Clausthal University of Technology
Institute of Applied Mechanics

Automatic Calculation and Evaluation of Flow in Complex Geometries Using Finite Volume and Lattice Boltzmann Methods

Doctoral Thesis
(Dissertation)

to be awarded the degree
Doctor of Engineering (Dr.-Ing.)

submitted by
Alexander Bufe
from Braunschweig

approved by the
Faculty of Mathematics/Computer Science and
Mechanical Engineering,
Clausthal University of Technology,

Date of oral examination:
28-04-2022

Abstract

Despite significant progress, computational fluid dynamics (CFD) can still not be used as a “black box approach” as meshing often requires manual intervention and the choosing of numerical parameters deep knowledge of the methods behind CFD. Improving CFD towards such a black box solution not only reduces the barrier of application as less specialized knowledge is required, but also allows for scientific insight. For example, much more data can be generated that is needed to develop accurate models for some problems. This thesis illustrates these benefits with three exemplary applications:

- The accurate prediction of the pressure drop of a sphere packed bed is of great importance in engineering. For geometries where the spheres are relatively large compared to the confinement, the wall effect plays another important role. Many correlations have been presented, usually based on experimental measurements that differ in a range of approx. 20 %. Here, the combination of simulated packing generation and CFD is used to evaluate the pressure drop for a very large number of packings with different sphere diameters and different geometries of the confining walls. It is shown that for small ratios of sphere diameter to hydraulic diameter of the reactor the pressure drop is a non-monotonic function which can explain the differences in experimental findings.

- The Fischer-Tropsch synthesis is again of increasing interest as it allows the production of carbon-neutral fuel. Transport pores can be added to the catalyst needed for the reaction to enhance transport and consequently the yield. A three-dimensional extension of a one-dimensional model from literature for transport and reaction is presented here. The automation of the calculation is used to enable the algorithmic optimization of the catalyst layers. The results show that for transport pores larger than 50 µm the problem must be treated as three-dimensional. Larger transport pores up to a diameter of 250 µm can also be used to achieve a gain in area-time yield, but thicker catalyst layers and a higher transport pore porosity are needed to overcome the drawbacks of larger pores.

- Nasal septum deviation is very common in general population but it is unclear why it causes symptoms for certain patients while others report no discomfort. Previous studies focused on the analysis of few selected cases which did not lead to clear results as the human nose shows high natural variations in geometry. Here, a fully automatic approach for calculating critical parameters like the pressure drop and the flow distribution between the two airways from computed tomography (CT) scans is presented. Furthermore, a method to reduce the computational time by removing paranasal sinuses from the scan incorporating machine learning algorithms is proposed. For this case, fully automatic processing can be used to convert a whole database of CT scans to fluid dynamic parameters that can be used for statistical analysis. Furthermore, it could allow the introduction of CFD analysis to clinical practice.

The lattice Boltzmann method (LBM) is an alternative method to "classical" finite-volume based solvers of the Navier-Stokes equations. Since it offers easy generation of grids, a novel LBM implementation is used here to calculate the flow through the sphere packings and the nasal cavity. The implementation features good portability to various systems and hardware like GPUs which due to their cost-effectiveness broaden the applicability of CFD. It can utilize grid refinement and a meshing algorithm suitable for GPUs is presented. To overcome slow IO and to simplify automatic evaluation, GPU assisted co-processing is implemented. Nevertheless, the application case of Fischer-Tropsch synthesis shows that "classical", finite volume based solvers like OpenFOAM are also valid choice for automatic processing if structured meshes can be used. Furthermore, for some applications, it is easier to model the problem using partial differential equations which can be directly solved using FVM.

Contents

List of Figures

List of Tables

List of Symbols

a	Channel Side Length, Reaction Rate Constant
A	Pressure Drop Parameter
A'	Pressure Drop of Infinitely Extended Bed
A_w	Wall Correction Term for Pressure Drop
b	Reaction Rate Constant
c_i	Concentration
c_L	Conversion Factor for Length
c_s	Speed of Sound
D	Diffusion Coefficient
D, D_{hyd}	Hydraulic Diameter
d_h	Hydraulic Diameter of Particles
d	Sphere Diameter
d_S	Sauter Diameter
D_{tube}	Tube Diameter
d_V	Volume Equivalent Diameter
$\vec{e}_i$	ith Discrete Velocity
f	Boltzmann Distribution Function, Friction Factor
f_i	Boltzmann Distribution Function (ith Component)
$\tilde{f}$	Post-Collision Distribution Function
g^A	Binary Function for Scan A
H	Plate Distance, Catalyst Layer Height
k	Constant
k_α	Coefficient for Chain Growth Probability
K	Dimensionless Drag
p	(Partial) Pressure
q	Dimensionless Wall Distance
r	Reaction Rate
Re	Reynolds Number
t	Time
$\vec{u}$	Macroscopic Velocity
v	Speed of Sound (Macroscopic)
V	Volume

