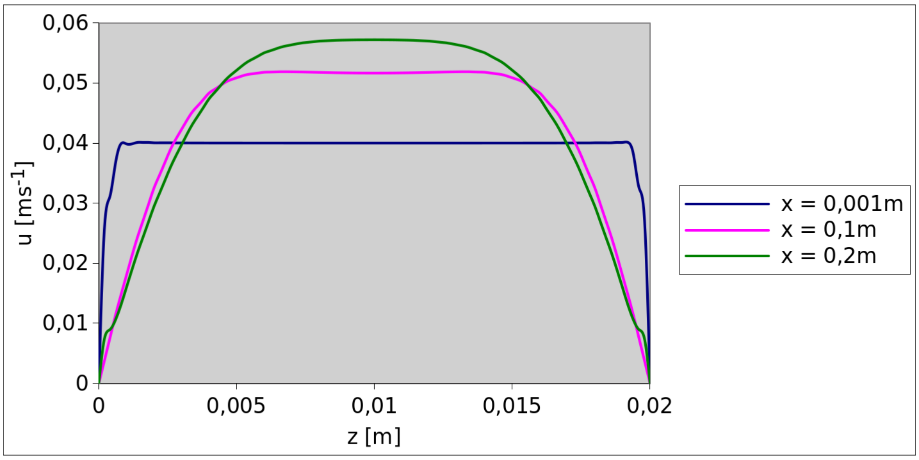


Figure 4.4: Velocity profiles at different locations on the compartment

It can be clearly seen that laminar flow regime is being formed. A fully formed laminar flow regime would likely form if the membrane module would be longer.

4.1.3 Concentration

The membrane concentration profiles are shown on figures 4.5, 4.6 and 4.7. The 0 value on the z axis represent contact with the dilute compartment, hence the concentration 10 mol.m^{-3}. The other end of the z axis represents contact with the concentrate compartment. In all three cases, data in 7 different times were taken.

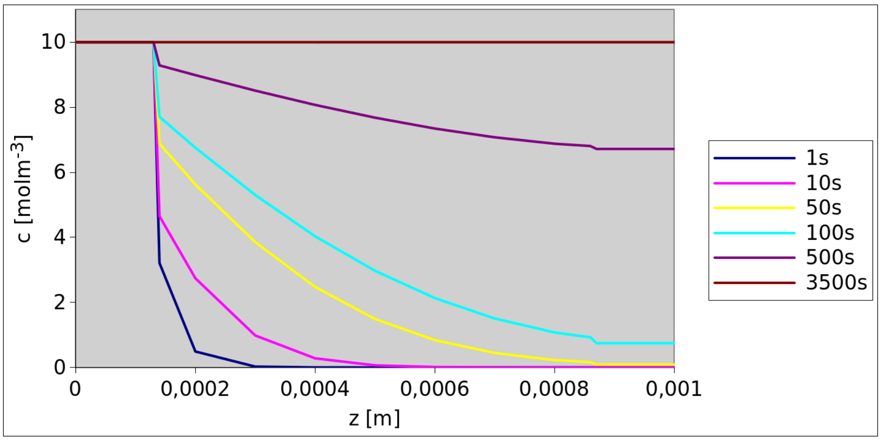


Figure 4.5: Membrane concetration profile using D_{m1}

It takes less than an hour for the membrane in the first simulation to have the same concentration of acetic ions as the dilute compartment. That is not the case in the second and third simulations. The real value of D_m is most likely between the three used values. This statement is based on diffusion coefficient of other ions in water and ion exchange membranes. The value of mentioned diffusion coefficient in ion exchange membrane is usually up to 100 times lower than in water [20] [21]. A diffusion coefficient for a specific membrane can be calculated from the correlation of current efficiency based on experiments [12].

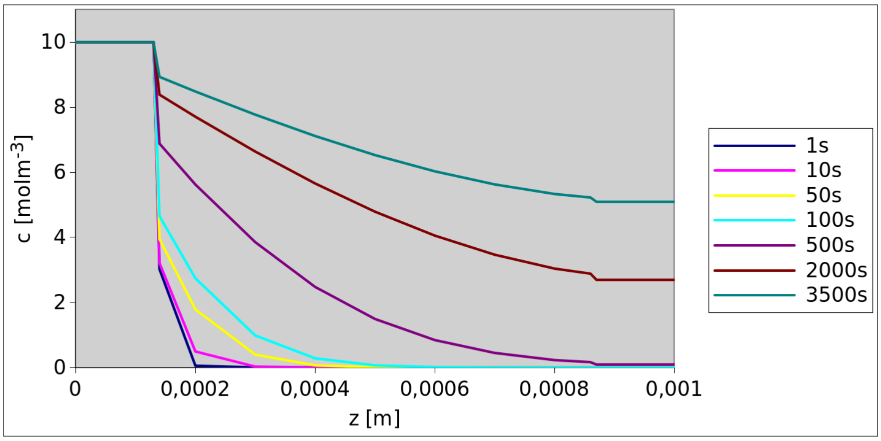


Figure 4.6: Membrane concetration profile using D_{m2}

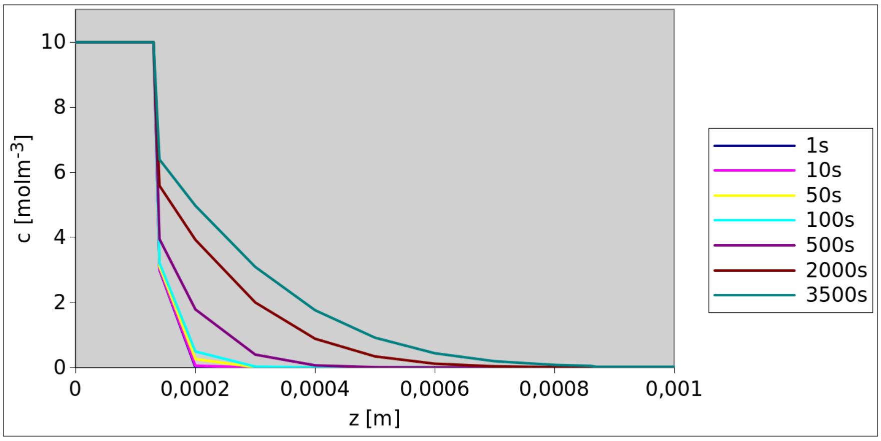


Figure 4.7: Membrane concetration profile using D_{m3}

4.2 Future development

There is a room for significant improvement of this model since many simplifications and conditions were assumed.

