

**Darcy and non-Darcy fluid flow simulation through well-structured porous medium**

Adriaan De Bolle

Thesis submitted to

obtain the degree of Master

in Engineering:

Chemical Engineering

**Supervisor:**

Prof. dr. ir. S. Kuhn

**Assessor:**

Prof. dr. ir. G. Stefanidis

Ir. M. Van den Eynde

**Mentors:**

Ir. M. Mottaghi

**Academic Year 2015 – 2016**

© Copyright by K.U.Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promoter(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot de K.U.Leuven, Faculteit Ingenieurswetenschappen - Kasteelpark Arenberg 1, B-3001 Heverlee (België). Telefoon +32-16-32 13 50 & Fax. +32-16-32 19 88.

Voorafgaande schriftelijke toestemming van de promoter(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma’s voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

© Copyright by K.U.Leuven

Without written permission of the supervisor(s) and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to K.U.Leuven, Faculty of Engineering - Kasteelpark Arenberg 1, B-3001 Heverlee (Belgium). Telephone +32-16-32 13 50 & Fax. +32-16-32 19 88.

A written permission of the supervisor(s) is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

# Foreword

Firstly, I would like to thank my coordinators that helped me to learn, investigate, discuss problems and solve some bottlenecks and always gave advise when needed to make this dissertation a great success. Prof. S. Kuhn always wanted to be involved and helped with high expertise. Ir. M. Mottaghi gave me daily supervision and together we followed a journey for better understanding. Not every solution is clear and straightforward, but M. Mottaghi guided me often to an alternative way of thinking to tackle problems.

Secondly, I need to give my sincere and honest gratitude to people who helped me “survive” five years of though engineering studies. For this, I look mainly to Jirka Delaere who pushed me to achieve every time an acceptable level or more for each course. Discussions led us to a better understanding. Enduring my studies to become Chemical Engineer was eased by Peter Van Puyvelde, Program Director. He gave me extraordinarily opportunities (internationally) to broaden my view surpassing the engineering domain.

Lastly, I thank my parents, brother, family and friends to shape me as a loving person that looks further than engineering. Life is made greater from experiences work related, leisure and pleasure.

Adriaan Hugo Elise De Bolle

# Abstract

Best kept to 1 page

Samenvatting 1 – 3 p

**Table of Contents**

[Foreword iii](#_Toc452370893)

[Abstract iv](#_Toc452370894)

[List of Figures and Tables vii](#_Toc452370895)

[List of abbreviations and symbols ix](#_Toc452370896)

[Chapter I: Introduction 11](#_Toc452370897)

[Goal 11](#_Toc452370898)

[Technical details 11](#_Toc452370899)

[Why simulation? 11](#_Toc452370900)

[Why porous media? 12](#_Toc452370901)

[Chapter II: Literature 14](#_Toc452370902)

[Flow through porous media 14](#_Toc452370903)

[Darcy's law 15](#_Toc452370904)

[Reynolds as onset for non-Darcy 16](#_Toc452370905)

[Flow characteristics through an explicit packed bed 17](#_Toc452370906)

[Experimental flow 17](#_Toc452370907)

[Apparent permeability 21](#_Toc452370908)

[Chapter III: Geometry and mesh generation 22](#_Toc452370909)

[Prerequisite information 22](#_Toc452370910)

[Randomization of spherical packings 23](#_Toc452370911)

[Building geometry and meshing 24](#_Toc452370912)

[Analytical control and validation of mesh 26](#_Toc452370913)

[Conclusion 29](#_Toc452370914)

[Chapter III: Solving 30](#_Toc452370915)

[Prior to running solve algorithms 30](#_Toc452370916)

[Solvers 30](#_Toc452370917)

[Convergence control/criteria 31](#_Toc452370918)

[Laminar versus Turbulency models 33](#_Toc452370919)

[Chapter V: Results and discussion 36](#_Toc452370920)

[2-dimensional 36](#_Toc452370921)

[Trial 1 37](#_Toc452370922)

[Normalisation 37](#_Toc452370923)

[Cases 38](#_Toc452370924)

[Trial 2 40](#_Toc452370925)

[Comparison explicit and implicit method 40](#_Toc452370926)

[What’s next? 40](#_Toc452370927)

[Conclusion 40](#_Toc452370928)

[Bibliography 41](#_Toc452370929)

[Appendices 44](#_Toc452370930)

[A. General computational procedure to characterise property distributions 44](#_Toc452370931)

[B. Structured packing 46](#_Toc452370932)

[C. Calculation of void space for spherical packing 48](#_Toc452370933)

[D. Randomization based on random coordinates 50](#_Toc452370934)

[E. Randomization based on DEM (Yade) 54](#_Toc452370935)

[F. Building geometry for structured / unstructured mesh 56](#_Toc452370936)

[G. Building geometry for unstructured mesh 57](#_Toc452370937)

# List of Figures and Tables

List of Figures

Figure 1 : Difference between simulation with and without explicit pore topology (Soulaine, 2015) 14

Figure 2 : Examples of porous media for a wide range of length scales (Soulaine, 2015) 15

Figure 3: Typical transducer cross section for pressure drop sensors 18

Figure 4: Reduced pressure in function of velocity for three different porous structures. (all data plotted) 20

Figure 5: Reduced pressure in function of velocity for three different porous structures within Darcy, transitional and Forchheimer regime. (low range velocity data) 20

Figure 6: Plot of 1/Kapp vs. ρU/ μ to get KF or KT and β 21

Figure 7: Randomization of BCC packing 23

Figure 8: Geometry based on divided disk with 15mm packing 24

Figure 9: Geometry based on one cylindrical volume with 3mm packing 25

Figure 13: Validated mesh cell density for structured, unstructured without viscous layers 28

Figure 14: Ongoing residual plot of a simpleFoam simulation that diverges after scheme change 32

Figure 15: Wall function are black boxes for near wall treatment based on a model and not the simulation and span at least viscous sub layer and buffer layer (Guerrero 2015) 35

Figure 16: Velocity magnitude profile at iteration step 500, 1000, 1500 and 2000 for a parallel 2D porous plane 36

Figure 17: Residual plot for 2D porous plane 37

Figure 18: Arrangement spherical packing (Brown, H. Eugene LeMay, Bursten, Murphy, & Woodward, 2012) 49

List of Tables

Table 1: Specifications of the tube dimensions (Dukhan, Bağcı, & Özdemir, 2014) 18

Table 2: Fluid properties (water) 18

Table 3: Velocity range of different flow regimes (Dukhan, Bağcı, & Özdemir, 2014) 19

Table 4: Permeability and Forchheimer coefficient in different flow regimes (Dukhan, Bağcı, & Özdemir, 2014) 19

Table 5: Different cases that this dissertation wants to simulate 19

Table 6: Summary analytical parameters of laminar fluid flow for inlet zone 27

Table 7: Tetrahedral meshes with its characteristics and velocity solution 27

Table 8: Comparison between analytical parameters and simulation obtained with octahedral and tetrahedral meshes 28

Table 9: Normalised pressure drop over a porous zone (1st case scenario: u=0,001m/s) 37

Table 10: Simulation results for different cases (Trial 1) 38

Table 11: Used solvers within this papers that are needed for simulation of porous structures (OpenFoam, 2015) 45

Table 12: Interesting terminal commands within OpenFOAM 45

# List of abbreviations and symbols

Parameters are listed in order of appearance.

Fluid property parameters

|  |  |
| --- | --- |
| Shortcut | Full name |
| u | Velocity (locally), m/s |
| u | Velocity vector, [m/s, m/s, m/s]T |
| g | Gravitational acceleration vector, m/s2 |
| q | Volumetric flow rate, m3/s |
| ρ or ρf | Mass density (of fluid), kg/m3 |
| µ | Dynamic viscosity, Pa.s |
| K [[1]](#footnote-1) | Permeability scalar, m2 |
| ν | Cinematic viscosity, m2/s |
| K 1 | Permeability 3 x 3 tensor, m2 |
| F [[2]](#footnote-2) | Forchheimer coefficient, - |
| β 2 | Non-Darcy flow coefficient, m-1 |
| Re [[3]](#footnote-3) | Reynolds number, - |
| Rep | Reynolds based on particle size, - |
| ReK | Reynolds based on permeability, - |
| Re\* | Reynolds based on Ergun, - |
| Kapp | Apparent permeability, m |
| ∆p/L | Normalised pressure drop, Pa/m |
| ∆p/Lu | Reduced pressure drop, Pa.s/m2 |
| xfd,H | Distance to fully developed, m |
| U [[4]](#footnote-4) | Superficial velocity, m/s |
| umax | Maximal velocity, m/s |
| hf | Friction loss, m |
| d | Darcy resistance vector, |
| f | Forchheimer resistance vector, |
| τ [[5]](#footnote-5) | Stress, |
| τw | Wall stress, |
| τR | Reynolds-averaged stress, |
| u' | Velocity relative to average velocity, m/s |
| µT | Turbulent viscosity, Pa.s |
| uτ | Friction velocity, m/s |
| Cf [[6]](#footnote-6) | Skin friction coefficient, |
| Co | Courant number, - |
| δt | Time interval |

Dimensional parameters (geometry)

|  |  |
| --- | --- |
| Shortcut | Full name |
| A | Surface, m2 |
| D | Tube diameter, m |
| ε | Porosity or void space, - |
| Li | Length of inlet zone, m |
| Lp | Length of porous zone, m |
| Lo | Length of outlet zone, m |
| Rc | Radius of cylinder/tube, m |
| Rs / Ds | Radius / diameter of (spherical) particle, m |
| τ [[7]](#footnote-7) | Tortuosity, - |
| xSV |  |
| y+ | yPlus or dimensionless wall distance, - |

# Chapter I: Introduction

Computational Fluid Dynamics (CFD) is a broad area. This dissertation looks to simulations of flow through porous media. Porous structures are widely used and even in unexpected applications. This section gives the goal of this dissertation, technical details how to build up the simulations made, why the need for computational approach and overview of applications in different domains.

## Goal

The main objective of this thesis is to obtain conclusions whether Darcy’s law can be used in simulations with a simpler “black box” method instead of using explicit porous topology and Navier-Stokes equations. Firstly, understanding of simulations, spherical packings, analytical control, computational models, theoretical background of Darcy’s law and experimental data is necessary to acquire the main objective.

## Technical details

Geometries are built in Salome V7\_6\_0, open source CAD (computer-aided design) software. Discrete Element Method (DEM) within framework Yade is used to create packings in geometry modules prior to simulation within OpenFOAM. Therefore, it was not necessary to couple CFD and DEM to generate solutions. For the simulations, within OpenFOAM-2.4.0, the finite volume method is used. This method is more understandable, because physical conservation equations (balance equations) are valid for each control volume. OpenFOAM is an open source software package used in Ubuntu – Linux (operational system).

Highly Recommended

All files to build up a simulation in all software mentioned above are located in a github repository and changes in this project can be followed:

<https://github.com/AdriaanDB/Thesis-2016-KULeuven>

## Why simulation?

Simulations want to achieve an accurate, precise representation of reality. It is interesting to build a simulation for creating new applications or scaling up existing applications. Real process variables have certain disturbances depending on the system. Therefore, it is also important that simulations give a stable solution that is not sensitive towards small changes. Errors mainly slip in simulations because of incorrect set of modelling equations, discretization schemes and iterative procedures that don’t converge linked to the tolerance level used.

Utilisation of simulations is limited by computational power. The field of computational simulation started in the ‘70s. From the ‘90s personal computers were powerful enough to simulate simple models. Still today, there are limitations. It takes several days to complete a simulation of microalgae using computers at a university. Maybe not expected, but simulating microalgae and the biological model of photosynthesis needs also knowledge of the streamlines (CFD) in a pond where the algae grow. (Chachuat, 2015) Simulation of the human brain during 1s with the fourth most powerful supercomputer in the world needs 40min of computation time. (Sparkes, 2014)

By experience, making the fluid domain around spherical packing takes at least a day on a personal computer. Meshing of this structure takes several hours and solving as well. But even more time-consuming is manual corrections and iterative procedures, but this will be more clear throughout this dissertation.

Gradually with further increasing computational power the use of simulations will intensify without doubt.

Intermezzo

General procedure to make simulations can be read in Appendix A.

## Why porous media?

Porous media are about porosity, permeability and Darcy's law (and corrections). Study and knowledge of porous media is important in many different industry branches. Applications are found in oil recovery, nuclear safety, chemical engineering and others. (Soulaine, 2015) Following some examples:

* **Catalysts**

Catalysts are widely used to improve reaction kinetics mostly in multitubular reactors. Generally, catalysis is a factor in 90% of all chemical processes, as such, a vital part in the chemical industry. (National Academy of Sciences, 2000) Catalyst is divided generally into homogeneous and heterogeneous catalysts. For the latter, solid particles are added to the reaction medium. Specific example is the reaction of ethylene to ethylene oxide where selectivity for this reaction is favoured against side reactions, because of the use of a silver catalyst with a porous, cylindrical structure. (US Brevet n° US 4368144 A, 1980) So catalyst particles can have a porous structure and as well form all together a porous medium inside a tube.

* **Mixing**

A random packing has an amorphous structure and gives irregular flow pathways, which increase the tortuosity τ. Therefore, better mixing of different phases is (temporarily) possible, which can increase overall interphase surface and faster transition to chemical products. Mixing reduces also the (bio)film thickness around solid particles, which give a smaller resistance to mass transfer from bulk to solid surface. (Taherzadeh, Picioreanu, & Horn, 2012)

Simulation example of mixing (interFoam)

* **Membrane Technology**

Membranes are in fact thin cuts of a porous structure. Therefore, also for pressure driven membranes Darcy’s law and specifically for spherical particles and laminar flow Kozeny-Carman’s law can be used. Membranes are used for separating a fluid from its solutes. The solutes are retained and the fluid permeates through the membrane. Depending on pore size one distinguishes micro-, ultra- and nanofiltration. For very narrow pores (<1nm) when filtering salt solution, osmotic pressure has to be taken into account (reversed osmosis). (Van Der Bruggen, Vandecasteele, Van Gestel, Doyen, & Leysen, 2003)

* **Filtration**

-Surface filtration: Particles are retained at the surface of a filter media. These particles form a *cake layer* that increases resistance to flow and increases the pressure drop. Correct estimation of this build-up and the pressure drop over this irregular packed bed gives better scheduling for removal of the cake layer. (Harvard Corporation, 2016)

-Depth filtration: Particles are retained within a packed bed e.g. sand filtration or trickling filters to purify water. Here large particles can also build a cake layer or biofilm that retains unwanted compounds in water at the particle surface. Smaller particles are blocked depending on the different pathways. (Harvard Corporation, 2016)

* **Packed beds**

Packed beds are widely used in the (bio)chemical industry. For biological systems it is very important to measure temperature gradients and pressure gradients. Temperature is important for optimal growth of microorganisms. A traditional, fixed packed bed can be found in solid state fermentation. For these applications the porosity and humidification are additionally important parameters. Air with water droplets and an inlet temperature flows through the bed of solid substrates and can cause movement of the particles and even fluidization of the bed. This can cause porosity changes through the bed. (Pandey, 2003)

* **Oil & Gas**

Fracking of Shale gas is also an example of porous media. Shale gas is found inside porous reservoirs and important is the viscosity of the gas to flow out of these pores. Therefore, viscosity changes are imposed by adding chemicals and breaking up of pores by high-pressured liquid (fracking) to allow more gas to flow out. As such extraction of shale gas requires investigation of the porous characteristics of the well. (King, 2005-2016)

It is clear that porous media and its extents are widely used in different industry branches (physical and chemical). For the further progress of this thesis the bed will be fixed without porosity changes, without chemical reactions and without heat transfer.

# Chapter II: Literature

Before addressing simulations, it is always recommended to get a theoretical background. This section gives therefore a literature overview concerning various aspects regarding porous media: difference in modelling, representative equations and experiments done in the past by third parties.