$V_{\vec{\xi}}$	Momentum Space Volume
$V_{\vec{x}}$	Space Volume
$\partial V_{\vec{x}}$	Balance Volume Surface (Space Part)
ΔV	Balance Volume
$\vec{x}$	Position Vector
α	Chain Growth Probability
β	Coefficient for Chain Growth Probability
ϵ	Porosity
δ	Marker Function
δ_t	Time Step Length
δ_x	Grid Cell Length
Λ	Dimensionless Pressure Drop
ν	Viscosity, Stoichiometric Coefficient
ν_L	Molar Volume
Ω	Collision Operator
Ω_i	Collision Operator (ith component)
ω_i	Weight Factor for Equilibrium Function
Ψ	Sphericity
ρ	Density
σ	Standard Deviation
τ	Relaxation Parameter, Tortuosity
$\vec{\xi}$	Microscopic Velocity
$\nabla_{\vec{x}}$	Nabla Operator in Space
$sim(A, B)$	Similarity Function between Scans A and B

List of Abbreviations

AABB	Axis-Aligned Bounding Box
BFS	Breadth-First Search
BVH	Bounding Volume Hierarchy
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CT	Computed Tomography
DFS	Depth-First Search
FTS	Fischer-Tropsch Synthesis
FVM	Finite Volume Method
LBM	Lattice Boltzmann Method
NASA	National Aeronautics and Space Administration
NOSE	Nasal Obstruction Symptom Evaluation
NSD	Nasal Septum Deviation
RCP	Random Close Packing
XOR	Exclusive Or

1 Introduction

Computational fluid dynamics (CFD) has evolved from a topic of mostly academic interest to a valuable tool in daily engineering practice thanks to the increased computational power and improved algorithms. The usability has also greatly improved by better meshing algorithms and integration into CAD software. However, CFD usually still needs a lot of manual work for the correct creation of a high quality mesh. The choosing of numerical settings and models still requires specialized CFD knowledge as well. NASA has recognized these problems and lists "a much higher degree of automation in all steps of the analysis process [...]" as one of their visions for CFD in 2030, also stating that "[...] full automation is essential in order for a conceptual designer to effectively exploit the capacity of high performance computers and physics-based simulation tools." (Slotnick et al., 2014). The NASA's main focus is aerosciences but this thesis underlines that other disciplines can profit as well from such an automated, black box solution with three exemplary applications:

- The pressure drop of particle packings is of great importance in chemical engineering and therefore accurate models are needed. As it is dependent on many parameters like particle shape and the distance of confining walls as well as the random distribution of particles, many data points are needed to conceive accurate models. Automatic calculations can help to greatly increase the data, as it minimizes the amount of manual work needed to generate a data point.
- The Fischer-Tropsch synthesis is a process to convert carbon monoxide and hydrogen to hydrocarbons that can be used as fuel. A catalyst is needed for the reaction. Transport pores can be introduced to the catalyst layer to enhance diffusion. The arrangement of these transport pores has to be optimized for maximum yield of the desired products. With availability of a fully automatic evaluation method for such catalyst layers, an algorithmic optimization can be used to find optimal catalyst geometries.
- Nasal obstruction like a deviated septum can lead to a variety of complaints. The connections between symptoms and type of obstruction are unclear and due to the normal statistical variations of the human nose, a high amount of data is needed to develop statistical sound models. A black box approach can again simplify the generation of this data by reducing the amount of needed work. Furthermore, it would allow the usage of CFD based models for medical practitioners.