4.2.1 Geometry

Two geometry approaches were presented, a simple and a more detailed one. In the case of the simple model, the x, y and z axis point spacing is different. Although this can be seen in the membrane as well as in the compartments, it is more prominent in the membrane. Since the membrane is very thin, the distance between two points in the z axis direction is 0.1 mm whereas the distances between two points in the x a y axis directions are 1.33 mm and 1.67 mm. Since the mesh grid is structured, these distances in between the points in membrane cannot be decreased without doing the same in the compartments. For this reason, the decrease of the distance between two points in the direction of x or y axis would significantly prolong the computation time.

In the case of more detailed geometry, the fields were not calculated properly. As was described in the previous sections, the mesh is not orthogonal. This can have effect on the precision of the calculations. Since inlet and outlet are circles in the middle of rectangular walls, it is possible that the fluid creates swirls when flowing trough. For this reason, a laminar flow regime may not be optimal and simulations with more turbulent flow regime could be performed.

4.2.2 Other acids and ions

Only acetic anions were taken into account in this model. Possible broadening of this models capabilities would be to accompany the electrochemical reactions.

This would mean calculations of other ion concentrations. The model could also the into account multiple acids dissociation, which would also mean more complex calculations of necessary diffusion coefficients.

4.2.3 Chemical reactions

Another matter to include could be generation of hydrogen and oxygen gasses on the electrodes, which would requite addition of multiphase layers and calculations and also geometry modification to include the gas outlets. A different approach would be to assume different set of reactions, meaning a different chemicals than hydrogen and oxygen would be produced.

4.2.4 Electric potential

A simplified approach to the calculation of electric field strength was taken. The values were calculated outside of the model and set as a constant value. A more sophisticated approach would be to calculate electric potential and electric field strength in the model. This could also take into account changes in the solution conductivity with different concentrations of ions.

Chapter 5

Conclusion

A simple CFD model for simulating the operation of an ion exchange membrane was created in this work. The geometry consists of a dilute and a concentrate compartment separated by an ion exchange membrane. Two different meshes were created, a simplified one and a more precise one. The simplified one was used in the simulations. A number of pre-process utilities used in this work are described along with applied boundary conditions. Then, a creation of used solver is described and results are shown. This model is able to calculate pressure, fluid velocity and acetic anion concentration fields. The model also takes into account the effect of an electric field. There is also room for future development. The more precise mesh could be reworked, since it did not produce accurate results. Other ions as well as other dissociating acids could also be taken into account. This would mean implementation of electrochemical reactions on the electrodes. If production of hydrogen and oxygen gasses is assumed, multiple phases along with multiphase layers have to be calculated. Lastly, electric potential could be calculated in more sophisticated way, taking into account current concentration of ions in solutions.

Chapter 6

Resumé

Cieľom predloženej diplomovej práce bolo získať zakladné znalosti o CFD modeloch a simuláciach. To zahŕňa aj znalosti o vytváraní a prácou s mriežkami. Následne bolo potrebné použiť získané vedomosti na návrh modelu membránového modulu. Tento model obsahuje iónovú membránu, ktorá oddeluje dve nádoby a slúži na separáciu zmesi organických kyselín. Model membránového modulu pracuje s kyselinou octovou. Konkrétne sa jedná o iónovú membránu, cez ktorú difundujú anióny kyseliny octovej. Membránový modul sa skladá z dvoch nádob oddelených membránou. Každá z nádov má vstup a výstup a usporiadanie prúdov je protiprúdne. V práci boli použité nasledujúce zjednodušenia. Aj keď sa v skutočnosti v systéme nachádza viacero kyselín, model pracuje iba s aniónmi kyseliny octovej. V celom systéme sa predpokladá laminárny režim prúdenia a konštantná teplota.

Mriežka je neoddelitelnou súčasťou CFD modelovania. Je možné ju vytvoriť priamo v programe OpenFOAM funkciou *blockMesh*. Druhá možnosť je importovať už hotovú mriežku z iných programov na vytváranie 3D geometrií. V tejto práci boli vytvorené dve mriežky. Obe sa skladajú z veľkého počtu malých šesťstranných elementov a obe sú štruktúrované. Prvá, zjednodušená varianta zanedbáva kruhové vstupy a výstupy z nádob, za vstupy a výstupy sa považujú celé steny ako je znázornené na obrázku 3.2. Zložitejšia mriežka presne opisuje spomínané kruhové vstupy a výstupy a je znázornená na obrázku 3.3. Simulácie so zložitejšou verziou mriežky však vykazovaly chybné výsledky, a preto bola v simuláciách použitá jednoduchšia mriežka.