## Flow through porous media

Usually flow is rather slow in porous media. This is called creeping flow. In this case the assumption of incompressible flow is accurate and flow is independent of flow rate and the type of the fluid. These conditions are governed by the Stokes equations (Re < 1), a subset of the Navier-Stokers equations. (Soulaine, 2015) Care should be addressed when putting thresholds such as Re<1, because this is not always correctly bounding different flow regimes. As well the term “creeping” can be misleading, because velocity can rise drastically in small pores. Direct modelling uses the Stokes equations and boundary conditions. This will lead to option A in the geometry section. Continuum or Darcy modelling uses the Darcy equations based on average velocity and average pressure within Representative Elementary Volumes (REV) without explicit pore topology. This will lead to option B. Difference between A and B can be seen in Figure 1. Figure 2 gives a wide range of porous media at different length scales. For example, to simulate a (burning) forest, it is very computational power consuming to represent all branches and leaves. When considering the forest as a porous continuum medium Darcy’s law can be used, which decreases needed power. (Soulaine, 2015)

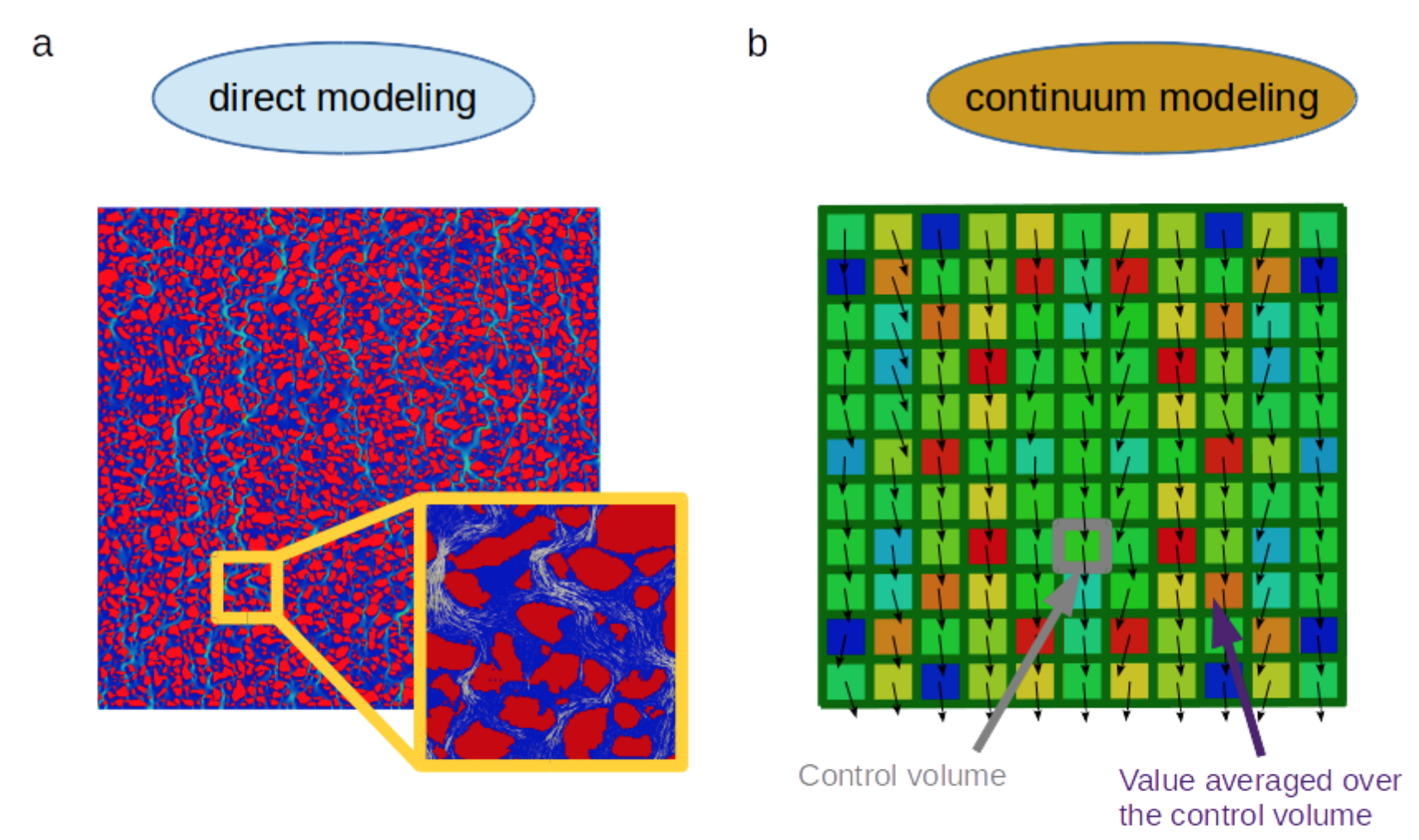


Figure 1 : Difference between simulation with and without explicit pore topology (Soulaine, 2015)

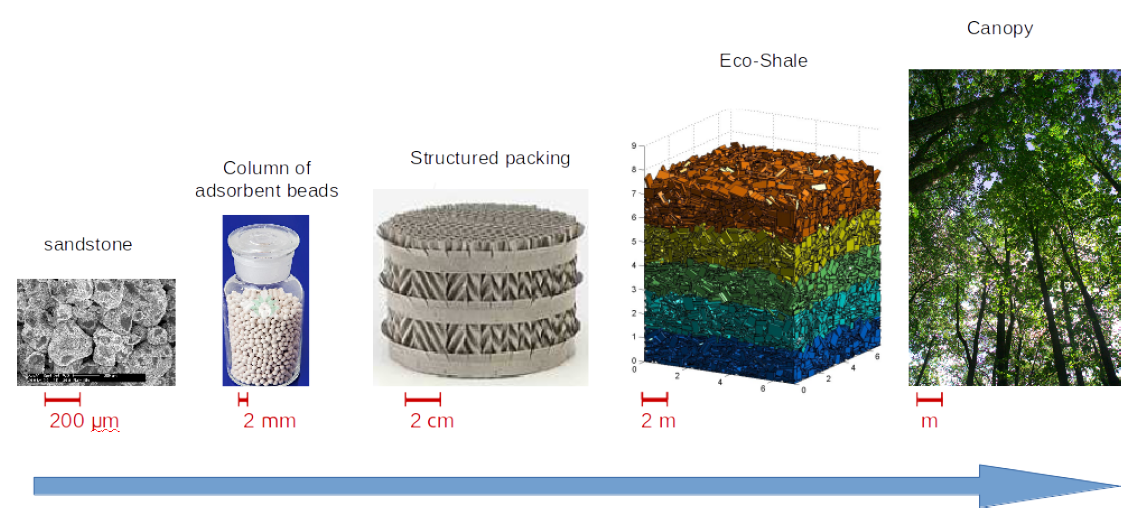


Figure 2 : Examples of porous media for a wide range of length scales (Soulaine, 2015)

## Darcy's law

Vector form of Darcy's law (and conservation of mass), where stands for liquid phase filling the void space:

Analytical form of Darcy's equation for one dimension, where U is the superficial velocity or flow volumetric flux:

Forchheimer proposed a quadratic correction term (1D) seen underneath. When viscous effects are dominant this correction term is zero. When inertial effects start to play a role, at higher velocity, the term becomes non-zero. The effect is shown in Figure 5.

Both parameters K and F strongly depend on the internal structure of the porous medium. Permeability stands for openness of the porous medium to fluid flow. (Dukhan, Bağcı, & Özdemir, 2014) This includes not only porosity or void space, but also the amount of possible paths and narrowness. Important to notice is that these parameters are not truly constant since they depend on the flow regime (Darcy, transitional, Forchheimer or turbulent). (Dukhan, Bağcı, & Özdemir, 2014) Therefore, one needs to distinguish different permeability parameters, and , in different flow regimes.

## Reynolds as onset for non-Darcy

Traditionally the Reynolds number (Re) characterises the laminar, transitional and turbulent flow regime with each one a different set of equations. For Hagen-Poiseuille tube flow (without any obstacles and tortuosity equals one) Re is given by:

For porous media several Re are found in literature depending on the context:

|  |  |  |
| --- | --- | --- |
|  |  | ( 1 ) |
|  |  | ( 2 ) |
|  |  | ( 3 ) |
|  |  |  |

Equation 1 is used when the geometry of the porous medium is known, because particle diameter or a characteristic length determines the value. Equation 2 uses permeability, found by experiments, so the explicit geometry details don’t have to be known. Last equation (3) is used for turbulent regimes (Ergun) and correlations of friction factor. To define Darcy and non-Darcy flow and the transition between these regimes mostly equation one and two are used as threshold between these regimes. This is important, because different equations are valid within these regimes and when trespassing non-Darcy threshold, the Forchheimer correction term, with different permeability coefficient, is needed.

## Flow characteristics through an explicit packed bed

This section is based on explicit knowledge of the porous structure and gives a summary of important equations. More detailed analysis can be found in Transport Phenomena or Powder Technology. Recommended book: *Introduction to Particle Technology – Second Edition: Fluid Flow Through a Packed Bed of Particles* (Rhodes, 2008)

Blake-Kozeny relationship, valid for laminar flow, spherical packing and ε < 0.50:

Generally, for all Re-regimes the pressure drop over the bed is given by Ergun, where the first term is for laminar and both for turbulent:

Defining an alternative Reynolds number based on the Ergun equation for laminar and turbulent flow:

one can get a correlation for the friction factor to calculate pressure drop:

and calculate the pressure drop:

## Experimental flow

Simulations are based on real experimental data to make comparison easier. For the well-structured porous medium, a packed bed of spheres of 3mm in diameter is chosen as investigated geometry for this dissertation. This traditional packed bed is experimentally studied by Dukhan et al. in following paper: *Experimental flow in various porous media and reconciliation of Forchheimer and Ergun relations* (Dukhan, Bağcı, & Özdemir, 2014).

Three different porous structures are used for experiments of water flow: 1-mm spherical packing, 3-mm spherical packing and a metal foam with each a porosity of respectively 35%, 35,5% and 87,6%. Dukhan et al. concluded discrepancies between reported permeability and form drag coefficient, because of incorrect detection of different flow regimes. Within this paper different pressure drop correlations, Forchheimer and Ergun, are discussed for the turbulent or post-Darcy regime. Experimental setup investigated is chosen as such all different regimes have experimental results and can be simulated. The setup is divided in an inlet-, porous and outlet section. Geometry specifications are bundled in Table 1 and fluid properties of water in Table 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Li (m) | Lp (m) | Lo (m) | Dc (m) | Ds (m) | (%) |
| 0,2 | 0,304 | 0,2 | 0,0514 | 0,003 | 35,5 |

Table 1: Specifications of the tube dimensions (Dukhan, Bağcı, & Özdemir, 2014)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (kg/m3) | μ (Pa s) | ν (m2/s) |  |  |  |
| 1000 | 0,001 | 1e-6 |  |  |  |

Table 2: Fluid properties (water)

The setup is built to withhold very low and very high flow rates, respectively Darcy and post-Darcy conditions. Pressure was measured by *Validyne* *pressure differential sensors*, model *DP15* and *DP45* for high and low range respectively. How pressure drop is measured is important to compare correctly experiments with simulations. Used sensors are transducers consisting of a diaphragm. (Validyne Engineering) Therefore, best comparison is made with simulated pressure drop between two cross sections. Details about this device is seen in Figure 3 and the operation manual. (Validyne Engineering) Accuracy of the low range sensor is ± 0,25% and of high range ± 0,5%. Including uncertainty of length and velocity gives for the Darcy region a maximum uncertainty of 1,56% for the pressure drop. For all other flow regimes, the uncertainty in the measured pressure drop is maximum 1,01% concerning the experiments of Dukhan et al.

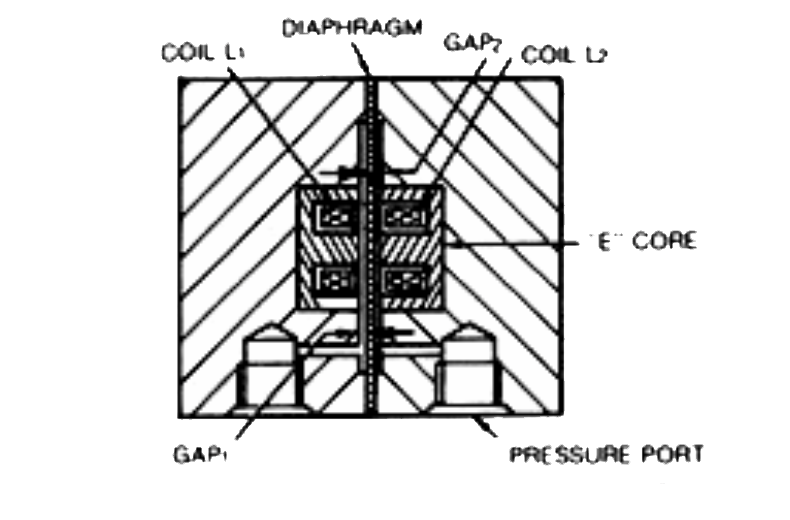


Figure 3: Typical transducer cross section for pressure drop sensors

Measured pressure drop is plotted by Dukhan et al. in Figure 4 and Figure 5. For a 3-mm spherical packing experimental data stretch from Darcy to turbulent regime and investigation of all regimes and transition(s) is possible. Reported flow regimes are summarized in Table 3. Each of these regimes hold different permeability and Forchheimer coefficient as seen in Table 4. Based on these values different cases to simulate are withdrawn as interesting and case choices are given in Table 5. Respectively by case number the case should simulate Darcy’s regime at low pressure drop, Darcy at its limit close to transition, transition regime, Forchheimer where inertia gets important and fully turbulent.

|  |  |  |  |
| --- | --- | --- | --- |
| Darcy | Transition | Forchheimer | Turbulent |
| Velocity Range (m/s) | [0; 0,007] | [0,007; 0,02] | [0,02; 0,09] | [0,09; …] |

Table 3: Velocity range of different flow regimes (Dukhan, Bağcı, & Özdemir, 2014)

|  |  |
| --- | --- |
|  | F (m) |
| Darcy | 6,18 | NA |
| Forchheimer | 7,34 | 0,43 |
| Turbulent | 6,61 | 0,39 |

Table 4: Permeability and Forchheimer coefficient in different flow regimes (Dukhan, Bağcı, & Özdemir, 2014)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case 1 | Case 2 | Case 3 | Case 4 | Case 5 |
| Velocity (m/s) | 0,001 | 0,006 | 0,0135 | 0,021 | 0,012 |

Table 5: Different cases that this dissertation wants to simulate

Dukhan et al. show with their experiments that also within the turbulent regimes an analogous Darcy’s law with correction can be stated as:

As such the Forchheimer correction factor is also not truly constant, but constant in a specific flow regime: Forchheimer or turbulent.

For an estimation of the permeability within Darcy’s regime Kozeny’s theory can be used. Estimation for spherical packing is given by:

which gives for , and :

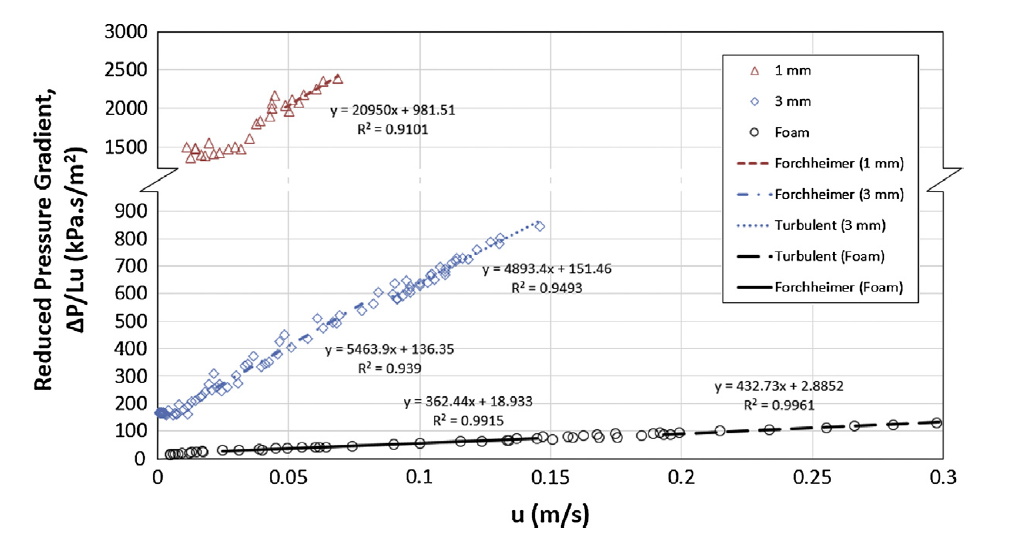


Figure 4: Reduced pressure in function of velocity for three different porous structures. (all data plotted)



Figure 5: Reduced pressure in function of velocity for three different porous structures within Darcy, transitional and Forchheimer regime. (low range velocity data)

## Apparent permeability

Another interesting article with additional information is: *The impact of porous media heterogeneity on non-Darcy flow behaviour from pore-scale simulation.* (Muljadi, Blunt, Raeini, & Bijeljic, 2015) This paragraph is based on this article.