The lattice Boltzmann method (LBM) is a CFD method for fluid simulation which in contrast to “classical” finite-volume based solvers of the Navier-Stokes equations solves the Boltzmann equation with a finite-difference method. It offers advantages as easy mesh generation and good suitability for massively parallel architectures like GPUs. Therefore, the presented black box approaches for flow calculation are mainly based on a new GPU oriented implementation of LBM, *lbm4e*.

Note that the structure of this thesis follows a *logical* instead of a *chronological* order. First, a short introduction to the theory of LBM and some common models for boundary conditions and grid refinement is given in chapter 2. Then, the key features of *lbm4e* will be highlighted in chapter 3, followed by the application cases, particle packings in chapter 4 and nasal cavity flow in chapter 6 before concluding with a summary in chapter 7. The analysis of the pressure drop in sphere packings was actually one of the earliest works whereas some features of the implementation like the GPU based mesh generation were done later despite being presented earlier. Because of that, not all features described in the implementation section could be used for the later presented cases. Nevertheless, this allows for an easier to follow structure of the thesis.

2 Introduction to the Lattice Boltzmann Method

2.1 Introduction

The classical approach to solve fluid dynamics problems numerically is based on the Navier-Stokes equations using the finite volume method (FVM).[1] The Navier-Stokes equations can be derived from the conversation of mass and momentum. These equations are discretized by a number of volumes, for each of which the conservation is expressed as a sum of fluxes over the boundaries, which can be expressed with the volume's and the neighboring volume's values leading to a sparse system of linear equations that is then solved. Due to the non-linear nature of the Navier-Stokes equation, an iterative scheme has to be used for solving.

In contrast, the lattice Boltzmann method is not even based on the Navier-Stokes equations and can be seen as a method to solve the Boltzmann equation. It is derived based on considerations from the kinetic theory of gases. One might first assume that therefore LBM is only valid for gases, but from a fluid dynamic point of view there is no difference between flows of gases and liquids with the same Reynolds number as long as the flow is incompressible and there are no special phenomena like cavitation.

This chapter only aims at giving a short and understandable introduction to the general ideas and models of LBM used in this thesis. For a broader introduction, the reader is referred to textbooks like those by Hänel (2004)[2] or Guo and Shu (2013).

2.2 Main Idea of Kinetic Theory

Fluids consists of a multitude of molecules and a simple model is used to describe the movement of the molecules: Molecules are treated as spheres with each molecule having a microscopic velocity $\vec{\xi}$ and their movement follows classical mechanics and their momentum changes only by external forces or by collision between molecules. For fluid dynamic problems usually quantities like pressure, density and velocity of a fluid are of interest. These macroscopic quantities are a result of the microscopic motion of molecules:

[1]In the context of LBM this is usually referred to as "classical" or "traditional" CFD.
[2]only available in German

Density is defined as mass per volume. Thus, the density ρ for a small volume V can be determined by multiplying the number of molecules in it with their molecular mass. The observed macroscopic velocity $\vec{u}$ for a small volume is the average of the microscopic velocities. When molecules collide with the walls of a container they exchange momentum. A force is defined as momentum exchange per time and therefore the force resulting from pressure can also be calculated from microscopic movement.

A straightforward, naïve simulation approach would therefore be to create an initial random distribution for position and momentum of the molecules and then simulate their movement according to classical mechanics. However, this approach is not feasible for practically relevant cases as the number of molecules in a cubic meter under standard conditions is of the order of 10^{25} molecules. Furthermore, knowing the individual position of each molecule is unnecessary as only macroscopic quantities are of interest.

A better way to model the microscopic movement is to use a statistical approach and to introduce a probability density function f. This function is dependent on space and momentum (the phase space) as well as on time t. It can be used to calculate the amount of molecules for given areas of space and momentum by integrating over this area. Mathematical definition usually requires a probability density function to be normalized, meaning that the integral over all possible space and momentum states should be equal to one. However, in context of LBM usually an unnormalized function is used, which if integrated over all space and momentum states gives the total mass or – depending on definition – the total number of molecules in the system.