Pred samotným výpočtom je potrebné model pripraviť použitím viacerých funkcií. Prvá je už spomínaná funkcia $blockMesh$, ktorá zostaví mriežku. Nasleduje funkcia $topoSet$, ktorá v mriežke vytvorí tri oblasti, dve pre nádoby a jednu pre membránu. Funkcia $splitMeshRegions$ potom na základe vytvorených oblastí vytvorí tri regióny. Regióny sú nezávislé časti v mriežke, v jednotlivých regiónoch môžu byť počítané rôzne rovnice alebo zadané iné okrajové podmienky. Steny medzi regiónymi je vhodné pretvoriť na okrajové steny použitím funkcie $createBaffles$. To umožní priradenie okrajových podmienok k týmto stenám. Okrajové podmienky na stenách medzi regiónmy sú podstatné, pretože určujú správanie sa jednotlivých počítaných polí. Napr. z pohľadu rýchlosti tekutiny a tlaku je rozhranie nádoby a membrány brané ako nepriechodná stena, no z hľadiska koncentrácie preieha difúzia častíc do membrány. Konkrétne použitie okrajových podmienok je uvedené v tabuľkách 3.1, 3.2 and 3.3.

Nasleduje samotný výpočet. Solver s názvom $membraneSolver$ bol vytvorený na základe solvera $icoFoam$. Ten rieši neustálené laminárne prúdenie nestlačiteľných Newtonovských tekutín. $MembraneSolver$ je rozšírený o výpočet koncentrácie a tiež pracuje s viacerými regiónmi. Intenzita elektrického poľa je vypočítaná mimo programu a dosadená ako konštantná hodnota.

Výsledky prezentujú jednotlivé vypočítané polia, a to tlak, rýchlosť a koncentráciu. V prípade tlaku sa naprieč modulom vytvoril jemný klesajúci gradient. Pole rýchlosti vykazuje postupné vytvorenie laminárneho režimu prúdenia. Boli uskutočnené tri simulácie s rôznymi hodnotami difúzneho koeficienta pre prestup octových aniónov v membráne. Presná hodnota tohto koeicienta nebola od výrobcu membŕany poskytnutá. Ako bolo očakávané, prestup látky membránou bol pomalší s nižšími hodnotami spomínaného difúzneho koeficienta. Tieto výsledky prezentujú funkčný jednoduchý model chodu membránového modulu.

Vytvorený model má veľký potenciál vývoja v budúcnosti. V prvom rade by bolo možné prerobiť zložitejšiu mriežku modulu tak, aby vykazovala správne výsledky. Takisto by bolo možné model rozšíriť o koncentrácie viacerých iónov, prípadne viacerých kyselín. To by znamenalo zakomponovať do modelu aj elektrochemické reakcie. Model by taktiež mohol obsahovať produkciu plynného vodíka a kyslíka.

Bibliography

[1] W. Malalasekara H. K. Versteeg. *An Introduction to Computational Fluid Dynamics*. Pearson Education, 2007. ISBN 978-0-13-127498-3.

[2] *Ion exchange membranes*. 1991.

[3] C. Joslin G. Stell. The donnan equilibrium. *Biophysical Journal*, 1986.

[4] The Gale Group Inc. Donnan equilibrium. *World of Microbiology and Immunology*, 2003.

[5] Yoshinobu Tanaka. *Ion Exchange Membranes: Fundamentals and Applications*. Elsevier, 2015. ISBN 978-0-444-63319-4.

[6] H. Mehrer. Diffusion in solids: Fundamentals, methods, materials, diffusion-controlled processes. *Springer series in solid state science*, 2007.

[7] Hennebel T. Gildemyn S. Coma M. Desloover J. Berton J. Tsukamoto J. Stevens C. Andersen, S. J. and K. Rabaey. Electrolytic membrane extraction enables production of fine chemicals from biorefinery sidestreams. *Enviromental Science and Technology*, 2014.

[8] Jurgen Mergel Detlef Stolten Marcelo Carmo, David L. Fritz. A comprehensive review on pem water electrolysis. *International Journal of Hydrogen Energy*, 2013.

[9] N. Krishnaswamy V. K. Indusekhar. Water transport studies on interpolymer ion-exchange membranes. 1984.

[10] P. W. M. Jacobs G. J. Hills and N. Lakshiminarayaniah. 262a. *Proc. Roy. Soc.*, 1961.

[11] P. Shewmon. Diffusion in solids. *The Minerals, Metals and Materials Society*, 1989.

[12] N. Aziz F.S. Rohman, M.R. Othman. Modeling of batch electrodialysis for hydrochloric acid recovery. *Chemical Engineering Journal*, 2010.

[13] C. H. Colwell. Physics lab online. *Electric Fields: Parallel Plates*, 2017.

[14] Fumatech. Functional membranen und anlagentechnologie. *fumatech.com.*

[15] Unofficial openfoam wiki. *Information about the OpenSource CFD toolbox OpenFOAM, openfoamwiki.net*, 2016.

[16] P. Plassmann M. Bern. Mesh generation. *Handbook of Computational Geometry, Elsevier Science*, 2000.

[17] H. Lewy R. Courant, K. Friedrichs. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 1967.

[18] Michael A. Day. The no-slip condition of fluid dynamics. *Kluwer Academic Publishers*, 1990.

[19] Petr Vanýsek. Ionic conductivity and diffusion at infinite dilution. *Handbook of Chemistry and Physics, CRC Press*, 1993.

[20] H. Miyoshi. Diffusion coeffcients of ions through ion excange membrane in donnan dialysis using ions of different valence. *Journal of Membrane Science*, 1997.

[21] J. Langmaier Z. Samec, A.Trojánek. Diffusion coeffcients of alkai metal cations in nafion from ion-exchange measurements. *The Electrochemical Society*, 1997.