Sometimes a flow with Rep below one is said to be creeping flow, but the onset of non-Darcy flow can vary significantly depending on the porous structure. Simulation showed even a difference of three orders of magnitude for critical ReK. Depending on the porous structure eddies can arise and increase tortuosity, which influences parameters. Determination of the porosity parameters can be done by experimental data, but also by X-ray imaging and direct pore-scale simulations. This paper defines the apparent permeability to highlight the transition from Darcy to non-Darcy flow. Equalling Darcy’s law and Forchheimer correction defines the apparent permeability as follows:

and when plotting the experimental data one can determine the intersect and slope to get a value for and respectively like seen, as example, in Figure 6. This approach can be used for Forchheimer and turbulent regime.

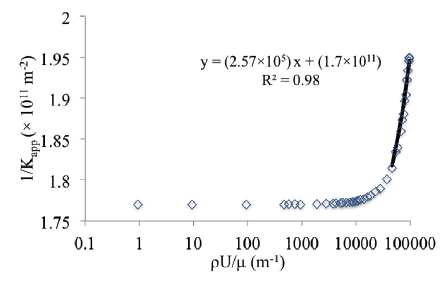
****

Figure 6: Plot of 1/Kapp vs. ρU/ μ to get KF or KT and β

# Chapter III: Geometry and mesh generation

The first step of a simulation is making the geometry and generating the mesh. This section gives different steps to take and gives specific information to build or treat porous structures.

## Prerequisite information

This thesis looks to a flow that enters an inlet zone, where the velocity profile gets fully developed, goes through a porous region and exits passing an outlet zone. Building a cylindrical geometry with porous zone has more flexibility within Salome than blockMesh, an OpenFOAM utility.

There are two options to verify flow characteristics of a porous structure:

1. Building specific **porous structure**: Ideally the mesh gets divided into a structured mesh for zones without porous structure and unstructured mesh for the porous structure. Flow characteristics are obtained by using simpleFoam or icoFoam (with steadyState ddt scheme) as solver. Porosity parameters can be calculated during post-processing steps.
2. Starting from **porosity parameters**: A structured mesh can be generated for the whole tube where a porous, volumetric group (cellZone) is defined. Based on permeability and Forchheimer parameters flow characteristics can be solved by porousSimpleFoam or porousInterFoam for multiphase systems.

For the well-structured porous medium a packed bed of spheres of 3mm in diameter is chosen and an article (Dukhan, Bağcı, & Özdemir, 2014) as experimental data reference, which was summarized in section Experimental flow page 18. Geometry specifications were bundled in Table 1 and fluid properties in Table 2. Building such geometry is heavy for a normal computer when the amount of spheres increases. By experience it seems Salome needs more time to make spheres than (extra) looping in a Python script. Therefore, it’s easier to make a Python script that efficiently creates automatically the whole geometry and groups boundary faces. Firstly, structured packings were generated as a simplification: simple cubic and cubic body centred (BCC). These scripts can be found in Appendix B. The first holds 48,4% of porosity or void space when adding spacing between spheres. Manipulation of the latter to spacing holds 35% like the experimental data of the article. Detailed calculation of porosity can be found in Appendix C. In reality spheres are randomly distributed inside a tube with fixed volume. For structured packings tortuosity is too low and unreal. Randomization gives a higher tortuosity. Next section explains how this randomization is achieved.

## Randomization of spherical packings

Possibilities concerning the randomization of a spherical packing depends on the porosity. With problems like start-up of clogging or crystallization of a compound, where a low amount of spheres is inside a larger volume, randomizing coordinates is quite easy for not-moving particles. Decreasing porosity gives less space between the spheres and makes it tougher to get extra random spheres that don’t overlap already added spheres. A script can be found in Appendix D where sphere coordinates are obtained with randomization of centroid coordinates when controlling the overlap or gap between spheres. This approach is limited and is interesting with high porous structures (porosity > 50%). Because of this limitation a DEM (Discrete Element Method) tool is used within the framework *Yade*. Within Yade creating a packing is automated and has two general possibilities:

1. Using **pre-build algorithms** like a high porous cloud (spherePack.makeCloud) and dense packing (spherePack.randomDensePack). But it is not possible to control and fix volume and porosity at the same time with these commands. Based on trial and error it is possible to reach a solution, but closely packed without overlaps by tension is an issue.
2. Using a **dynamical simulation** with forces that are applied on faces or volumes. At the end it is difficult to get a stress-free solution so overlaps of spheres are possible, because of tension. A script is made which can be found in Appendix E. This script generates a structured BCC packing inside a compact volume, uses mechanical engines to add harmonic motions to randomize (“shake” the volume), compresses to a wanted volume and exports a text file with the co-ordinates of the randomly distributed spheres, which can be read in Salome and should filter spheres too close to each other. Following Figure 7 gives the start and end from BCC to random packing, underneath a video where the whole mechanical evolution is simulated.

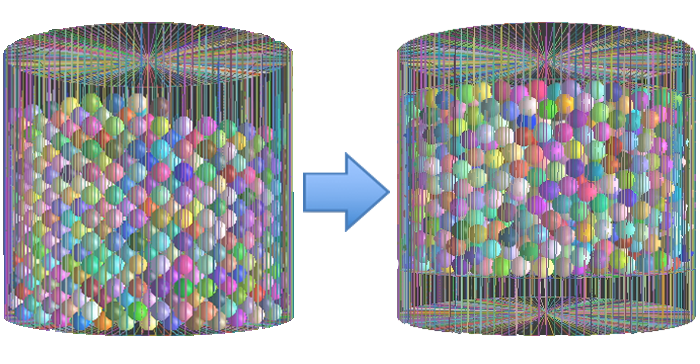
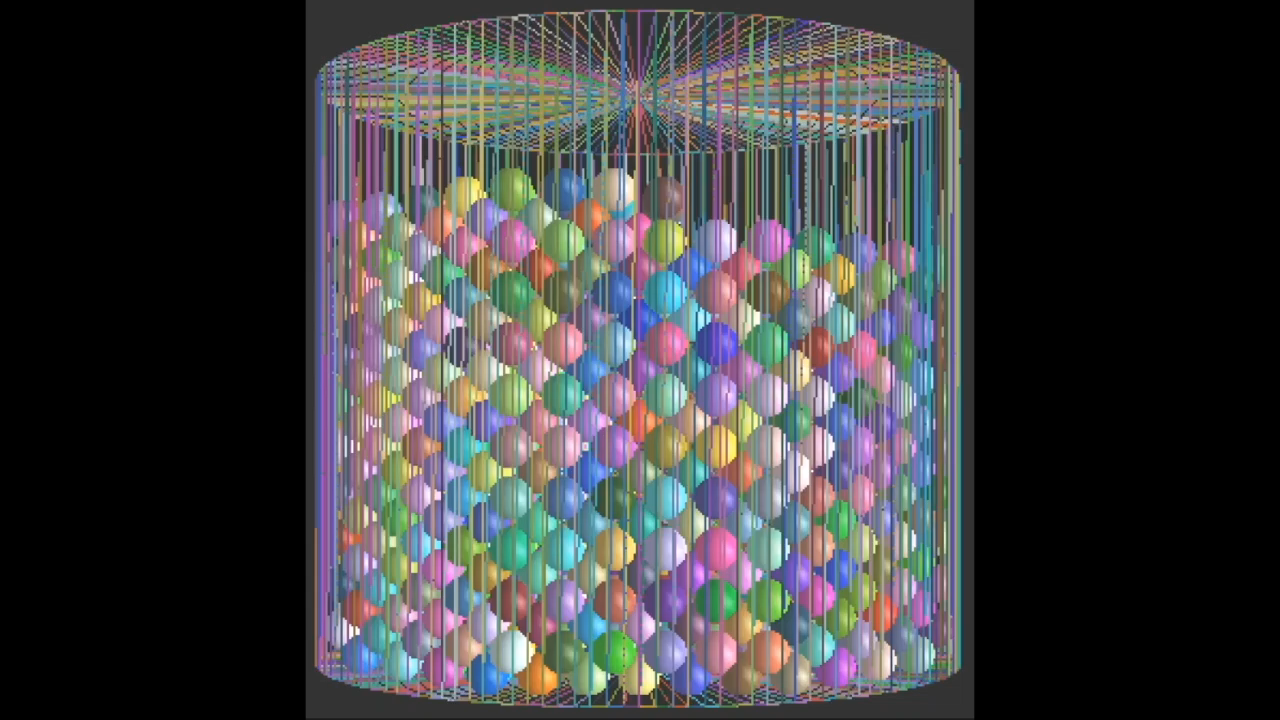


Figure 7: Randomization of BCC packing



## Building geometry and meshing

All geometries in this section are created with scripts loaded into Salome. The flexibility of the Python script allows us to create the geometry and group automatically for different sphere radii of the packing. Scripts can be run from a closable (*disown*) terminal window without graphical interface (-*t*) in background (*&*) with command “*salome –t name.py >log & disown*”. If spheres are too close to each other geometrical and mesh problems can follow, therefore sphere radii are adapted so an overlap () or gap () is created. (Filtering script.) There are different options to build a geometry depending on meshing choice. Following are the options ordered by decreasing complexity:

1. **Structured/unstructured**

For a structured mesh it is necessary to use *Divided Disk* to build a cylinder. This is a cylinder divided in sub-shapes, wherein it is possible to create a hexagonal mesh. To define the fluid domain spheres are cut. In this case one sphere needs to be cut out of several sub shapes. Internal faces and different cylinder zones can be seen in Figure 8, where a spherical packing of 15mm in diameter is taken. Decreasing the diameter or increasing amount of spheres in a fixed volume leads to problems of the cutting operation. Script can be found in Appendix F with extra information about the issues. Information about structured, unstructured meshing and building a script for a packed bed was found in topics on *cfd-online*. (Forum Salome-platform, 2015) Combining a structured, hexagonal mesh and unstructured, tetrahedral mesh needs a small trick. At the face between meshes there are pyramids that can't be treated by an UNV-file. Therefore, it is necessary to export the created mesh in Salome to a MED-file, import in gmsh, another external CAD software, and use the OpenFOAM command *gmshToFoam* instead of *ideasUnvToFoam* to add into an OpenFOAM case (Forum Salome-platform, 2009)

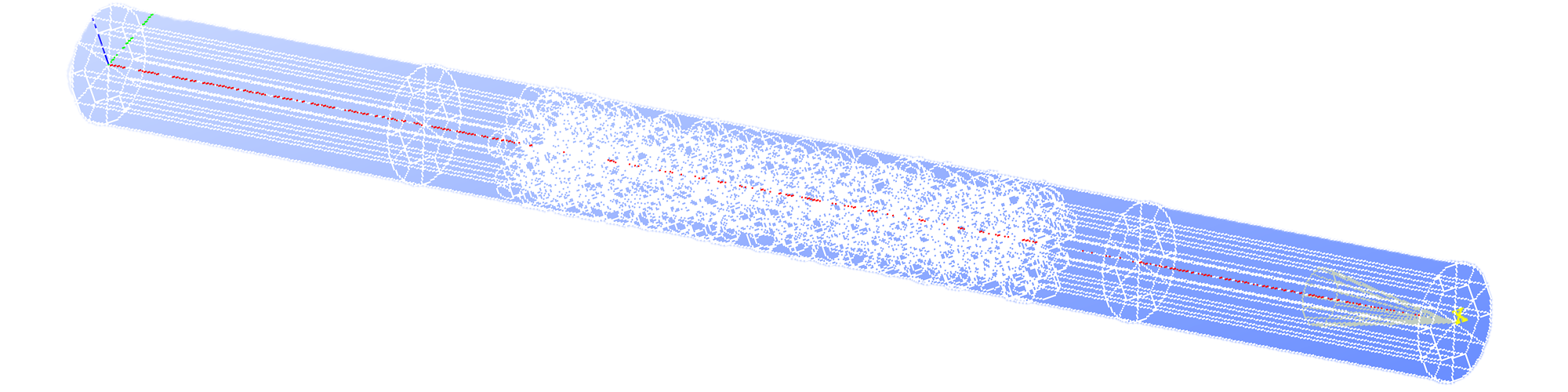


Figure 8: Geometry based on divided disk with 15mm packing

1. **Unstructured with viscous layer**

For an unstructured mesh there are no sub-shapes, but one cylindrical volume. This gives less internal faces so mesh algorithms have less difficulties. To add viscous layers (denser mesh closer to a boundary layer) it is necessary to have a transition zone between the inlet- and outletzone. Meshing needs to be separated into algorithms *Tetrahedron (Netgen)* and *Netgen 1D-2D* to define viscous layers. This separation leaded to problems.

1. **Unstructured without viscous layer**

The least complex approach has one cylindrical volume without transition region. Therefore, one meshing algorithm, *Netgen 1D-2D-3D*, can be used and reduces meshing errors. The script can be found in Appendix G. The whole geometry is built by periodic repetition of one tenth of the porous zone. This gives unreal interfaces. This geometry can be seen in Figure 9. Meshing was not achieved, because of narrow regions between the fused periodic parts.

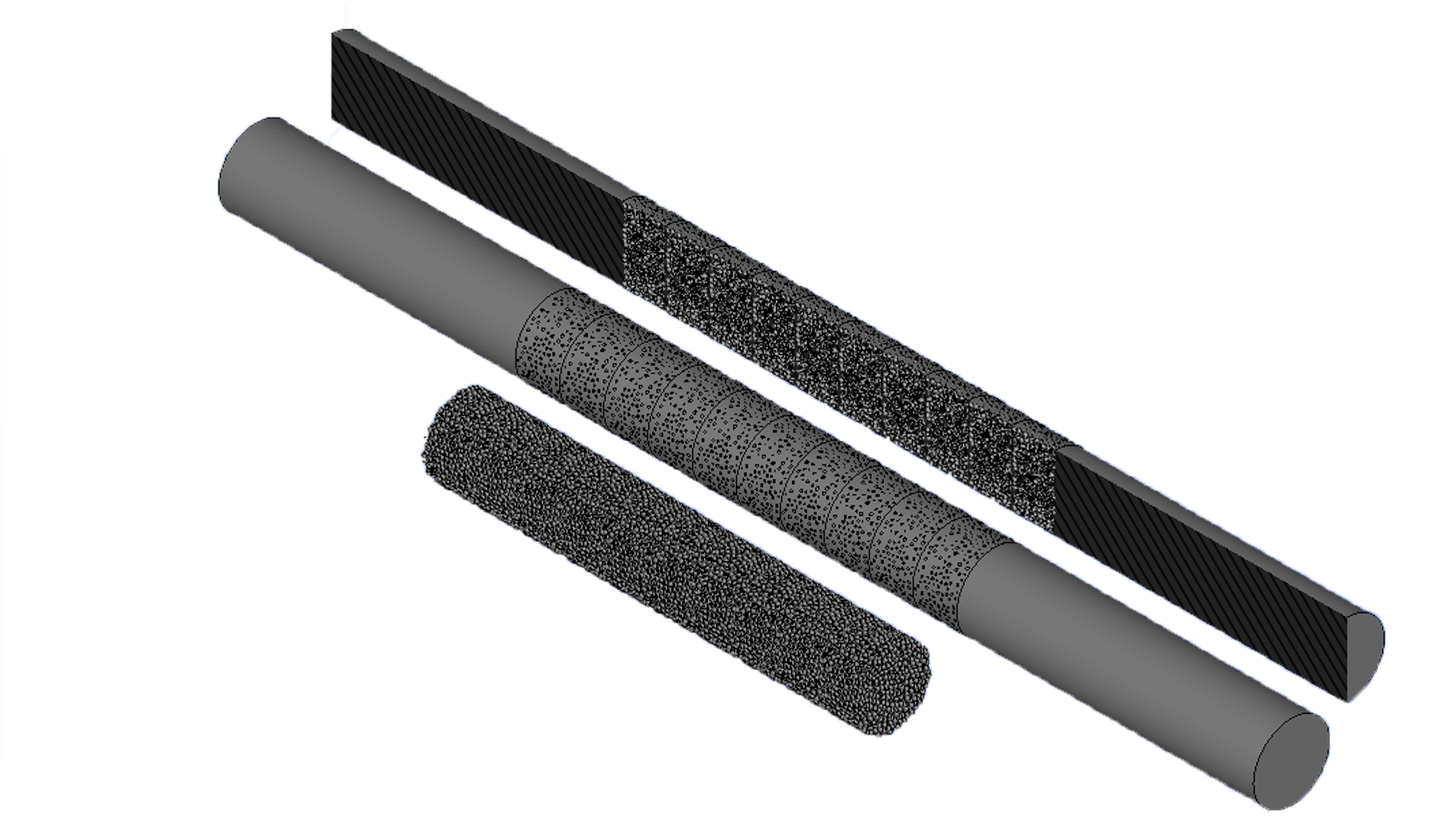


Figure 9: Geometry based on one cylindrical volume with 3mm packing

All the options are validated in next section Analytical control and validation of mesh to see if the error with respect to Hagen-Poiseuille tube flow increases (too much) with decreasing complexity. This analysis concludes that an unstructured mesh gives a rather good solution and will be used to calculate flow characteristics for the five cases.