This distribution function f can be used to calculate the desired macroscopic quantities. The density is simply the total mass of all molecules of arbitrary momentum for a given point:

$$\rho(\vec{x},t) = \int f(\vec{x},\vec{\xi},t)\mathrm{d}\vec{\xi} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\vec{x},\vec{\xi},t)\mathrm{d}\xi_1\mathrm{d}\xi_2\mathrm{d}\xi_3 \tag{2.1}$$

As stated before, the macroscopic velocity $\vec{u}$ is the average of the microscopic velocities $\vec{\xi}$:

$$\vec{u}(\vec{x},t) = \frac{\int \vec{\xi} f(\vec{x},\vec{\xi},t)\mathrm{d}\vec{\xi}}{\int f(\vec{x},\vec{\xi},t)\mathrm{d}\vec{\xi}} = \frac{1}{\rho(\vec{x},t)}\int \vec{\xi} f(\vec{x},\vec{\xi},t)\mathrm{d}\vec{\xi} \tag{2.2}$$

Other quantities like internal energies can be calculated as higher order moments of the distribution function as well.

2.3 The Boltzmann Equation

A method to calculate the unknown distribution function f is needed for solving actual problems. The Navier-Stokes equations are derived from the conservation of mass and momentum. A similar derivation can be used to obtain an equation needed to calculate the distribution function, the Boltzmann equation.

Since the distribution function depends on the 6-dimensional phase space, a 6-dimensional volume for the balance has to be used. Such a volume can be interpreted as a collection of molecules of similar positions and similar microscopic velocities. Then all “flows” over the volume boundaries have to be calculated. The following changes are possible:

- The mass in the volume can vary by time (eq. 2.3). As the integration volume is not time-dependent, integral and derivative can be exchanged (eq. 2.4).

$$\Delta N_t = \frac{\partial}{\partial t} \int_{V_\xi} \int_{V_x} f \,\mathrm{d}\vec{x}\,\mathrm{d}\vec{\xi} \tag{2.3}$$

$$= \int_{V_\xi} \int_{V_x} \frac{\partial f}{\partial t} \,\mathrm{d}\vec{x}\,\mathrm{d}\vec{\xi} \tag{2.4}$$

- Molecules can move outside the space volume $V_{\vec{x}}$ due to the microscopic movement. To calculate the amount of molecules which are lost for the volume, the flux over the boundary, the product $\vec{\xi} f \cdot \vec{n}$, has to be integrated over the boundary $\partial V_{\vec{x}}$ (eq. 2.5) and over the volume in momentum space $V_{\vec{\xi}}$. This surface integral can be transformed to a volume integral using the divergence theorem (eq. 2.6). The nabla operator $\nabla_{\vec{x}} = (\partial/\partial x, \partial/\partial y, \partial/\partial z)^T$ is here used with an index $\vec{x}$ to denote derivation in spatial coordinates. The microscopic velocity $\vec{\xi}$ is independent of spatial coordinates and can therefore be put in front of the nabla operator $\nabla_{\vec{x}}$ which only contains derivatives in spatial coordinates (eq. 2.7). The term $\vec{\xi} \cdot \nabla_{\vec{x}}$ in the final equation 2.8 can be interpreted as a directional derivative in the direction of the momentum in space coordinates.

$$\Delta N_{\vec{x}} = \int_{V_{\vec{\xi}}} \int_{\partial V_{\vec{x}}} (\vec{\xi} f \cdot \vec{n}) \,\mathrm{d}S\,\mathrm{d}\vec{\xi} \tag{2.5}$$

$$= \int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} (\nabla_{\vec{x}} \cdot \vec{\xi} f) \,\mathrm{d}\vec{x}\,\mathrm{d}\vec{\xi} \tag{2.6}$$

$$= \int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} (\vec{\xi} \cdot \nabla_{\vec{x}} f) \,\mathrm{d}\vec{x}\,\mathrm{d}\vec{\xi} \tag{2.7}$$

$$= \int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} (\vec{\xi} \cdot \nabla_{\vec{x}}) f \,\mathrm{d}\vec{x}\,\mathrm{d}\vec{\xi} \tag{2.8}$$

- Molecules can accelerate or slow down by an external force field and leave the momentum volume. This leads to a similar term as for the space volume.

$$\Delta N_{\vec{\xi}} = \int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \vec{F} \cdot \nabla_{\vec{\xi}} f \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi} \tag{2.9}$$

- Finally, molecules can collide and - due to the momentum exchange - lose or gain momentum and therefore leave the momentum volume. A collision operator Ω is used to calculate those changes. Originally, the collision operator was derived from binary sphere collisions leading to an integral and making the Boltzmann equation a difficult to solve integro-differential equation. To simplify the equation, usually less complex models for the collision operator like the BGK-model (Bhatnagar et al., 1954) are used (eq. 2.11). The idea of the BGK-model is that according to the so called H-theorem the distribution function converges to the analytical Maxwell-Boltzmann distribution f^{eq} and therefore the collision can be modeled as a relaxation towards that Maxwell-Boltzmann distribution.