Appendix A

Solver and case code

membraneSolver

fluid

createFluidFields.H

```
PtrList<uniformDimensionedScalarField> epsilon0Fluid(fluidRegions.size());
PtrList<uniformDimensionedScalarField> nuFluid(fluidRegions.size());
PtrList<uniformDimensionedScalarField> DFluid(fluidRegions.size());
PtrList<uniformDimensionedScalarField> zFluid(fluidRegions.size());
PtrList<uniformDimensionedScalarField> boltzFluid(fluidRegions.size());
PtrList<uniformDimensionedScalarField> TFluid(fluidRegions.size());
PtrList<volScalarField> pFluid(fluidRegions.size());
PtrList<volVectorField> UFluid(fluidRegions.size());
PtrList<volScalarField> concFluid(fluidRegions.size());
PtrList<volVectorField> EFluid(fluidRegions.size());
PtrList<surfaceScalarField> phiFluid(fluidRegions.size());
// Populate fluid field pointer lists
forAll(fluidRegions, i)
{
Info<< "*** Reading fluid mesh properties for region "
    << fluidRegions[i].name() << nl << endl;
Info<< "    Adding to epsilon0\n" << endl;
     epsilon0Fluid.set
     (
          i,
          new uniformDimensionedScalarField
          (
               IOobject
               (
                    "epsilon0",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to nu\n" << endl;
    nuFluid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "nu",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to D\n" << endl;
    DFluid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "D",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to z\n" << endl;
    zFluid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "z",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to boltz\n" << endl;
    boltzFluid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "boltz",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to T\n" << endl;
    TFluid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "T",
                runTime.constant(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to p\n" << endl;
    pFluid.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "p",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            fluidRegions[i]
        )
    );
Info<< "    Adding to U\n" << endl;
    UFluid.set
    (
        i,
        new volVectorField
        (
            IOobject
            (
                "U",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            fluidRegions[i]
        )
    );
```

```
Info<< "    Adding to conc\n" << endl;
    concFluid.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "conc",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            fluidRegions[i]
        )
    );
Info<< "    Adding to E\n" << endl;
    EFluid.set
    (
        i,
        new volVectorField
        (
            IOobject
            (
                "E",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
           fluidRegions[i]
        )
    );
Info<< "    Adding to phi\n" << endl;
    phiFluid.set
    (
        i,
        new surfaceScalarField
        (
            IOobject
            (
                "phi",
                runTime.timeName(),
                fluidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
           fvc::flux(UFluid[i])
        )
    );
label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell(pFluid[i], fluidRegions[i].solutionDict().subDict("PISO"), pRefCell, pRefValue);
fluidRegions[i].setFluxRequired(pFluid[i].name());
}
```

createFluidMeshes.H

```
    const wordList fluidNames(rp["fluid"]);
    PtrList<fvMesh> fluidRegions(fluidNames.size());
    forAll(fluidNames, i)
    {
        Info<< "Create fluid mesh for region " << fluidNames[i]
            << " for time = " << runTime.timeName() << nl << endl;
        fluidRegions.set
        (
            i,
            new fvMesh
            (
                IOobject
                (
                    fluidNames[i],
                    runTime.timeName(),
                    runTime,
                    IOobject::MUST_READ
                )
            )
        );
    }
```

setRegionFluidFields.H

```
    const fvMesh& mesh = fluidRegions[i];
    uniformDimensionedScalarField& epsilon0 = epsilon0Fluid[i];
    uniformDimensionedScalarField& nu = nuFluid[i];
    uniformDimensionedScalarField& D = DFluid[i];
    uniformDimensionedScalarField& z = zFluid[i];
    uniformDimensionedScalarField& boltz = boltzFluid[i];
    uniformDimensionedScalarField& T = TFluid[i];
    volScalarField& p = pFluid[i];
    volVectorField& U = UFluid[i];
    volScalarField& conc = concFluid[i];
    volVectorField& E = EFluid[i];
    surfaceScalarField& phi = phiFluid[i];
label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell(p, fluidRegions[i].solutionDict().subDict("PISO"), pRefCell, pRefValue);
fluidRegions[i].setFluxRequired(p.name());
```

solid

createSolidFields.H

```
PtrList<uniformDimensionedScalarField> epsilon0Solid(solidRegions.size());
PtrList<uniformDimensionedScalarField> DSolid(solidRegions.size());
PtrList<uniformDimensionedScalarField> zSolid(solidRegions.size());
PtrList<uniformDimensionedScalarField> boltzSolid(solidRegions.size());
PtrList<uniformDimensionedScalarField> TSolid(solidRegions.size());
PtrList<volScalarField> concSolid(solidRegions.size());
PtrList<volVectorField> ESolid(solidRegions.size());
```

```
// Populate solid field pointer lists
forAll(solidRegions, i)
{
Info<< "*** Reading solid mesh properties for region "
    << solidRegions[i].name() << nl << endl;
Info<< "    Adding to epsilon0\n" << endl;
    epsilon0Solid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "epsilon0",
                runTime.constant(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to D\n" << endl;
    DSolid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "D",
                runTime.constant(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to z\n" << endl;
    zSolid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "z",
                runTime.constant(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to boltz\n" << endl;
    boltzSolid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "boltz",
                runTime.constant(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
Info<< "    Adding to T\n" << endl;
    TSolid.set
    (
        i,
        new uniformDimensionedScalarField
        (
            IOobject
            (
                "T",
                runTime.constant(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        )
    );
   Info<< "    Adding to conc\n" << endl;
    concSolid.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "conc",
                runTime.timeName(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            solidRegions[i]
        )
    );
   Info<< "    Adding to E\n" << endl;
    ESolid.set
    (
        i,
        new volVectorField
        (
            IOobject
            (
                "E",
                runTime.timeName(),
                solidRegions[i],
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
           solidRegions[i]
        )
    );
}
```