## Analytical control and validation of mesh

Validation is always necessary to make a simulation more reliable. Comparison of analytical parameters and velocity profile needs to be done for a simple case (here: 3-dimensional tube flow). Chose of inlet velocity is based on the experimental data of Figure 5 from previous chapter. For the porous structure with 3-mm mono-sized spherical packing velocity 0,001 m/s lies in the Darcy-regime, where the reduced pressure is constant, because of the proportionality between pressure drop and velocity. (Dukhan, Bağcı, & Özdemir, 2014)

Analytical parameters

Calculation of analytical parameters is automated for different configurations for simple Hagen-Poiseuille tube flow in an excel sheet. Specific calculations of flow parameters are given below and summarized for all wanted cases in Table 6. Case choices could be found in Table 5 section Experimental flow.

with Hagen-Poiseuille (only for laminar flow) ∶

with friction losses (for laminar flow) over inlet zone:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Re | xfd,H (%) | umax (m/s) | (Pa/m) | f (-) | hf/L (-) |
| 51,4 | 66 | 0,002 | 0,012 | 1,25 | 1,24e-6 |
| 308,4 | 396 | 0,012 | 0,073 | 0,208 | 7,41e-06 |
| 693,9 | 892 | 0,027 | 0,164 | 0,092 | 1,67e-05 |
| 1079,4 | 1387 | 0,042 | 0,497 | 0,059 | 2,59e-05 |
| 6168 | 257 | 0,24 | 1,453 | 0,036 | 1,48e-04 |

Table 6: Summary analytical parameters of laminar fluid flow for inlet zone

After this analytical control one can make several conclusions:

* **xfd,H**

The length towards fully developed flow is given by a percentage of the inlet zone length. In nearly all cases the flow won’t achieve fully development before entering the porous zone. If it is necessary to have a fully developed flow before the porous zone the inlet length should be simulated longer as 0,2m or a coded inlet boundary layer should be used. But it should be pointed when looking to whole experimental setup the flow comes from a narrow tube to the inlet zone. As such the setup won’t achieve fully development either.

The pressure drop for the non-porous zones are at least 1000 magnitudes smaller compared to the pressure drop over the porous zone.

Intermezzo

It is interesting to note that for duct simulations the maximum velocity is 1,5 times inlet velocity. Making a 2-dimensional simulation (z-direction has 1 cell and top and button faces are empty) between two walls gives the results for a duct and not for tube flow. So simulations should use 3D mesh.

Validation mesh cell density and meshing algorithms

Using the sampleDict it is possible to export Ux, velocity in the flow direction, data points over a specific line. Then comparison of these data points to analytical flow profile gives information of whether the mesh grading needs to be finer in the flow direction or close to boundary layers. Several tetrahedral meshes are investigated and summarized in the following Table 7.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| maxSize | 0,0703764 | 0,00703 | 0,0035 |  | 0,00175 |
| minSize | 0,0161183 | 0,00161 | 0,0007 |  | 0,00035 |
| viscousLayers | no | no | yes |  | yes |
| quadrangles | no | no | no |  | no |
| non-orthogonality | 73 | 49,8 | 60,5 |  | 61,8 |
| umax | [0,00116; 0,00139] | [0,00173; 0.0019] | [0,00188; 0,00193 |  | [0,00192; 0,001935] |

Table 7: Tetrahedral meshes with its characteristics and velocity solution

Looking to non-orthogonality of tetrahedral meshes choice of schemes is very important. Depending in the mesh type and geometry schemes should been chosen differently. Unstructured, tetrahedral meshes can take time to converge. Therefore, it is important to use the “checkMesh” utility. This gives information about orthogonality, skewness and controls the mesh for errors. Depending on non-orthogonality different schemes with *limiters*[[8]](#footnote-8) and *non-orthogonalCorrectors* should be taken. (Guerrero, Introductory OpenFOAM® Course: Tips and tricks, 2015) Incorrect choice can lead to more instable calculation and slow convergence. The utility “*renumberMesh*” further decreases the amount of iteration needed.

Tetrahedral mesh is compared to octahedral mesh for tube flow and summarized in Table 8 and meshes are shown in Figure 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Uin (m/s)** | **umax (m/s)** | **xfd,H (m)** | **(Pa/m)** |
| Analytical | 0,001 | 0,002 | 0,132 | 0,012 |
| Octahedral | 0,001 | 0,001985 | 0,13 – 0,15 | ± 0,0127 |
| Tetrahedral | 0,001 | [0,00192; 0,001935] | 0,13 – 0,15 | ± 0,0132 |

Table 8: Comparison between analytical parameters and simulation obtained with octahedral and tetrahedral meshes

|  |  |
| --- | --- |
|  |  |

Figure 13: Validated mesh cell density for structured, unstructured without viscous layers

## Conclusion

After this part of making the geometry and meshing, several obstacles are tackled and some unsolved. Following two paragraphs try to explain some difficulties and the last paragraph concludes on how to go further.

Why is it a complex geometry?

Next summation gives several difficulties and reasons why a dense spherical packing is not that easy to handle:

* Spheres may not approach each other and the wall too closely. This is related to tolerance and which distances can be treated by computational calculation. Information about Salome development can be found in *Developer’s documentation*. (Salome Platform, 2007-2015)
* Boolean operations like *Cut* are difficult operations and depend on tolerance, interference and intersection. (Salome Platform, 2001-2010) Therefore, the cut tool can sometimes give trouble when cutting out a sphere to define the fluid domain. In python this is tackled by using try-except blocks.
* Packing is random and not structured therefore all possible outcomes need to be implemented.
* Salome can sometimes crash. Therefore, it is interesting to make brep.file backups during the calculation. These backups can be used as starting point when changing manually sphere radii. (see meshing problems)

How to solve manually mesh problems?

Mesh problems can always arise, mostly intersecting triangles or multiple edges. These (geometrical) errors seem random and the reason is not found. Therefore, these errors are solved manually by using “*bad mesh to group*” and “*bounding box*” or “*Modification – Add Node*” so the error can be located and assigned to a specific sphere. Then the following method is taken:

* Getting “*Centre of mass*” of specific sphere. If only a close by point is found the script “*findPoint.py*” can be used to get the closest centroid. Make a slightly bigger sphere and cut again. If the *Cut* operation succeeds, go to step 3 immediately.
* Localize the sphere and rerun script (starting from a brep.file backup) to build geometry and make specific sphere smaller so this sphere has no intersections with other boundaries.
* Rerun mesh algorithm “Netgen 3D-2D-1D”.

How to go further?

To start it is easier to simulate partly the porous zone without fusing parts together and normalise fluid properties. Tetrahedral meshes can give also good solutions and for the porous zone unstructured mesh is the only possibility. One should remember that the spherical packing is not totally comparable to reality, because of induced overlaps and gaps. Next chapters will try to achieve good simulation results for the five cases.

# Chapter III: Solving

When the mesh is generated the solving stage can start. This chapter gives information about the solvers used in OpenFOAM, how a solution is controlled by convergence criteria and schemes and takes a look at turbulency models.

## Prior to running solve algorithms

After the geometry is build and a mesh is created without errors within Salome solving can still be aborted, because of choice of schemes, correctors and “checkMesh” failures. It is very important to run “*checkMesh*” prior to solving. This utility makes sets of incorrect cells, like *zeroVolumeCells*. The cells can be deleted using “*setSet”*[[9]](#footnote-9) to define subsets and using “*subsetMesh”[[10]](#footnote-10)* to delete them. When using *Netgen 3D-2D-1D* it is possible to play with minSize/maxSize and remake mesh. A bad mesh will always give troubles so normally remaking is better. For example, in case of adding turbulence models mesh manipulation is more sensitive and can give solving abortion and mesh remake is necessary anyway. Alternatively, one can try how the solution quality differs when leaving checkMesh failures or deleting “incorrect” cells. For example, 4 too skew faces on a million cells won’t alter solution quality drastically (normally).

## Solvers

Depending on geometry there are two solvers used. For explicit porous structures (spherical packing) the solver simpleFoam is used. Using Darcy’s law and correction the solver porousSimpleFoam should be used and a “black box” geometry. More detailed description follows here:

1. **SimpleFoam**

Using simpleFoam solvers it is possible to have laminar Navier-Stokes or turbulent Navier-Stokes. Following equation is Navier Stokes and could have an extra term to describe turbulency effects, which will be discussed in section Laminar versus Turbulency models page 33.

1. **PorousSimpleFoam**

To use this solver there has to be a cellZone (porousZone) defined. It is even possible to define different cellZones with each a different set of porous parameters. PorousSimpleFoam can use two methods to obtain a source term for the momentum equation. Firstly, there is *DarcyForchheimer* that can define a local co-ordinate system to define permeability multidirectional for anisotropic structures. (Greenshield, 2014) Built-in solver can only define one viscous resistance and one inertial resistance term for each zone (CFD Support) Therefore, it would be interesting to include a mechanism for thresholds and changing of the constants. For this mechanism the solver will need to be adapted. Following equations show how the resistance vectors, and , are defined, with a and b coefficients of the Darcy polynomial:

and give the source term in three dimensions (CFD Support):

Secondly, the *powerLaw* method can used, which is isotropic and uses following source term with two empirical constants, C0 and C1. (CFD Support):

## Convergence control/criteria

Each iteration (or time step) OpenFOAM will give a “solution”. Depending on what tolerance the user gives OpenFOAM will stop its calculation. To check whether this solution is converged and is a good solution following control mechanisms are used for this thesis’ simulations:

* **Residuals**

With SIMPLE solvers it is possible to add *ResidualControl* to stop when specific residuals are reached for wanted variables. Only this won’t give a certainty on a good solution. Normally for steady solution residuals will decrease over iterations, but for unsteady solutions due to turbulency, changing boundaries, … residuals can increase as well. (Guerrero, 2015) During calculation one can add “*>log*” to save iteration steps into a text file. Then *Residual Plots* can be drawn using *gnuplot*. (tutorial Heydlauff, 2009) In Figure 14 a residual plot is shown. Steady solution’s residuals only decrease, but if runTimeModifiable is set on schemes can be changed during the calculation to more accurate (higher order) ones. Here grad(U) scheme was changed from *Gauss Linear* to *leastSquares* as example, but becomes rapidly divergent.

It is sometimes possible to reduce initial instability and oscillations by using initial p and U files from another case, for example when stepping from laminar to a turbulency model.

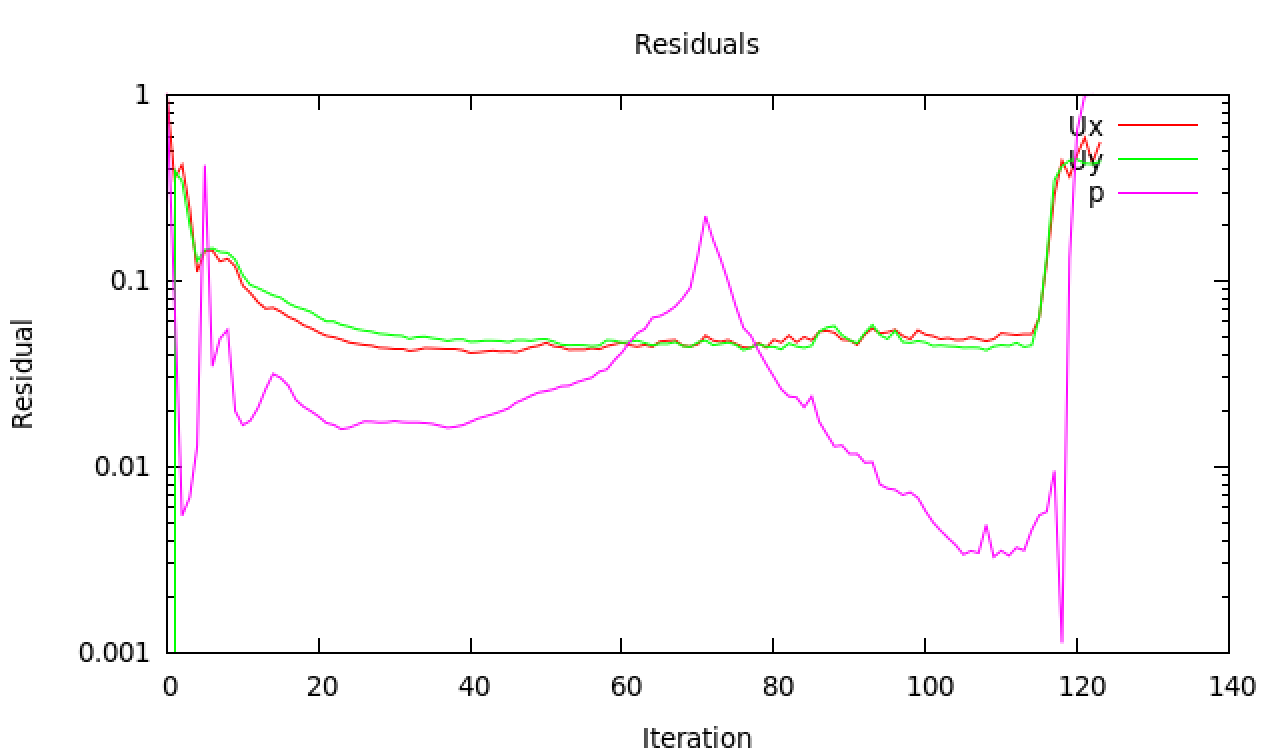


Figure 14: Ongoing residual plot of a simpleFoam simulation that diverges after scheme change

* **FunctionObjects**

With function objects in the *controlDict* it is possible to “postprocess” during run time. Tracking of variable values are possible to include in iteration steps with specific functions. Locally, variables could have unreal values. Therefore, it is interesting to use *FieldMinMax* to get minimum and maximum values of velocity and pressure. (OpenFOAM docu) Evolution of maximum values could also give information about convergence. Even better are *faceSource* functions. With such functions it is possible to define new areas that doesn’t have to be patches or cellZones and get variable values, averages or integrations. (OpenFOAM github, 2012) Immediately during run time over the porous zone can be tracked without need of waiting for full convergence and post processing steps. Adding following lines to the controlDict does the trick:

functions

{

outletPressurePlane

{

type faceSource;

functionObjectLibs ("libfieldFunctionObjects.so");

enabled true;

outputControl outputTime;

log true;

valueOutput true;

source sampledSurface;

sourceName bla;

surfaceFormat vtk;

sampledSurfaceDict

{

type plane; // always triangulated

basePoint (0.233 0 0);

normalVector (1 0 0);

}

operation areaAverage;

fields

(

p

);

}

}

It seems by experience that for highly non-orthogonal meshes for irregular randomly shaped structures the Residuals don’t all evolve to low values (order: 1e-4), but nevertheless property values can be stable and give the “correct” (simulation) value. As such high Residuals don’t mean necessarily a bad solution. (Guerrero, Tips and tricks, 2015)

## Laminar versus Turbulency models

Traditionally flow characteristics are divided between laminar and turbulent, because laminar Navies Stokes Equations (NSE) have an analytical solution, e.g. Poiseuille flow. In contrast to turbulent flows who are always based on empiricism. That’s why different turbulency models, within OpenFOAM, exist similar to all different correlations to characterize friction factor for example. Using simpleFoam gives the possibility of adding such turbulency models. There are various turbulency models that are advisable depending on the application. Different models are summed up and explained why it is (not) used for this dissertation:

* **RAS / RANS**

Generally, a stress term is added to the NSE to include turbulency effects:

and stress is generally expressed as two terms, where the turbulent term takes into account the variation with respect to the average velocity and defines the turbulent eddy viscosity, which is not a real physical property:

RAS simulations estimate the stress based on Reynolds averaging (Boussinesq Approximation) and gives the Reynolds-averaged stress (RAS) and Reynolds-Averaged Naviers Stokes (RANS):

Within the RAS group there are several methods such as k-Epsilon and k-Omega. Each one adds two transport equations to describe and . The k-Epsilon model adds two variables allowing to calculate the diffusion and convection effect of turbulence energy. Respectively the transported variables named dissipation of energy, and turbulent kinetic energy, k. (CFD Online) The k-Epsilon is normally for full turbulent flows, but because it is easy to implement it is still widely used. (CFD Online, 2010) Nevertheless, this last statement is not generally applicable and in some cases incorrect. k-Omega models introduce the variable , which is turbulency frequency or vorticity and works well for rotational flows. This model has more accuracy for near wall treatment and is better for low-Reynolds flow. (CFD Online)

Again: all these extra variables and equations serve to measure and .

Practically it is necessary to find appropriate initial and boundary conditions. Equations underneath help to get initial values (Guerrero, 2015), with *I* the turbulence intensity and viscosity ratio on :

For boundary conditions there are different options depending on the coarseness of the mesh: wall function models (nutWallFunction, nutRoughWallFunction, epsilonWallFunction, …), scalable wall functions or no wall function. (Greenshields, 2015) These functions are independent from the simulation, but influenced by the mesh. Defining a dimensionless wall distance:

with friction velocity and y the distance to the first cell centre normal to the wall. The friction velocity can be calculated and cinematic viscosity is (normally) fixed so yPlus () can only be influenced by the mesh size. Following equations help to get an initial estimation of yPlus and needed coarseness of the mesh.

The difference between the simulation and wall function (black box) is shown in Figure 15. Wall functions should not be used when .

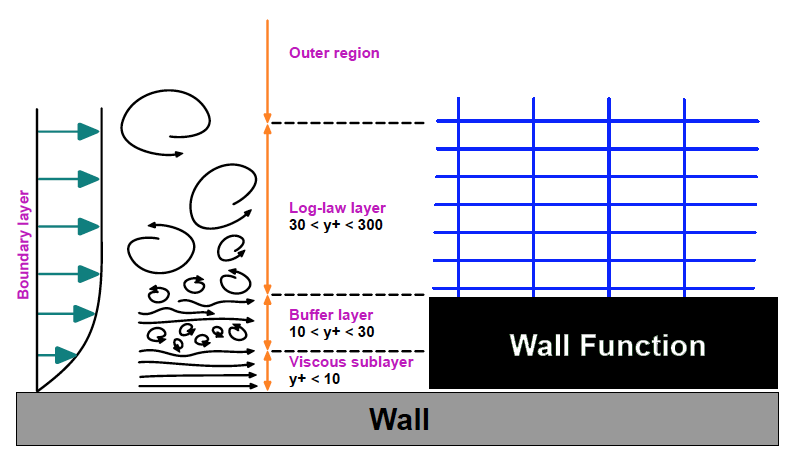


Figure 15: Wall function are black boxes for near wall treatment based on a model and not the simulation and span at least viscous sub layer and buffer layer (Guerrero 2015)

More information about boundary conditions for turbulency models can be found on CFD-Online (CFD-Online, 2014) and tools exist that help estimating all parameters.

(CFD-Online)

* **SST** (Shear Stress Transport)

This method combines different turbulence models. It formulates the k-ε behaviour for free flow and switches to k-ω for the inner parts of boundary layers. (Menter, 1994) It introduces around 10 extra coefficients and auxiliary relations. (CFD Online, s.d.) Therefore, the necessary computer power increases and slows down calculation time.

* **LES** (Large Eddy Simulation)

For very non-orthogonal meshes it is not advisable to use LES (Guerrero, Introductory OpenFOAM® Course: Tips and tricks, 2015). Using tetrahedral meshes is already non-orthogonal and introducing a complex porous structure will increase this non-orthogonality even further.

# Chapter V: Results and discussion

Building the whole 3-dimensional geometry of 28779 spheres (3 mm) inside a millitube is not easy as earlier discussed. This is in fact only a matter of computer capacity. Therefore, it is preferred in this dissertation to make 3-dimensionally only partly the porous zone and then normalise by length. After inlet boundaries there is always an entrance section where the flow is developing and a fully developed section. The flow collapses against the spheres and “searches” its new way through it. Simulating only partly the whole structure will give different fraction of importance of entrance section and fully developed section. Furthermore, after a porous zone the flow expulses into the outletzone and creates vorticity (unsteadiness). Firstly, it is interesting to “see” these phenomena in a 2D slice including the whole porous length and keeping these effects in mind for the 3D case.

## 2-dimensional

In 2D the flow cannot move up and down, but only in a plane. The upper and lower part are empty and do not hold any physical condition as such the flow feels less wall restrictions. Therefore, the flow acts like moving through two parallel walls. The number of possible paths gets drastically decreased and the flow needs to find its way in a very dense packed plane. As such when maintaining the porosity, it is not possible to permeate through the whole length. Therefore, sphere radius is decreased from 1,5 mm average to 1,4 mm average to build simulations for this part.

In Figure 16 and Figure 17 simulation for an inlet velocity of 0,001 m/s is showed, which gives 0,0015 m/s as maximum velocity between parallel plates. (Caplow) The turbulent zone after the porous region is still changing even tough residuals are steady. Pressure drop and maximum velocity magnitude are also steady. Only velocity components are responsible for the shown changes. over the porous zone equals to 0,214 kPa, which gives a reduced pressure drop of 7379 kPa s / m2. This is a lot more than the 3D experiments.

It can be concluded that the zone after the porous zone is already quite turbulent at lowest velocity case. Normally this part is of interest, because it alters the pressure measurement and turbulency models could help to simulate this part. Besides the possibility of high velocities within pores this is a second reason why turbulency models could help simulations of porous structures.

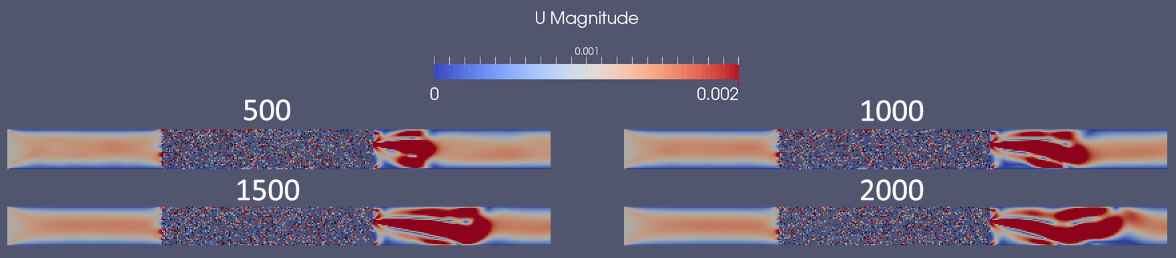


Figure 16: Velocity magnitude profile at iteration step 500, 1000, 1500 and 2000 for a parallel 2D porous plane

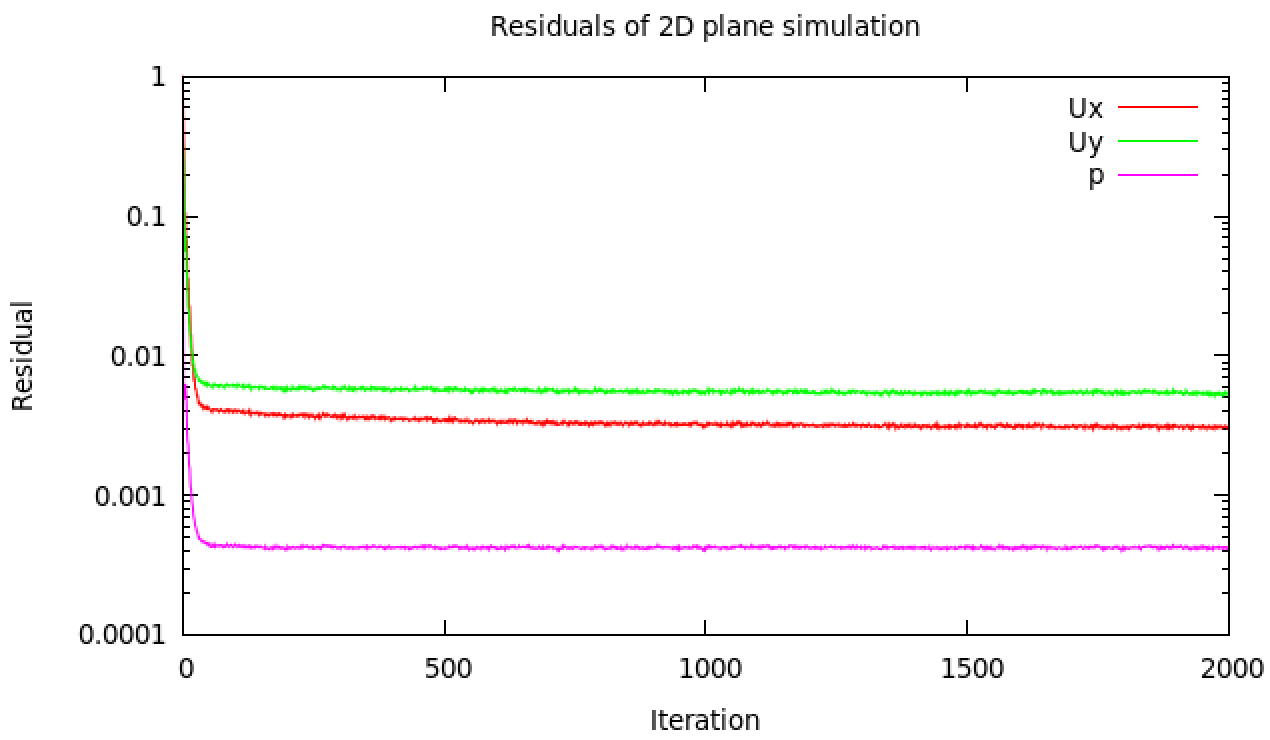


Figure 17: Residual plot for 2D porous plane

## Trial 1

Simulation are mostly an iterative process. After some results, one can find some necessities to improve the geometry or mesh. The conclusion of the first trial is that the geometry doesn’t represent enough the reality of the experiments. The mesh is important for the turbulency models () and showed for high-velocity cases better results. Next is normalisation discussed and the cases set in Experimental flow page 17.

### Normalisation

Like discussed in the introduction, normalisation won’t represent reality. The more spheres (longer simulated porous part) the better fraction entrance – fully developed represents reality. Following Table 9 gives the normalised pressure drop and Darcy permeability.

|  |  |  |  |
| --- | --- | --- | --- |
| # Spheres | ∆p/L | Simulation | Experiments |
| 479 spheres | 0,204 kPa/m | KD = 4,9e-9 m2 | KD = 6,18e-9 m2 (21% error) |
| 2877 spheres | 0,181 kPa/m | KD = 5,5e-9 m2 | KD = 6,18e-9 m2 (11% error) |

Table 9: Normalised pressure drop over a porous zone (1st case scenario: u=0,001m/s)

Calculation of reduced pressure in Darcy’s regime as experiments presented gives:

As such experiments gave a normalised pressure drop of 0,1618 kPa/m (case 1). The trend of decreasing normalised pressure gives a good perspective, but is still too high.

### Cases

Here simulation results are obtained for the different cases. Table 10 gives the results of the first trial where different RASModels are tried at different mesh sizes. The reduced pressure drop, between two cross sections, is always compared to the reduced pressure drop found by Dukhan et al. in their experiments.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| U (m/s) | RASModel | ε (%) | Mesh size | Reduced ∆p (Sim) kPa.m2/s | Reduced ∆p (Exp) kPa.m2/s |
| 0,001 | Laminar | 30-35 | Coarse | 181 | 161,8 |
|  |  | 30-35 | Medium | 186 | 161,8 |
|  |  | 30-35 | Fine | 193 | 161,8 |
| 0,006 | Laminar | 30-35 | Coarse | 258 | 161,8 |
| 0,0135 | Laminar | 30-35 | Coarse | 360 | 210 |
|  | kEpsilon | 30-35 | Medium | 360 | 210 |
|  | kEpsilon | 50 | Fine | 148 |  |
|  | kOmega | 50 | Fine | 74 |  |
| 0,021 | Laminar | 30-35 | Coarse | 437 | 251 |
| 0,12 | Laminar | 30-35 | Coarse | 1972 | 739 |
|  | kOmega | 30-35 | Medium | 1369 | 739 |
|  | kOmega | 30-35 | Coarse |  |  |
|  | kEpsilon | 50 | Fine | 340 |  |

Table 10: Simulation results for different cases (Trial 1)

Following points give conclusions after investigation of the results:

* Geometry and porosity

For randomization of the bed (Randomization of spherical packings) a dynamical simulation was made. To achieve to wanted porosity the bed got compressed at the end. It seems *Yade* compressed slightly too much even though script did not contain errors. The amount of spheres is too much. To oppose this slight over-compression 2797 spheres will be taken in trail 2 instead of 2788. To motivate the normalisation also 1/5th will be tried.

Because of tolerance, there are induced gaps and overlaps. It is possible that more overlaps were created as such lowering the possible paths, which increases pressure drop for the simulation. The filtering script is revisited and a good threshold would be maximum ± 5% Rs interval of touching to get a good interchange between gap and overlap.

Looking to Blake-Kozeny and Ergun equations the pressure is highly dependent on the porosity (power 2 -or 3). Results are very sensitive for incorrect porosity and therefore geometry is very important and should represent reality correctly to get good results.

* Mesh

Up to Forchheimer regime it seems laminar models are sufficient to give appropriate results. At high velocity range introducing turbulency models can give quite some difference. The boundary walls of the mesh are divided in inletWalls, porousWalls, sphereSurfaces and outletWalls. In this way fineness of the mesh could be manipulated close to each patch. The yPlus value was not always correct, because it should be between 30 and 300 to use wall function.

* Packing

Another possibility of incorrect representation could be the packing. The method taken is static and uncoupled from DEM. Spheres could be still flexible and move slightly which decreases the pressure needed. But for such a dense packing () this movement could be considered rather small. If needed it would be possible to make CFDEM simulations to couple fluid flow and particles.

* Measurement

As seen the turbulency after the porous zone can alter the measurement. It is important to exactly know how the measurement are taken and if this turbulency effect is included in the measurement.

## Trial 2

## Comparison explicit and implicit method

Degree of mixing

Ease of simulation

Correctness of different parameters

Turbulency and vortices after the bed

Sensitivity of correct geometry representation

Possibilities to couple with other models (reaction etc.)?

porousSimpleFoam and onset for different regimes (adapting solver)

## What’s next?

## Conclusion

# Bibliography

Brown, T. L., H. Eugene LeMay, J., Bursten, B. E., Murphy, C. J., & Woodward, a. P. (2012). Sample Exercise 12.1 Calculating Packing Efficiency. In *Chemistry, The Central Science, 12th Edition.* Pearson.