createSolidMeshes.H

```
const wordList solidsNames(rp["solid"]);
PtrList<fvMesh> solidRegions(solidsNames.size());
forAll(solidsNames, i)
{
    Info<< "Create solid mesh for region " << solidsNames[i]
        << " for time = " << runTime.timeName() << nl << endl;
    solidRegions.set
    (
        i,
        new fvMesh
        (
            IOobject
            (
                solidsNames[i],
                runTime.timeName(),
                runTime,
                IOobject::MUST_READ
            )
        )
    );
}
```

setRegionSolidFields.H

```
    const fvMesh& mesh = solidRegions[i];
    uniformDimensionedScalarField& D = DSolid[i];
    uniformDimensionedScalarField& z = zSolid[i];
    uniformDimensionedScalarField& boltz = boltzSolid[i];
    uniformDimensionedScalarField& T = TSolid[i];
    volScalarField& conc = concSolid[i];
    volVectorField& E = ESolid[i];
```

createFields.H

```
#include "createFluidFields.H"
#include "createSolidFields.H"
```

createMeshes.H

```
regionProperties rp(runTime);
#include "createFluidMeshes.H"
#include "createSolidMeshes.H"
```

createMeshesPostProcess.H

```
#include "createMeshes.H"
 if (!fluidRegions.size())
{
    FatalErrorIn(args.executable())
        << "No fluid meshes present" << exit(FatalError);
}
fvMesh& mesh = fluidRegions[0];
```

membraneSolver.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           |  Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.
    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.
    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
Application
    membraneSolver
Description
    This solver uses PISO algorithm to compite fluid velocity and pressure
    for transient cases.
\*---------------------------------------------------------------------------*/
#include "fvCFD.H"
#include "pisoControl.H"
#include "rhoThermo.H"
#include "turbulentFluidThermoModel.H"
#include "fixedGradientFvPatchFields.H"
#include "regionProperties.H"
#include "solidThermo.H"
#include "radiationModel.H"
#include "fvOptions.H"
#include "coordinateSystem.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
    #define NO_CONTROL
    #define CREATE_MESH createMeshesPostProcess.H
    #include "postProcess.H"
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMeshes.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"
    while (runTime.loop())
```

```
{
    Info<< "Time = " << runTime.timeName() << nl << endl;
    forAll(fluidRegions, i)
    {
        Info<< "\nSolving for fluid region "
            << fluidRegions[i].name() << endl;
        #include "setRegionFluidFields.H"
    pisoControl piso(fluidRegions[i]);
    #include "CourantNo.H"
    // Momentum predictor
    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
      + fvm::div(phi, U)
      - fvm::laplacian(nu, U)
    );
    if (piso.momentumPredictor())
    {
        solve(UEqn == -fvc::grad(p));
    }
    // --- PISO loop
    while (piso.correct())
    {
        volScalarField rAU(1.0/UEqn.A());
        volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
        surfaceScalarField phiHbyA
        (
            "phiHbyA",
            fvc::flux(HbyA)
          + fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
        );
        adjustPhi(phiHbyA, U, p);
        // Update the pressure BCs to ensure flux consistency
        constrainPressure(p, U, phiHbyA, rAU);
        // Non-orthogonal pressure corrector loop
        while (piso.correctNonOrthogonal())
        {
            // Pressure corrector
            fvScalarMatrix pEqn
            (
                fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
            );
            pEqn.setReference(pRefCell, pRefValue);
            pEqn.solve(fluidRegions[i].solver(p.select(piso.finalInnerIter())));
            if (piso.finalNonOrthogonalIter())
            {
                phi = phiHbyA - pEqn.flux();
            }
        }
        #include "continuityErrs.H"
        U = HbyA - rAU*fvc::grad(p);
        U.correctBoundaryConditions();
    }
    fvScalarMatrix concEqn
    (
            fvm::ddt(conc)
```

```
                + fvm::div(phi, conc)
                - fvm::laplacian(D, conc)
                + D*conc*z/boltz/T*fvc::div(E)
        );
        concEqn.solve();
        }
        //SOLID
        forAll(solidRegions, i)
        {
            Info<< "\nSolving for solid region "
                << solidRegions[i].name() << endl;
            #include "setRegionSolidFields.H"
        fvScalarMatrix concEqn
        (
                fvm::ddt(conc)
                - fvm::laplacian(D, conc)
                + D*conc*z/boltz/T*fvc::div(E)
        );
        concEqn.solve();
        }
        runTime.write();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;
    return 0;
}
// ************************************************************************* //
```

membraneSolver

0/fluidRtop

conc

```
/*---------------------------------*- C++ -*-----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*-----------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0/fluidRtop";
    object      conc;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [ 0 -3 0 0 1 0 0 ];
internalField   uniform 0;
boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    inlet2
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet2
    {
        type            zeroGradient;
    }
    anode
    {
        type            zeroGradient;
        value           uniform 0;
    }
    fluidRtop_to_solidR
    {
        type            mappedFixedInternalValue;
        value           uniform 0;
        interpolationScheme cell;
        setAverage      no;
        average         0;
    }
}
// ************************************************************************* //
```

E