Caplow, T. (n.d.). *Poiseuille Flow Between Parallel Plates (Laminar).* Retrieved from http://www.columbia.edu/: http://www.columbia.edu/cu/gsapp/BT/RESEARCH/Arch-atmos/plates.html

CFD Online. (2010, April 28). *Use of k-epsilon and k-omega Models.* Retrieved from CFD Online forum: http://www.cfd-online.com/Forums/main/75554-use-k-epsilon-k-omega-models.html

CFD Online. (n.d.). *cfd-online.com*. Retrieved May 11, 2016, from SST k-omega model: http://www.cfd-online.com/Wiki/SST\_k-omega\_model

CFD Online. (n.d.). *K-epsilon models.* Retrieved March 30, 2016, from CFD Online: http://www.cfd-online.com/Wiki/K-epsilon\_models

CFD Support. (n.d.). *Defining porosity using Darcy law.* Retrieved May 6, 2016, from cfdsupport.com: http://www.cfdsupport.com/OpenFOAM-Training-by-CFD-Support/node223.html

CFD Support. (n.d.). *Defining porosity using Power law.* Retrieved May 6, 2016, from cfdsupport.com: http://www.cfdsupport.com/OpenFOAM-Training-by-CFD-Support/node224.html

CFD-Online. (2014, January). *Turbulence free-stream boundary conditions.* Retrieved from CFD Online Wiki: http://www.cfd-online.com/Wiki/Turbulence\_free-stream\_boundary\_conditions

CFD-Online. (n.d.). *Turbulence Properties, Conversions & Boundary Estimations.* Retrieved from CFD Online Tools: http://www.cfd-online.com/Tools/turbulence.php

Chachuat, B. (2015, December 10). Multi-scale Modeling of Light-limited Growth in Microalgae Production Systems. Leuven.

Dukhan, N., Bağcı, Ö., & Özdemir, Ö. (2014, June 14). Experimental flow in various porous media and reconciliation of Forchheimer and Ergun relations. *Elsevier Science Direct*, 9.

Forum Salome-platform. (2009, August). *Openfoam conversion*. Retrieved from Salome-platform: http://www.salome-platform.org/forum/forum\_10/thread\_3005

Forum Salome-platform. (2015, March). *How to mesh a cylindrical tube packed randomly with contacting spheres*. Retrieved from salome-platform: http://salome-platform.org/forum/forum\_10/950427366

Greenshield, C. (2014, February 17). *OpenFOAM 2.3.0: Physical Modelling; Co-ordinate System Specification.* Retrieved May 6, 2016, from The OpenFOAM Foundation: http://openfoam.org/release/2-3-0/physical-modelling/

Greenshields, C. (2015, March 1). *OpenFOAM User Guide: 7.2 Turbulence models.* Retrieved April 30, 2016, from CFD Direct: http://cfd.direct/openfoam/user-guide/turbulence/

Guerrero, J. (2015, July 13-17). *Introduction Course OpenFOAM: Steady and Unsteady*. Retrieved from University of Genoa, DICCA: http://www.dicat.unige.it/guerrero/oftraining/6\_4fvm\_transient\_simulations.pdf

Guerrero, J. (2015, July 13-17). *Introductory OpenFOAM® Course: Tips and tricks.* Retrieved February 10, 2016, from University of Genoa, DICCA: http://www.dicat.unige.it/guerrero/oftraining/9tipsandtricks.pdf

Guerrero, J. (2015, July 13-17). *Introductory OpenFOAM® Course: turbulence modeling*. Retrieved from University of Genoa, DICCA: http://www.dicat.unige.it/guerrero/oftraining/8\_1OF\_advanced\_modeling\_turbulence.pdf

Harvard Corporation. (2016). *Surface & Depth Filter Types.* Retrieved May 25, 2016, from Harvard Corporation: http://www.harvardcorp.com/filtration-process/surface-depth-filters.html

Heydlauff, W. (2009, April 30). *Tutorial of how to plot residuals !* Retrieved from cfd-online.com: http://www.cfd-online.com/Forums/openfoam-solving/64146-tutorial-how-plot-residuals.html

Ingham, D., & Pop, I. (1998). In D. B. Ingham, & I. Pop, *Transport Phenomena in Porous Media.* Oxford: Elsevier.

King, H. (2005-2016). *Hydraulic Fracturing of Oil & Gas Wells Drilled in Shale.* Retrieved from Geoscience News and Information: http://geology.com/articles/hydraulic-fracturing/

Masashi, M., Fumio, W., & Toshihiko, K. (1980). *US Patent No. US 4368144 A.*

Menter, F. (1994). In *Two Equation Eddy-Viscosity Turbulence Models for Engineering Applications.* California: AIAA Journal. Retrieved from http://www.cfd-online.com/Wiki/SST\_k-omega\_model

Muljadi, B. P., Blunt, M. J., Raeini, Q. A., & Bijeljic, B. (2015, June 13). The impact of porous media heterogeneity on non-Darcy flow behaviour from pore-scale simulation. *ScienceDirect*, 12.

National Academy of Sciences. (2000). *Catalytic Process Technology.* Retrieved from NAP OpenBook: http://www.nap.edu/read/10038/chapter/4

OpenFoam. (2015, May 21). *User Guide Version 2.4.0.* Retrieved from CFD Direct: the architects of OpenFOAM: http://cfd.direct/openfoam/user-guide/

OpenFOAM docu. (2011-2016). *fieldMinMax Class Reference Field function objects*. Retrieved from OpenFOAM documentation: http://openfoam.com/documentation/cpp-guide/html/a00802.html

OpenFOAM github. (2012, March). *faceSource: add area-normal integration or averaging*. Retrieved from github.com/OpenFOAM: https://github.com/OpenFOAM/OpenFOAM-2.1.x/blob/master/src/postProcessing/functionObjects/field/fieldValues/faceSource/faceSource.H

OpenFOAM wiki. (n.d.). *SetSet.* Retrieved May 6, 2016, from Unofficial OpenFOAM wiki: https://openfoamwiki.net/index.php/SetSet

Pandey, A. (2003, March). Solid-state fermentation. *Elsevier*, 81-84.

Rhodes, M. (2008). Fluid Flow Through a Packed Bed of Particles. In M. Rhodes, *Introduction to Particle Technology – Second Edition* (Vol. 6, p. 466). Monash University, Australia: John Wiley & Sons Ltd.

Salome Platform. (2001-2010). *General Fuse Algorithm, Partition Algorithm, Boolean Operations Algorithm Backgrounds.* Retrieved February 9, 2016, from SALOME : The Open Source Integration Platform for Numerical Simulation: http://docs.salome-platform.org/7/gui/GEOM/SALOME\_BOA\_PA.pdf

Salome Platform. (2007-2015). *Developer's Documentation.* Retrieved February 9, 2016, from Salome Platform Documentation: http://docs.salome-platform.org/7/

Smets, I. (2015, November). Solid State Fermentation. *Microbial Process Technology*. Leuven, Belgium.

Soulaine, C. (2015). On the origin of Darcy's law. In C. Soulaine. Stanford: Stanford.edu.

Sparkes, M. (2014, Jan 13). Supercomputer models one second of human brain activity. *The Telegraph*, 1.

Taherzadeh, D., Picioreanu, C., & Horn, H. (2012, April 4). *Mass Transfer Enhancement in Moving Biofilm Structures.* Retrieved from National Center for Biotechnology Information: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3318122/

Validyne Engineering. (n.d.). *General Operating Instructions of Validyne Variable Reluctance Pressure Transducer.* Retrieved from http://validyne.com/query/DP15\_Variable\_Reluctance\_Pressure\_Sensor\_Capable\_of\_Range\_Changes

Van Der Bruggen, B., Vandecasteele, C., Van Gestel, T., Doyen, W., & Leysen, R. (2003). A review of pressure-driven membrane processes in wastewater treatment and drinking water production. *Amer inst chemical engineers*, 46. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/ep.670220116/abstract

Versteeg, H. K., & Malalasekera, W. (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method 2nd edition.* Harlow, Essex, England: Pearson.

# Appendices

## General computational procedure to characterise property distributions

Computational Fluid Dynamics (CFD) characterises fluid flows. This section gives a general procedure how property distributions are evaluated (e.g. velocity and pressure distributions). The procedure follows three steps: pre-processing, solving and post-processing. A complete introduction about simulation and its steps can be found in the book: *An Introduction to Computational Fluid Dynamics: The Finite Volume Method 2nd edition* (Versteeg & Malalasekera, 2007)

**Pre-processor**

Pre-processing consists of making the geometry and mesh generation. This can be done by an OpenFOAM build-in mesh generator blockMesh or by external software like Salome, version 7.6, where the possibility is to make a mesh using the Salome modules or loading a python script. For all geometries made Salome is used. Meshing divides the full domain into discrete control volumes or mesh cells.

**Solver**

This part describes how Partial Differential Equations (PDEs) that describe transport phenomena (nature laws) are solved. Internally the solver integrates and discretises the equations to get finally the solution.

Integration

Finite volume method uses the transport equation integrated over a finite control volume as starting point for computational procedures. (Versteeg & Malalasekera, 2007)

Discretisation

Integration of a specific transport equation at a nodal point yields a discretised equation, wherein property and (proportional) coefficient are evaluated at the nodal point and gradients linearly approximated. For parameters at cell boundary, surrounding the nodal point, the average between the parameter's value at the two adjacent nodal points is taken (central differencing method). All of this gives a discretised, algebraic form. For convection problems the method is different, because it spreads influence in the flow direction and not in all directions like diffusion. For convective fluxes property values are necessary at faces between adjacent nodes. Calculating of the value of transported properties at control volume faces can give trouble. Therefore, there are different discretisation schemes (QUICK, power-law and differencing schemes) to obey conservativeness, boundedness (convergence of iteration) and transportiveness (Peclet Number). (example eq. 4.10 book of:) (Versteeg & Malalasekera, 2007)

#### Solution

Matrix solution methods are designed to solve the set of discretised, algebraic equations of each node to obtain a distribution of a property. (examples section 4.3 and 4.6 book of:) (Versteeg & Malalasekera, 2007) For transient solvers the Courant number is important. For steady state solvers, e.g. simpleFoam, Courant is meaningless. This number gives how many mesh cell a particle has “seen” during a time step. For Co higher than 1 the Lagrangian path of a particle is longer than a mesh cell size. This gives a numerical instability. Following the definition: . OpenFOAM has different build-in solver. The important solvers used for this dissertation can be found in following Table 6.

|  |  |
| --- | --- |
| Solver | Description |
| icoFoam | Transient solver for incompressible, laminar flow of Newtonian fluids |
| simpleFoam | Steady-state solver for incompressible, turbulent flow |
| porousSimpleFoam | Steady-state solver for incompressible, turbulent flow with implicit or explicit porosity treatment |
| porousInterFoam | Solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface  capturing approach, with explicit handling of porous zones |

Table 11: Used solvers within this papers that are needed for simulation of porous structures (OpenFoam, 2015)

**Post-processor**

Validation of the code requires detailed knowledge of boundary conditions, used solver (fvSolution, fvSchemes) and experimental data. Important is to validate firstly the geometry and mesh. A post-processing tool is paraFoam.

OpenFOAM has several interesting command to simplify some operations. A summary can be found in Table 7.

|  |  |  |
| --- | --- | --- |
| Command | Procedure Step | Definition |
| refineMesh | After mesh generation | makes a mesh finer (better flexibility in Salome) |
| mirrorMesh | After mesh generation | mirrors a mesh when a mirror axis is defined in mirrorMeshDict |
| transformPoints | After mesh generation | scales each dimension with (different scalar) |
| checkMesh | After mesh generation | checks mesh and prints warnings and errors |
| Co | Post-processing | prints Courant number for each written timestep |
| foamCalc | Post-processing | calculates for example the different components of U |
| sample | Post-processing | generates matrices of property distributions when a sampleDict is defined |

Table 12: Interesting terminal commands within OpenFOAM

## Structured packing

Following two subparagraphs are pieces of code to add to a script to create respectively a simple cubic and bcc packing.

Simple Cubic Packing

#parameters

Rc=0.0514/2

Rs=0.005/2

Li=0.2

Lp=0.304

Lo=0.2

#Transition from structured mesh to position of sphere (not too abrupt)

Transition = 0.015

#number of loop steps to have enough spheres

lengthSteps = int(Lp//(2\*Rs))

otherSteps = int((2\*Rc)//(2\*Rs))

# adding all spheres in an array

# centres can be used for extra calculations like finding faces closest to sphere centre (Sphere surface)

centres = []

spheres = []

#range (3,5) gives 3 and 4; so end in not included

#Because of adding space between spheres: lengthStep (Otherwise sphere outside zone)

#Removing spheres outside of packedZone: otherSteps +2 for beginning because beginning is included and -1 for end point

#adapted steps is not necessary because of if-test

#Add loop to add all spheres

for i in range (2,3) :

for k in range (-otherSteps+2, otherSteps-1) :

for l in range (-otherSteps+2, otherSteps-1) :

#Meshing problem: solved by adding 2% of Rs space between adjacent spheres

#coordinates of new sphere

x = Li+i\*2.01\*Rs

y = k\*2.01\*Rs

z = l\*2.01\*Rs

#Don't make the sphere if it will be outside the tube

#distance from tube centre to new sphere centre:

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs :

#Translation = geompy.MakeTranslation(Sphere\_1,i\*2\*Rs+0.01\*Rs,k\*2\*Rs+0.01\*Rs,l\*2\*Rs+0.01\*Rs)

p = geompy.MakeVertex(x,y,z)

sphere = geompy.MakeSpherePntR(p, Rs)

geompy.addToStudy(sphere, 'p'+str(i)+str(k)+str(l))

spheres.append( sphere )

centres.append( p )

pass

pass

pass

BCC packing

#parameters

Rc=0.0514/2

Rs=0.015/2

Li=0.2

Lp=0.304

Lo=0.2

#Transition from structured mesh to position of sphere (not too abrupt)

Transition = 0.015

#number of loop steps to have enough spheres

lengthSteps = int(Lp//(2\*Rs))

ySteps = int((2\*Rc)//(2\*Rs))

zSteps = int(ySteps//math.sqrt(3))

# adding all spheres in an array

# centres can be used for extra calculations like finding faces closest to sphere centre (Sphere surface)

centres = []

spheres = []

#A layer

for i in range (0,lengthSteps) :

for k in range (-ySteps,ySteps) :

for l in range (-zSteps,zSteps) :

#Meshing problem: solved by adding 2% of Rs space between adjacent spheres

#coordinates of new sphere

x = Li+i\*2.01\*Rs

y = k\*2.01\*Rs

z = l\*2.01\*Rs\*math.sqrt(3)

#Don't make the sphere if it will be outside the tube

#distance from tube centre to new sphere centre:

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs :

#Translation = geompy.MakeTranslation(Sphere\_1,i\*2\*Rs+0.01\*Rs,k\*2\*Rs+0.01\*Rs,l\*2\*Rs+0.01\*Rs)

p = geompy.MakeVertex(x,y,z)

#geompy.addToStudy(p, 'p'+str(i)+str(k)+str(l))

sphere = geompy.MakeSpherePntR(p, Rs)

spheres.append( sphere )

centres.append( p )

#B layer

for i in range (0,lengthSteps) :

for k in range (-ySteps,ySteps) :

for l in range (-zSteps,zSteps) :

x = Li+i\*2.01\*Rs

y = (1+k\*2.01)\*Rs

z = (1+l\*2.01)\*Rs\*math.sqrt(3)

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs :

p = geompy.MakeVertex(x,y,z)

sphere = geompy.MakeSpherePntR(p, Rs)

spheres.append( sphere )

centres.append( p )

## Calculation of void space for spherical packing

This appendix looks to the calculation of two arrangements of spherical packing. The first is called *primitive cubic*, which is a repetition of one A-layer, and the second *body-centred cubic,* which repeats alternating A- and B-layers. Normally in these arrangements spheres are touching, but to avoid (meshing) problems it is better to add an additional spacing, , between the spheres. Following the calculation of the porosity :

Primitive cubic

where

where

Body-centred cubic

Trigonometry (Pythagoras) teaches us that side length of the cube, is:

then one can calculate the rest:

where

where is so that porosity equals to 35% (in a box):

Body centred cubic is closer to a random spherical packing. The experimental data of (Dukhan, Bağcı, & Özdemir, 2014) have a porosity of 35%. Therefore, the last calculation looks how much additional spacing needs to be added between the spheres to obtain this porosity. It is important to note that not only porosity determines fluid flow properties. Fluid flow characteristics are different when the amount of possible paths and narrowness is different. (Dukhan, Bağcı, & Özdemir, 2014)

More examples and information about calculation of packing efficiency, crystal structures and ionic structures can be found in a typical handbook like *Chemistry, the Central Science*. (Brown, H. Eugene LeMay, Bursten, Murphy, & Woodward, 2012)

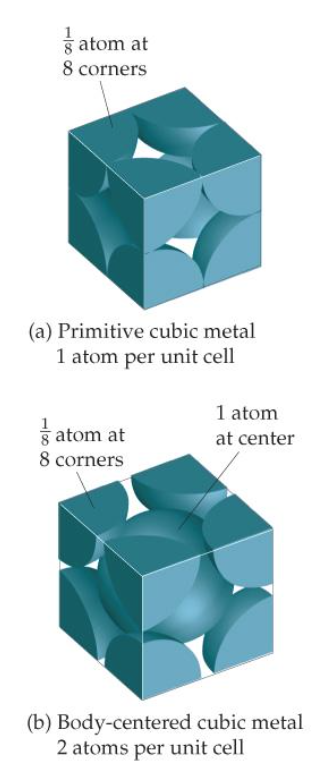


Figure 18: Arrangement spherical packing (Brown, H. Eugene LeMay, Bursten, Murphy, & Woodward, 2012)

## Randomization based on random coordinates

import math

import numpy

import random

#parameters

Rc=0.0257

Rs=0.0005

Li=0.2

Lp=0.304

Lo=0.2

Transition = 0.015

centres = []

spheres = []

#parameters

eps = 0.35

gap = -0.05\*Rs

Vp = Lp\*math.pi\*Rc\*\*2

Vs = 1.33333333333333333\*math.pi\*Rs\*\*3

Lpi=4\*Rs

steps=int(Lp/Lpi)

Vpi=Lpi\*math.pi\*Rc\*\*2

amount = int((1-eps)\*Vp/Vs)

amounti = int((1-eps)\*Vpi/Vs)

#3963\*(1-eps)

# radialstepgenerator

radialsteps = []

bla = []

bla.append(0)

k = 0

radialsteps.append(k)

#step at least 20 spheres

while k < 1 :

if k < 0.5 :

k = k + 0.05

elif k < 1 :

k = k + 0.025

else :

k = k + 0.05

radialsteps.append(k)

#looping until amount of spheres is reached

for step in range (0,steps) :

amountir = 0

amountirlast = 0

bla[0] = step

numpy.savetxt('centresTemp.txt', bla)

for iterator in range (1, len(radialsteps)) :

i=0

iterations = 0

rcupper=radialsteps[iterator]\*Rc

rclower=Rc\*(radialsteps[iterator-1])

if radialsteps[iterator] != radialsteps[len(radialsteps)-1] :

rcmoreupper=Rc\*(radialsteps[iterator+1])

else :

rcmoreupper = 0

Vpir=Lpi\*math.pi\*rcupper\*\*2-Lpi\*math.pi\*rclower\*\*2

Vpirmoreupper=Lpi\*math.pi\*rcmoreupper\*\*2-Lpi\*math.pi\*rcupper\*\*2

amountirlast = amountir

amountir = int(amounti\*Vpir/Vpi)

amountirnext = int(amounti\*Vpirmoreupper/Vpi)

while i < amountir :

#creating random coordinates

x = Li+random.uniform(step\*Lpi,(step+1)\*Lpi)

r = random.uniform(radialsteps[iterator-1]\*Rc,radialsteps[iterator]\*Rc)

theta = random.uniform(0,2\*math.pi)

y = r\*math.cos(theta)

z = r\*math.sin(theta)

if iterations < 2\*amountir :

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs-gap :

add = 1

dist = 0

startrange = len(centres)-i-amountirlast

#check if new sphere overlaps with spheres in own cell or neighboring

for index in range(startrange,len(centres)) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

iterations=iterations+1

break

if step != 0 and rcmoreupper!=0:

for index in range(len(centres)-i-amountirlast-amounti-2,len(centres)-i-amounti+amountir+amountirnext+2) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

iterations=iterations+1

break

elif step != 0 and rcmoreupper==0 :

for index in range(len(centres)-i-amountirlast-amounti-2,len(centres)-i-amounti+amountir+2) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

iterations=iterations+1

break

if add == 1 :

centres.append([x,y,z])

i = i + 1

numpy.savetxt('centresInDisksTemp.txt', centres)

else :

#too much iteration; changing coordinates of already added spheres

change = int(i\*0.1)

iterations = 0

changeposition = len(centres)-1-random.randint(0,i)

for x in range (0,change) :

x = Li+random.uniform(step\*Lpi,(step+1)\*Lpi)

r = random.uniform(radialsteps[iterator-1]\*Rc,radialsteps[iterator]\*Rc)

theta = random.uniform(0,2\*math.pi)

y = r\*math.cos(theta)

z = r\*math.sin(theta)

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs-gap :

add = 1

dist = 0

startrange = len(centres)-i-amountirlast

for index in range(startrange,len(centres)) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

break

if step != 0 and rcmoreupper!=0:

for index in range(len(centres)-i-amountirlast-amounti-2,len(centres)-i-amounti+amountir+amountirnext+2) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

break

elif step !=0 and rcmoreupper==0 :

for index in range(len(centres)-i-amountirlast-amounti-2,len(centres)-i-amounti+amountir+2) :

dist = math.sqrt((x-centres[index][0])\*\*2+(y-centres[index][1])\*\*2+(z-centres[index][2])\*\*2)

if dist < 2\*Rs+gap :

#don't add

add = 0

break

if add == 1 :

centres[changeposition]=[x,y,z]

#end of while loop

#Centres of random packing generated

numpy.savetxt('centresCompletelyRandom.txt', centres)

## Randomization based on DEM (Yade)

import numpy

import math

#parameters

Rc=0.0514/2

Rsreal=0.0015

Rs=1\*Rsreal

Li=0.2

Lp=0.0304

Lo=0.2

volumep = Lp\*math.pi\*Rc\*\*2

volumes = 1.3333333333\*math.pi\*Rsreal\*\*3

eps = 0.355

amount = int(((1-eps)\*volumep)/volumes)

# import yade modules that we will use below

from yade import utils, export

# create cylindre and two movable faces to compress

body = O.bodies.append(geom.facetCylinder(center=(0,0,0.75\*Lp),radius=Rc,height=1.5\*Lp,segmentsNumber=150))

top = O.bodies.append(geom.facetCylinder(center=(0,0,1.5\*Lp),radius=Rc,height=0,segmentsNumber=150))

bottom = O.bodies.append(geom.facetCylinder(center=(0,0,0),radius=Rc,height=0,segmentsNumber=150))

#create dense regular packing

lengthSteps = 2\*int(Lp//(4\*Rs/math.sqrt(3)))

ySteps = 2\*int((2\*Rc)//(4\*Rs/math.sqrt(3)))

zSteps = ySteps

added = 0

for i in range (0,lengthSteps) :

for k in range (-ySteps,ySteps) :

for l in range (-zSteps,zSteps) :

x = Rs+(2+i\*4)\*Rs/math.sqrt(3)

y = (2+k\*4)\*Rs/math.sqrt(3)

z = (2+l\*4)\*Rs/math.sqrt(3)

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs and added < amount:

O.bodies.append(utils.sphere((z,y,x),Rs))

added=added+1

x = Rs+i\*4\*Rs/math.sqrt(3)

y = k\*4\*Rs/math.sqrt(3)

z = l\*4\*Rs/math.sqrt(3)

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

if centreToCentre < Rc-Rs and added < amount:

O.bodies.append(utils.sphere((z,y,x),Rs))

added=added+1

#apply motion and compression

O.engines=[

ForceResetter(),

InsertionSortCollider([Bo1\_Sphere\_Aabb(),Bo1\_Facet\_Aabb()]),

InteractionLoop(

# handle sphere+sphere and facet+sphere collisions

[Ig2\_Sphere\_Sphere\_L3Geom(),Ig2\_Facet\_Sphere\_L3Geom()],

[Ip2\_FrictMat\_FrictMat\_FrictPhys()],

[Law2\_L3Geom\_FrictPhys\_ElPerfPl()]

),

NewtonIntegrator(gravity=(0,0,-9.81),damping=0.9),

HarmonicMotionEngine(A=(0.3\*Rc,0.3\*Rc,0.3\*Rc),f=(100,100,100),ids=body, label='shaker'),

TranslationEngine(translationAxis=(0,0,1),velocity=2.5\*Lp,ids=bottom,dead=True, label='bottomComp'),

TranslationEngine(translationAxis=(0,0,-1),velocity=2.5\*Lp,ids=top,dead=True, label='topComp'),

PyRunner(command='changeShake1()',virtPeriod=0.02),

PyRunner(command='changeShake2()',virtPeriod=0.14),

PyRunner(command='changeShake3()',virtPeriod=0.04),

PyRunner(command='changeShake3()',virtPeriod=0.16),

PyRunner(command='startCompression()',virtPeriod=0.04),

PyRunner(command='stopCompression()',virtPeriod=0.04+0.099),

PyRunner(command='write()',virtPeriod=0.15),

]

O.dt=.5\*PWaveTimeStep()

def changeShake1():

shaker.A = (0.3\*Rc,0.3\*Rc,-0.3\*Rc)

def changeShake2():

shaker.A = (0,0,0.1\*Rc)

shaker.f=(0,0,100)

shaker.fixed = False

def changeShake3():

shaker.A=(0,0,0)

shaker.f=(0,0,0)

shaker.fixed = True

def startCompression():

topComp.dead = False

bottomComp.dead = False

def stopCompression():

topComp.dead = True

bottomComp.dead = True

topComp.velocity = 0

topComp.fixed = True

bottomComp.velocity = 0

bottomComp.fixed = True

def write():

export.text('randomPacking'+str(2000\*Rsreal)+'mm')

O.pause()

## Building geometry for structured / unstructured mesh

## Building geometry for unstructured mesh

Building one porous unit

import sys

import salome

import numpy

salome.salome\_init()

theStudy = salome.myStudy

#import salome\_notebook

#notebook = salome\_notebook.NoteBook(theStudy)

sys.path.insert( 0, r'/home/adriaan')

###

### GEOM component

###

import GEOM

from salome.geom import geomBuilder

import math

import SALOMEDS

#parameters

Rc=51.4/2

Rsreal=1.5

scaleFactor = 1

Rs=scaleFactor\*Rsreal

#Transition from structured mesh to position of sphere (not too abrupt)

Transition = 25

gap = 0.025\*Rs

tolerance = 1e-7

#Loading centres

data = numpy.loadtxt('myYade/centresRandom3mm')

maxi = 0

for index in range (0,len(data)) :

if data[index][0] > maxi :

maxi = data[index][0]

#Lp=30.4+2\*gap

Lp = maxi\*1000 + Rs + gap

Li=0

Lo=0

geompy = geomBuilder.New(salome.myStudy)

O = geompy.MakeVertex(0, 0, 0)

OX = geompy.MakeVectorDXDYDZ(1, 0, 0)

OY = geompy.MakeVectorDXDYDZ(0, 1, 0)

OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

Disk\_1 = geompy.MakeDiskR(Rc, 2)

part = geompy.MakePrismVecH(Disk\_1, OX, Lp)

geompy.addToStudy( O, 'O' )

geompy.addToStudy( OX, 'OX' )

geompy.addToStudy( OY, 'OY' )

geompy.addToStudy( OZ, 'OZ' )

geompy.addToStudy( part, 'part' )

# adding all spheres in an array

# centres can be used for extra calculations like finding faces closest to sphere centre (Sphere surface)

centres = []

spheres = []

fluid = part

#Enable to import BREP and continu a (crashed) operation

#fluid = geompy.ImportBREP('/home/adriaan/Export/fluid\_CombinedScale\_ProcessShape'+str(2000\*Rsreal)+'mm.brep' )

notadded = []

volume\_spheres = 0

volume\_spheres\_wanted = 0

#Set 1 to have only gaps

onlyGaps = 0

#Add loop to add all spheres

for index in range (0,len(data)) :

x = 1000\*data[index][0]

y = 1000\*data[index][1]

z = 1000\*data[index][2]

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

p = geompy.MakeVertex(x,y,z)

if geompy.MinDistance(fluid, p) != 0 :

print('sphere already cutted ' +str(index))

sphere = geompy.MakeSpherePntR(p, Rs+2\*gap)

spheres.append(sphere)

else :

#Filter/Scale intersecting spheres

radius = Rs

#Sphere too close at border or not?

centreToCentre = math.sqrt(abs(y)\*\*2+abs(z)\*\*2)

atBorder=0

if centreToCentre > Rc-Rs-gap and centreToCentre < Rc+Rs+gap :

y=y\*(Rc+Rs+gap)/centreToCentre

z=z\*(Rc+Rs+gap)/centreToCentre

atBorder=1

iterationRoom = spheres + notadded

for sphereCompare in iterationRoom :

centre\_sphereCompare = geompy.MakeCDG(sphereCompare)

[x2,y2,z2] = geompy.PointCoordinates(centre\_sphereCompare)

dist = math.sqrt((x-x2)\*\*2+(y-y2)\*\*2+(z-z2)\*\*2)

volume\_sphereCompare = geompy.BasicProperties(sphereCompare)[2]

power = 0.33333333333333333333333

radius\_sphereCompare = (0.75\*volume\_sphereCompare/math.pi)\*\*power

#Sphere overlapping not enough or too close?

if dist < Rs+gap+radius\_sphereCompare and dist > Rs-gap+radius\_sphereCompare :

#scale sphere

if onlyGaps == 0 and volume\_spheres < volume\_spheres\_wanted or atBorder == 1 :

#make overlapping

print('overlap')

newradius = dist+gap-radius\_sphereCompare

if radius < newradius :

radius = newradius

else :

#make gap

print('gap')

newradius = dist-gap-radius\_sphereCompare

if radius > newradius :

radius = newradius

try :

sphere = geompy.MakeSpherePntR(p, radius)

except :

print('Failed making sphere')

pass

try:

fluid = geompy.MakeCut(fluid, sphere)

except:

print('step ' + str(index) + ' added to notadded')

notadded.append(sphere)

try:

fluid = geompy.ProcessShape(fluid, ["FixShape"], ["FixShape.Tolerance3d", "FixShape.MaxTolerance3d"], ["1e-07", "1"])

except:

print('Processing shape failed')

else:

print('Processing shape succeeded')

pass

else:

#run block if try succeeded (no exception occured)

centres.append([x,y,z])

spheres.append( sphere )

volume\_spheres = (Lp+Li+Lo)\*math.pi\*Rc\*\*2-geompy.BasicProperties(fluid)[2]

volume\_spheres\_wanted = len(spheres)\*1.333333333333333333\*math.pi\*Rsreal\*\*3

print('step ' + str(index))

if index%100 == 0 or index == len(data)-1 and index != 0:

geompy.ExportBREP(fluid, '/home/adriaan/Export/fluid\_CombinedScale\_ProcessShape2'+str(2000\*Rsreal)+'mm.brep' )

print('ExportBREP')

#Try again to add failed spheres

loops = 0

maxloop = 1.1\*len(notadded)

while len(spheres) < len(data) and loops < maxloop :

tries = 0

loops = loops + 1

try:

fluid = geompy.ProcessShape(fluid, ["FixShape"], ["FixShape.Tolerance3d", "FixShape.MaxTolerance3d"], ["1e-07", "1"])

except:

print('Processing shape failed')

else:

print('Processing shape succeeded')

for sphere in notadded: #[::-1] :

try:

#Try again with processed shape (corrected intersected edge and disable detection of self-intersection)

fluid = geompy.MakeCut(fluid, sphere)

except:

tries = tries + 1

print('Retry Failed')

#Between ProcessShape and success increases number of fails...