```
/*---------------------------------*- C++ -*-----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*-----------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0/fluidRtop";
    object      E;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [ 1 1 -3 0 0 -1 0 ];
internalField   uniform ( 0 0 -61.125 );
boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    inlet2
    {
        type            fixedValue;
        value           uniform ( 0 0 -61.125 );
    }
    outlet2
    {
        type            fixedValue;
        value           uniform ( 0 0 -61.125 );
    }
    anode
    {
        type            zeroGradient;
        value           uniform ( 0 0 -61.125 );
    }
    fluidRtop_to_solidR
    {
        type            zeroGradient;
        value           uniform ( 0 0 -61.125 );
    }
}
// ************************************************************************* //
```

p

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0/fluidRtop";
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [ 0 2 -2 0 0 0 0 ];
internalField   uniform 0;
boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    inlet2
    {
        type            zeroGradient;
    }
    outlet2
    {
        type            fixedValue;
        value           uniform 0;
    }
    anode
    {
        type            zeroGradient;
        value           uniform 0;
    }
    fluidRtop_to_solidR
    {
        type            zeroGradient;
        value           uniform 0;
    }
}
// ************************************************************************* //
```

U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0/fluidRtop";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [ 0 1 -1 0 0 0 0 ];
internalField   uniform ( 0 0 0 );
boundaryField
{
    walls
    {
        type            noSlip;
    }
    inlet2
    {
        type            fixedValue;
        value           uniform ( -0.04 0 0 );
    }
    outlet2
    {
        type            zeroGradient;
    }
    anode
    {
        type            noSlip;
        value           uniform ( 0 0 0 );
    }
    fluidRtop_to_solidR
    {
        type            noSlip;
        value           uniform ( 0 0 0 );
    }
}
// ************************************************************************* //
```

constant

fluidRtop

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedScalarField;
    object      boltz;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [1 2 -2 -1 0 0 0];
value           1.38064852e-23;
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedScalarField;
    object      D;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -1 0 0 0 0];
value           1.089e-9;
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedScalarField;
    object      D;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -1 0 0 0 0];
value           1.004e-6;
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 1 0 0 0];
value           293.15;
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedScalarField;
    object      z;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions    [0 0 1 0 0 1 0];
value           1.60217662e-19;
// ************************************************************************* //
```

regionProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      regionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
regions
(
    fluid       (fluidRbot fluidRtop)
    solid       (solidR)
);
// ************************************************************************* //
```

system

fluidRtop

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      changeDictionaryDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
U
{
    internalField   uniform (0 0 0);
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type            noSlip;
            }
            anode
            {
                type            noSlip;
            }
    }
}
p
{
    internalField   uniform 0;
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type            zeroGradient;
            }
            anode
            {
                type            zeroGradient;
            }
    }
}
conc
{
    internalField   uniform 0;
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type                mappedFixedInternalValue;
                interpolationScheme cell;
                setAverage          no;
                average             0;
                value                   uniform 0;
            }
            anode
            {
                type                zeroGradient;
            }
    }
}
E
{
    internalField    uniform ( 0 0 0 );
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type                zeroGradient;
            }
            anode
            {
                type                zeroGradient;
            }
    }
}
phiE
{
    internalField    uniform 0;
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type                zeroGradient;
            }
            anode
            {
                type                zeroGradient;
            }
    }
}
rho
{
    internalField    uniform 0;
    boundaryField
    {
            fluidRtop_to_solidR
            {
                type                zeroGradient;
            }
            anode
            {
                type                zeroGradient;
            }
    }
}
// ********************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
numberOfSubdomains 4 ;
method          simple;
simpleCoeffs
{
    n               (4 1 1);
    delta           0.001;
}
/*
hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}
*//*
manualCoeffs
{
    dataFile        "";
}
*/
distributed     no;
roots           ( );
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default         Euler;
}
gradSchemes
{
    default         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwind grad(U);
    div(phi,k)      bounded Gauss upwind;
    div(phi,epsilon) bounded Gauss upwind;
    div(phi,R)      bounded Gauss upwind;
    div(R)          Gauss linear;
    div(phi,nuTilda) bounded Gauss upwind;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div(rhoFlux,rho)    Gauss upwind;
    div(phi,conc)       Gauss linear;
    div(E)                      Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    p
    {
        solver           GAMG;
        tolerance        1e-7;
        relTol           0.01;
        smoother         DICGaussSeidel;
    }
    pFinal
    {
        $p;
        relTol          0;
    }
    "(U|k|epsilon)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }
    "(U|k|epsilon)Final"
    {
        $U;
        relTol          0;
    }
    conc
    {
        solver          smoothSolver;
        smoother                symGaussSeidel;
        tolerance               1e-06;
        relTol                  0;
    }
    phiE
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-05;
        relTol          0.1;
    }
    rho
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol           0.2;
    }
}
PIMPLE
{
    nNonOrthogonalCorrectors 0;
    nCorrectors          2;
}
PISO
{
    nCorrectors      1;
    nNonOrthogonalCorrectors 1;
    pRefCell         0;
    pRefValue        0;
}
SIMPLE
{
    //momentumPredictor yes;
    nNonOrthogonalCorrectors 0;
    pRefCell         0;
    pRefValue        0;
   // rhoMin           0.2;
   // rhoMax           2;
}
// ************************************************************************* //
```