#So disable this limit (increases drastically time if it would always fail)

#if tries > 20 :

# break

pass

else :

spheres.append( sphere )

print('Retry succeeded ' + str(len(notadded)))

notadded.remove(sphere)

#It seems after each success shape needs to be processed again, so immediate break

break

#Try to partition leftovers and cut

try:

leftover = MakePartition(notadded)

fluid = MakeCut(fluid, leftover)

except:

print('Failed Partition rest')

pass

else:

print('Partition cut succeeded')

#Calculation of real porosity

amountOfSpheres = geompy.MakeVertex(0,0,0)

geompy.addToStudy(amountOfSpheres, 'amountOfSpheres='+str(len(spheres)))

volume\_packedzone = Lp\*math.pi\*Rc\*\*2

volumes = len(data)\*1.3333333333333333333\*math.pi\*Rsreal\*\*3

epsWanted = 1-volumes/volume\_packedzone

epsilonWanted = geompy.MakeVertex(0,0,0)

geompy.addToStudy(epsilonWanted, 'epsilonWanted='+str(epsWanted))

epsilonAchieved = geompy.MakeVertex(0,0,0)

volume\_inletoutletzone = (Li+Lo)\*math.pi\*Rc\*\*2

volume\_fluidAroundPacking = geompy.BasicProperties(fluid)[2]-volume\_inletoutletzone

epsAchieved = volume\_fluidAroundPacking/volume\_packedzone

geompy.addToStudy(epsilonAchieved, 'epsilonAchieved='+str(epsAchieved))

geompy.addToStudy(fluid, 'fluidAroundPacking')

# Set HDF file path

path='/home/adriaan/Export/Unstructured2'+str(Rsreal\*2000)+'mm.hdf'

# Save the study

salome.myStudyManager.SaveAs(path,salome.myStudy,False)

import os

from killSalomeWithPort import killMyPort

killMyPort(os.getenv('NSPORT'))

Fusing periodic porous zones with inlet- and outletzone

import sys

import salome

salome.salome\_init()

theStudy = salome.myStudy

import salome\_notebook

notebook = salome\_notebook.NoteBook(theStudy)

sys.path.insert( 0, r'/home/adriaan/Export')

###

### GEOM component

###

import GEOM

from salome.geom import geomBuilder

import math

import SALOMEDS

geompy = geomBuilder.New(theStudy)

Li=200

part = geompy.ImportBREP("/home/adriaan/Export/Cut\_1\_Shell.brep" )

[Xmin,Xmax, Ymin,Ymax, Zmin,Zmax] = geompy.BoundingBox(part, True)

Lp=Xmax-Xmin

Lo=200

Rsreal=1.5

Rc = 25.7

#Transition = 25

Transition = 0

O = geompy.MakeVertex(0, 0, 0)

OX = geompy.MakeVectorDXDYDZ(1, 0, 0)

OY = geompy.MakeVectorDXDYDZ(0, 1, 0)

OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

Disk\_1 = geompy.MakeDiskR(Rc, 2)

inletzone = geompy.MakePrismVecH(Disk\_1, OX, Li-Transition, theName='inletzone')

[Xmin,Xmax, Ymin,Ymax, Zmin,Zmax] = geompy.BoundingBox(inletzone, True)

part1 = geompy.MakeTranslation(part, Xmax, 0, 0)

outletzone = geompy.MakePrismVecH(Disk\_1, OX, Lo-Transition)

#Transit = geompy.MakePrismVecH(Disk\_1, OX, Transition)

#Transit1 = geompy.MakeTranslation(Transit, Li-Transition, 0, 0, theName='Transition1')

#Transit2 = geompy.MakeTranslation(Transit, Li+Lp, 0, 0, theName='Transition2')

#Fuse\_1 = geompy.MakeFuseList([inletzone, Transit1], False, False)

#print('fuse inlet')

Fuse\_1 = geompy.MakeFuseList([inletzone, part1], False, False)

print('fuse first part')

for i in range (1,10) :

[Xmin,Xmax, Ymin,Ymax, Zmin,Zmax] = geompy.BoundingBox(Fuse\_1, True)

Translation = geompy.MakeTranslation(part, Xmax, 0, 0)

#parts.append(Translation)

print('Translation ' + str(i))

Fuse\_1 = geompy.MakeFuseList([Fuse\_1, Translation], False, False)

print('Fuse ' + str(i))

geompy.ExportBREP(Fuse\_1, '/home/adriaan/Export/fluid\_Whole'+str(2\*Rsreal)+'mm.brep' )

print('ExportBREP')

#Fuse\_1 = geompy.MakeFuseList([Fuse\_1, Transit2], False, False)

#print('fuse transit')

[Xmin,Xmax, Ymin,Ymax, Zmin,Zmax] = geompy.BoundingBox(Fuse\_1, True)

outletzone = geompy.MakeTranslation(outletzone, Xmax, 0, 0, theName='outletzone')

Fuse\_1 = geompy.MakeFuseList([Fuse\_1, outletzone], False, False)

print('fuse outlet')

geompy.addToStudy( O, 'O' )

geompy.addToStudy( OX, 'OX' )

geompy.addToStudy( OY, 'OY' )

geompy.addToStudy( OZ, 'OZ' )

geompy.addToStudy( part, 'part' )

geompy.addToStudy( Fuse\_1, 'fluidAroundPacking' )

geompy.addToStudy( O, 'amountOfSpheres=2877' )

geompy.addToStudy( O, 'epsilonWanted=0.355854204121' )

volume\_packedzone = Lp\*math.pi\*Rc\*\*2

volume\_fluidAroundPacking = geompy.BasicProperties(part)[2]

epsAchieved = volume\_fluidAroundPacking/volume\_packedzone

geompy.addToStudy(O, 'epsilonAchieved='+str(epsAchieved))

geompy.ExportBREP(Fuse\_1, '/home/adriaan/Export/fluid\_test'+str(2\*Rsreal)+'mm.brep' )

print('ExportBREP')

import os

from killSalomeWithPort import killMyPort

killMyPort(os.getenv('NSPORT'))

if salome.sg.hasDesktop():

salome.sg.updateObjBrowser(1)

Define boundary faces and mesh

import sys

import salome

salome.salome\_init()

theStudy = salome.myStudy

import salome\_notebook

notebook = salome\_notebook.NoteBook(theStudy)

sys.path.insert( 0, r'/home/adriaan/Export')

###

### GEOM component

###

import GEOM

from salome.geom import geomBuilder

import math

import SALOMEDS

#parameters

Li=200

Lpi=29.75

#amount of periodic porous units (parts)

aparts = 10

Lp=aparts\*29.75

Lo=200

Rsreal=1.5

Rc = 25.7

amount = 28770

Transition = 25

geompy = geomBuilder.New(theStudy)

packedTube = geompy.ImportBREP("/home/adriaan/Export/fluid\_test3.0mm.brep", theName='packedTube' )

#Calculation of real porosity

amountOfSpheres = geompy.MakeVertex(0,0,0)

geompy.addToStudy(amountOfSpheres, 'amountOfSpheres='+str(amount))

volume\_packedzone = Lp\*math.pi\*Rc\*\*2

volumes = amount\*1.3333333333333333333\*math.pi\*Rsreal\*\*3

epsWanted = 1-volumes/volume\_packedzone

epsilonWanted = geompy.MakeVertex(0,0,0)

#geompy.addToStudy(epsilonWanted, 'epsilonWanted='+str(epsWanted))

epsilonAchieved = geompy.MakeVertex(0,0,0)

volume\_inletoutletzone = (Li+Lo)\*math.pi\*Rc\*\*2

volume\_fluidAroundPacking = geompy.BasicProperties(packedTube)[2]-volume\_inletoutletzone

epsAchieved = volume\_fluidAroundPacking/volume\_packedzone

geompy.addToStudy(epsilonAchieved, 'epsilonAchieved='+str(epsAchieved))

# CREATE GROUPS FACES

#INLET/OUTLET

inletFaces = []

outletFaces = []

#If you loop on interestion: error: Multiple faces near the given point are found

#inner loop starts at 30 degrees and turns 60 degrees

left = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex( 0, 0, 0))

inletFaces.append(left)

right = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex( Li+Lp+Lo, 0, 0 ))

outletFaces.append(right)

#CREATE GROUPS VOLUMES

#inletShapes = []

#outletShapes = []

#inner loop starts at 30 degrees and turns 60 degrees

#left = geompy.GetBlockNearPoint( packedTube, geompy.MakeVertex( 0, 0, 0 ))

#inletShapes.append(left)

#right = geompy.GetBlockNearPoint( packedTube, geompy.MakeVertex( Li+Lp+Lo, 0, 0 ))

#outletShapes.append(right)

#WALLS

walls = []

internals = []

left = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex( Li-Transition, 0, 0))

internals.append(left)

right = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex( Li+Lp+Transition, 0, 0 ))

internals.append(right)

wall = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li-Transition+0.01\*Rsreal, Rc, Rc ))

walls.append(wall)

wall = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li+Lp+Transition-0.01\*Rsreal, Rc, Rc ))

walls.append(wall)

viscous = []

for i in range (0,aparts) :

wall = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li+i\*Lpi+0.01\*Rsreal, Rc, Rc ))

walls.append(wall)

internal = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li+i\*Lpi, Rc, Rc ))

internals.append(internal)

wall = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li/2, Rc, Rc ))

walls.append(wall)

viscous.append(wall)

wall = geompy.GetFaceNearPoint( packedTube, geompy.MakeVertex(Li+Lp+Lo/2, Rc, Rc ))

walls.append(wall)

viscous.append(wall)

#Add groups to Study

#volumes

#inletzone = geompy.CreateGroup( packedTube, geompy.ShapeType["SOLID"], "inletzone" )

#geompy.UnionList( inletzone, inletShapes)

#outletzone = geompy.CreateGroup( packedTube, geompy.ShapeType["SOLID"], "outletzone" )

#geompy.UnionList( outletzone, outletShapes)

#faces

all = geompy.SubShapeAll( packedTube, geompy.ShapeType["FACE"])

inlet = geompy.CreateGroup( packedTube, geompy.ShapeType["FACE"], "inlet" )

geompy.UnionList( inlet, inletFaces)

outlet = geompy.CreateGroup( packedTube, geompy.ShapeType["FACE"], "outlet" )

geompy.UnionList( outlet, outletFaces)

wall = geompy.CreateGroup( packedTube, geompy.ShapeType["FACE"], "fixedWall" )

geompy.UnionList( wall, walls)

viscousLayers = geompy.CreateGroup( packedTube, geompy.ShapeType["FACE"], "viscousLayers" )

geompy.UnionList( viscousLayers, viscous)

sphereSurfaces = geompy.CreateGroup( packedTube, geompy.ShapeType["FACE"], "sphereSurfaces")

geompy.UnionList( sphereSurfaces, all )

geompy.DifferenceList( sphereSurfaces, inletFaces + outletFaces + walls + internals)

print ("saving hdf")

try:

# Save the study

salome.myStudyManager.SaveAs(r'/home/adriaan/Export/Mesh\_PackedBed\_Total\_'+str(2\*Rsreal)+'mm.hdf',salome.myStudy,False)

except:

print 'Saving .hdf failed. Invalid file name?'

###

### SMESH component

###

print("starting mesh")

import SMESH, SALOMEDS

from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)

Mesh\_1 = smesh.Mesh(packedTube)

NETGEN\_2D3D = Mesh\_1.Tetrahedron(algo=smeshBuilder.NETGEN\_1D2D3D)

NETGEN\_3D\_Parameters = NETGEN\_2D3D.Parameters()

#NETGEN\_3D\_Parameters.SetMaxSize( 0.5 )

#NETGEN\_3D\_Parameters.SetMinSize( 0.000674977 )

NETGEN\_3D\_Parameters.SetMaxSize( 5 )

NETGEN\_3D\_Parameters.SetMinSize( 0.00674977 )

NETGEN\_3D\_Parameters.SetSecondOrder( 0 )

NETGEN\_3D\_Parameters.SetOptimize( 1 )

NETGEN\_3D\_Parameters.SetFineness( 2 )

NETGEN\_3D\_Parameters.SetUseSurfaceCurvature( 1 )

NETGEN\_3D\_Parameters.SetFuseEdges( 1 )

NETGEN\_3D\_Parameters.SetQuadAllowed( 0 )

isDone = Mesh\_1.Compute()

inlet\_1 = Mesh\_1.GroupOnGeom(inlet,'inlet',SMESH.FACE)

inlet\_2 = Mesh\_1.GroupOnGeom(inlet,'inlet',SMESH.NODE)

outlet\_1 = Mesh\_1.GroupOnGeom(outlet,'outlet',SMESH.FACE)

outlet\_2 = Mesh\_1.GroupOnGeom(outlet,'outlet',SMESH.NODE)

sphereSurfaces\_1 = Mesh\_1.GroupOnGeom(sphereSurfaces,'sphereSurfaces',SMESH.FACE)

sphereSurfaces\_2 = Mesh\_1.GroupOnGeom(sphereSurfaces,'sphereSurfaces',SMESH.NODE)

fixedWall\_1 = Mesh\_1.GroupOnGeom(wall,'fixedWall',SMESH.FACE)

fixedWall\_2 = Mesh\_1.GroupOnGeom(wall,'fixedWall',SMESH.NODE)

## Set names of Mesh objects

smesh.SetName(NETGEN\_2D3D.GetAlgorithm(), 'NETGEN\_2D3D')

smesh.SetName(NETGEN\_3D\_Parameters, 'NETGEN 3D Parameters')

smesh.SetName(Mesh\_1.GetMesh(), 'Mesh\_1')

smesh.SetName(outlet\_2, 'outlet')

smesh.SetName(inlet\_2, 'inlet')

smesh.SetName(fixedWall\_2, 'fixedWall')

smesh.SetName(sphereSurfaces\_2, 'sphereSurfaces')

smesh.SetName(outlet\_1, 'outlet')

smesh.SetName(inlet\_1, 'inlet')

smesh.SetName(fixedWall\_1, 'fixedWall')

smesh.SetName(sphereSurfaces\_1, 'sphereSurfaces')

print("meshing completed")

try:

Mesh\_1.ExportUNV( r'/home/adriaan/Export/Mesh\_PackedBed\_Total\_'+str(2\*Rsreal)+'mm.unv' )

except:

print 'ExportUNV() failed. Invalid file name?'

print ("saving hdf")

try:

# Save the study

salome.myStudyManager.SaveAs(r'/home/adriaan/Export/Mesh\_PackedBed\_Total\_'+str(2\*Rsreal)+'mm.hdf',salome.myStudy,False)

except:

print 'Saving .hdf failed. Invalid file name?'

import os

from killSalomeWithPort import killMyPort

killMyPort(os.getenv('NSPORT'))

1. Permeability depends on orientation and flow regime. A tensor gives difference by orientation. is valid within Darcy’s regime and within non-Darcy or Forchheimer regime. [↑](#footnote-ref-1)
2. Forchheimer is important when inertia effects play a role (increased Re) and corrects Darcy’s low. [↑](#footnote-ref-2)
3. Reynolds number (Re) is the ratio of inertial and viscous forces. [↑](#footnote-ref-3)
4. Velocity is the superficial velocity or volumetric flow divided by cross section. Within a porous media this velocity is not the absolute velocity of a particle. [↑](#footnote-ref-4)
5. Stress , fluid parameter is not be confused with tortuosity , geometrical parameter [↑](#footnote-ref-5)
6. Not to confuse with Darcy friction coefficient, fD, and Fanning friction coefficient, f. [↑](#footnote-ref-6)
7. Tortuosity is the ratio between the actual path following a flow particle and the ideal (straight) pathway. [↑](#footnote-ref-7)
8. When non-orthogonality is higher than 80 (checkMesh), facelimiters are added to gradSchemes and «blending factor» 0,333. More orthogonal meshes use celllimiters and blending factor up to 1 (orthogonal correction) or don’t need correction. Within fvSolution non-orthogonalCorrectors can be changed as well, from 1 to 3 for respectively orthogonal and non-orthogonal mesh. (Guerrero, 2015) [↑](#footnote-ref-8)
9. SetSet is a command-line utility. Following commands make a subset: 1) “*cellSet <name> new cellToCell zeroVolumeCells*” any 2) “*cellSet <name>” invert*” 3) “*cellSet <name> subset”*. Now subsetMesh can be run. (OpenFOAM wiki) [↑](#footnote-ref-9)
10. SubsetMesh puts a subset in an empty patch. Example: “*subsetMesh <name> -overwrite*”. (OpenFOAM wiki) It can be necessary to remove this extra patch manually. [↑](#footnote-ref-10)