blockMeshDict simple mesh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.01;
vertices
(
        (0 0 0)                 //0
        (20 0 0)                //1
        (20 5 0)                //2
        (0 5 0)                 //3
        (0 0 2)                 //4
        (20 0 2)                //5
        (20 5 2)                //6
        (0 5 2)                 //7
        (0 0 2.1)               //8
        (20 0 2.1)              //9
        (20 5 2.1)              //10
        (0 5 2.1)               //11
        (0 0 4.1)               //12
        (20 0 4.1)              //13
        (20 5 4.1)              //14
        (0 5 4.1)               //15
);
blocks
(
        hex (0 1 2 3 4 5 6 7)           (150 30 30) simpleGrading (1 1 1)
        hex (4 5 6 7 8 9 10 11)         (150 30 10) simpleGrading (1 1 1)
        hex (8 9 10 11 12 13 14 15)     (150 30 30) simpleGrading (1 1 1)
)
edges
(
);
boundary
(
        walls
        {
                type wall;
                faces
                (
                        (0 1 5 4)
                        (3 2 6 7)
                        (4 5 9 8)
                        (5 6 10 9)
                        (7 6 10 11)
                        (4 7 11 8)
                        (8 9 13 12)
                        (11 10 14 15)
                );
        }
        inlet
        {
                type patch;
                faces
                (
                        (0 3 4 7)
                );
        }
        outlet
        {
                type patch;
                faces
                (
                        (1 2 6 5)
                );
        }
        inlet2
        {
                type patch;
                faces
                (
                        (9 10 14 13)
                );
        }
        outlet2
        {
                type patch;
                faces
                (
                        (8 11 15 12)
                );
        }
        cathode
        {
                type patch;
                faces
                (
                        (0 1 2 3)
                );
        }
        anode
        {
                type patch;
                faces
                (
                        (12 13 14 15)
                );
        }
);
mergePatchPairs
(
);
// ************************************************************************* //
```

blockMeshDict detailed mesh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.01;
vertices
(
        (0 0 0)                 //0
        (20 0 0)                //1
        (20 5 0)                //2
        (0 5 0)                 //3
        (0 0 2)                 //4
        (20 0 2)                //5
        (20 5 2)                //6
        (0 5 2)                 //7
        (20 5 0.66667)          //8
        (20 5 1)                //9
        (20 5 1.33333)          //10
        (20 0 1.33333)          //11
        (20 0 1)                //12
        (20 0 0.66667)          //13
        (20 2.5 0.66667)        //14
        (20 2.83333 1)          //15
        (20 2.5 1.33333)        //16
        (20 2.16667 1)          //17
        (0 5 0.66667)           //18
        (0 5 1)                 //19
        (0 5 1.33333)           //20
        (0 0 1.33333)           //21
        (0 0 1)                 //22
        (0 0 0.66667)           //23
        (0 2.5 0.66667)         //24
        (0 2.83333 1)           //25
        (0 2.5 1.33333)         //26
        (0 2.16667 1)           //27
        (0 0 2.1)               //28
        (20 0 2.1)              //29
        (20 5 2.1)              //30
        (0 5 2.1)               //31
        (0 0 4.1)               //32
        (20 0 4.1)              //33
        (20 5 4.1)              //34
        (0 5 4.1)               //35
        (20 5 2.76667)          //36
        (20 5 3.1)              //37
        (20 5 3.43333)            //38
        (20 0 3.43333)            //39
        (20 0 3.1)                //40
        (20 0 2.76667)            //41
        (20 2.5 2.76667)          //42
        (20 2.83333 3.1)          //43
        (20 2.5 3.43333)          //44
        (20 2.16667 3.1)          //45
        (0 5 2.76667)             //46
        (0 5 3.1)                 //47
        (0 5 3.43333)             //48
        (0 0 3.43333)             //49
        (0 0 3.1)                 //50
        (0 0 2.76667)             //51
        (0 2.5 2.76667)           //52
        (0 2.83333 3.1)           //53
        (0 2.5 3.43333)           //54
        (0 2.16667 3.1)           //55
        (20 2.5 0)                //56
        (0 2.5 0)                 //57
        (20 2.5 2)                //58
        (0 2.5 2)                 //59
        (20 2.5 2.1)              //60
        (0 2.5 2.1)               //61
        (20 2.5 4.1)              //62
        (0 2.5 4.1)               //63
);
blocks
(
        //lower section
        hex (0 1 56 57 23 13 14 24)     (40 10 5) simpleGrading (1 1 1)
        hex (57 56 2 3 24 14 8 18)      (40 10 5) simpleGrading (1 1 1)
        hex (23 13 14 24 22 12 17 27)   (40 10 4) simpleGrading (1 1 1)
        hex (22 12 17 27 21 11 16 26)   (40 10 4) simpleGrading (1 1 1)
        hex (21 11 16 26 4 5 58 59)     (40 10 5) simpleGrading (1 1 1)
        hex (26 16 10 20 59 58 6 7)     (40 10 5) simpleGrading (1 1 1)
        hex (25 15 9 19 26 16 10 20)    (40 10 4) simpleGrading (1 1 1)
        hex (24 14 8 18 25 15 9 19)     (40 10 4) simpleGrading (1 1 1)
        hex (27 17 14 24 26 16 15 25)   (40 4 4) simpleGrading (1 1 1)
        //membrane
        hex (4 5 58 59 28 29 60 61)     (40 10 5) simpleGrading (1 1 1)
        hex (59 58 6 7 61 60 30 31)     (40 10 5) simpleGrading (1 1 1)
        //upper section
        hex (28 29 60 61 51 41 42 52)   (40 10 5) simpleGrading (1 1 1)
        hex (61 60 30 31 52 42 36 46)   (40 10 5) simpleGrading (1 1 1)
        hex (52 42 36 46 53 43 37 47)   (40 10 4) simpleGrading (1 1 1)
        hex (53 43 37 47 54 44 38 48)   (40 10 4) simpleGrading (1 1 1)
        hex (49 39 44 54 32 33 62 63)   (40 10 5) simpleGrading (1 1 1)
        hex (54 44 38 48 63 62 34 35)   (40 10 5) simpleGrading (1 1 1)
        hex (50 40 45 55 49 39 44 54)   (40 10 4) simpleGrading (1 1 1)
        hex (51 41 42 52 50 40 45 55)   (40 10 4) simpleGrading (1 1 1)
        hex (55 45 42 52 54 44 43 53)   (40 4 4) simpleGrading (1 1 1)
);
edges
(
        arc 14 15(20 2.73566 0.76434)
        arc 15 16(20 2.73566 1.23566)
        arc 16 17(20 2.26434 1.23566)
        arc 17 14(20 2.26434 0.76434)
        arc 24 25(0 2.73566 0.76434)
        arc 25 26(0 2.73566 1.23566)
        arc 26 27(0 2.26434 1.23566)
        arc 27 24(0 2.26434 0.76434)
        arc 42 43(20 2.73566 2.86434)
        arc 43 44(20 2.73566 3.33566)
        arc 44 45(20 2.26434 3.33566)
        arc 45 42(20 2.26434 2.86434)
        arc 52 53(0 2.73566 2.86434)
        arc 53 54(0 2.73566 3.33566)
        arc 54 55(0 2.26434 3.33566)
        arc 55 52(0 2.26434 2.86434)
);
boundary
(
        walls
        {
                type wall ;
                faces
                (
//bottom front
(0 1 13 23)
(23 13 12 22)
(22 12 11 21)
(21 11 5 4)
//bottom right
(1 56 14 13)
(56 2 8 14)
(14 8 9 15)
(15 9 10 16)
(16 10 6 58)
(11 16 58 5)
(12 17 16 11)
(13 14 17 12)
//bottom back
(3 2 8 18)
(18 8 9 19)
(19 9 10 20)
(20 10 6 7)
//bottom left
(0 57 24 23)
(57 3 18 24)
(24 18 19 25)
(25 19 20 26)
(26 20 7 59)
(21 26 59 4)
(22 27 26 21)
(23 24 27 22)
//membrane walls
(4 5 29 28)
(5 58 60 29)
(58 6 30 60)
(7 6 30 31)
(4 59 61 28)
(59 7 31 61)
// top front
(28 29 41 51)
(51 41 40 50)
(50 40 39 49)
(49 39 33 32)
//top right
(29 60 42 41)
(60 30 36 42)
(42 36 37 43)
(43 37 38 44)
(44 38 34 62)
(39 44 62 33)
(40 45 44 39)
(41 42 45 40)
//top back
(31 30 36 46)
(46 36 37 47)
(47 37 38 48)
(48 38 34 35)
//top left
(28 61 52 51)
(61 31 46 52)
(52 46 47 53)
(53 47 48 54)
(54 48 35 63)
(49 54 63 32)
(50 55 54 49)
(51 52 55 50)
                );
        }
        inlet
        {
                type patch;
                faces
                (
                        (24 25 26 27)
                );
        }
        outlet
        {
                type patch;
                faces
                (
                        (14 15 16 17)
                );
        }
        inlet2
        {
                type patch;
                faces
                (
                        (42 43 44 45)
                );
        }
        outlet2
        {
                type patch;
                faces
                (
                        (52 53 54 55)
                );
        }
        cathode
        {
                type patch;
                faces
                (
                        (0 1 56 57)
                        (57 56 2 3)
                );
        }
        anode
        {
                type patch;
                faces
                (
                        (32 33 62 63)
                        (63 62 34 35)
                );
        }
);
mergePatchPairs
(
);
// ************************************************************************* //
```

controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     pimpleFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         100000;
deltaT          0.01;
writeControl    timeStep;
writeInterval   50;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable no;
adjustTimeStep  no;
maxCo           5;
// ************************************************************************* //
```

createBafflesDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      createBafflesDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
internalFacesOnly true;
baffles
{
    memb1
    {
        type        faceZone;
        zoneName    membraneBoundary1;
        patches
        {
           master
            {
                name            master1;
                type            mapped Patch;
              sampleMode             nearestPatchFace;
            }
            slave
            {
                name            slave1;
                type            mappedPatch;
              sampleMode             nearestPatchFace;
            }
        }
    }
    memb2
    {
        type        faceZone;
        zoneName    membraneBoundary2;
        patches
        {
           master
            {
                name            master2;
                type            mappedPatch;
              sampleMode             nearestPatchFace;
            }
            slave
            {
                name            slave2;
                type            mappedPatch;
              sampleMode             nearestPatchFace;
            }
        }
    }
}
// ************************************************************************* //
```

topoSetDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
actions
(
    {
        name    bottomCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (0 0 0)(0.2 0.05 0.02);
        }
    }
    {
        name    fluidRbot;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        sourceInfo
        {
            set bottomCellSet;
        }
    }
    {
        name    topCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (0 0 0.021)(0.2 0.05 0.041);
        }
    }
    {
        name    fluidRtop;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        sourceInfo
        {
            set topCellSet;
        }
    }
    {
        name    topCellSet;
        type     cellSet;
        action   new;
        source   boxToCell;
        sourceInfo
        {
            box (0 0 0.02)(0.2 0.05 0.021);
        }
    }
    {
        name     solidR;
        type     cellZoneSet;
        action   new;
        source   setToCellZone;
        sourceInfo
        {
            set topCellSet;
        }
    }
    {
        name     membraneBoundary1FaceSet;
        type     faceSet;
        action   new;
        source   boxToFace;
        sourceInfo
        {
            box (0 0 0.019999) (0.2 0.05 0.020001);
        }
    }
    {
        name     membraneBoundary1;
        type     faceZoneSet;
        action   new;
        source   setToFaceZone;
        sourceInfo
        {
            faceSet membraneBoundary1FaceSet;
        }
    }
    {
        name     membraneBoundary2FaceSet;
        type     faceSet;
        action   new;
        source   boxToFace;
        sourceInfo
        {
            box (0 0 0.020999) (0.2 0.05 0.021001);
        }
    }
    {
        name     membraneBoundary2;
        type     faceZoneSet;
        action   new;
        source   setToFaceZone;
        sourceInfo
        {
            faceSet membraneBoundary2FaceSet;
        }
    }
);
// ******************************************************************* //
```