$$\Delta N_{\Omega} = \int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \Omega(f) \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi} \tag{2.10}$$

$$\Omega(f) = \omega(f^{\mathrm{eq}} - f) \tag{2.11}$$

Combining all the presented possible changes leads to the Boltzmann equation:

$$\underbrace{\int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \frac{\partial f}{\partial t} \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi}}_{\Delta N_t} + \underbrace{\int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \vec{\xi} \cdot \nabla_{\vec{x}} f \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi}}_{\Delta N_x} + \underbrace{\int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \vec{F} \cdot \nabla_{\vec{\xi}} f \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi}}_{\Delta N_\xi} = \underbrace{\int_{V_{\vec{\xi}}} \int_{V_{\vec{x}}} \Omega(f) \, \mathrm{d}\vec{x} \, \mathrm{d}\vec{\xi}}_{\Delta N_\Omega} \tag{2.12}$$

Since this equation holds true for an arbitrary volume, the integrals can be omitted:

$$\frac{\partial f}{\partial t} + \vec{\xi} \cdot \nabla_{\vec{x}} f + \vec{F} \cdot \nabla_{\vec{\xi}} f = \Omega(f) \tag{2.13}$$

2.4 From Boltzmann Equation to LBM

In order to actually use the Boltzmann equation to solve fluid dynamics problems, a discretization scheme is needed. Although other schemes like the finite volume method (Guo and Shu, 2013, 90 ff.) are possible, usually the finite difference method with a Cartesian grid is used. One can think of the values for a Cartesian grid as being at the center of a cell or at the cell points, the corner points of the cell. Here the first is used. The discrete velocities $\vec{e}_i$ are chosen so that a molecule moves from a cell to exactly a neighbored cell with distance δ_x within one time step δ_t. An additional discrete velocity is the zero vector. It is sufficient to limit the discrete velocities to neighbored cells as under standard conditions the average distance between two collisions is very small. Although there are 27 discrete velocities with this scheme in three-dimensional space, some models use less discrete velocities. LBM models are usually abbreviated with DnQm meaning a model with n dimensions and m discrete velocities. The discrete distribution functions for the i-th discrete velocity are notated with an index:

$$f_i(\vec{x}, t) = f(\vec{x}, \vec{e}_i, t) \tag{2.14}$$

The density and the macroscopic velocity can then be calculated by a sum instead of an integral:

$$\rho(\vec{x}_k, t_n) = \sum_{i=1}^{Q} f_i(\vec{x}_k, t_n) \tag{2.15}$$

$$\vec{u}(\vec{x}_k, t_n) = \frac{1}{\rho(\vec{x}_k, t_n)} \sum_{i=1}^{Q} \vec{e}_i f_i(\vec{x}_k, t_n) \tag{2.16}$$

The Boltzmann equation has to be discretized. For the sake of simplicity, external forces are neglected here. The microscopic velocity $\vec{\xi}$ in the Boltzmann equation is replaced by the discrete velocities e_i:

$$\frac{\partial f_i}{\partial t} + \vec{e}_i \cdot \nabla_{\vec{x}} f_i = \Omega(f) \tag{2.17}$$

The time derivative is discretized with a backwards difference:

$$\frac{\partial f_i}{\partial t} \approx \frac{f_i(\vec{x}_k, t_n) - f_i(\vec{x}_k, t_n - \delta t)}{\delta t} \tag{2.18}$$

The term $\vec{e}_i \cdot \nabla_{\vec{x}}$ can be interpreted as a directional derivative and therefore also be approximated with a finite difference:

$$\vec{e}_i \cdot \nabla_{\vec{x}} f_i \approx \frac{f_i(\vec{x}_k + \vec{e}_i \delta t, t_n) - f_i(\vec{x}_k, t_n)}{\delta t} \tag{2.19}$$

Combining both yields the final lattice Boltzmann scheme, where Ω_i is the discretized collision operator. The right side of the equation is usually called the collision step in which for each cell new post collision distribution values $\widetilde{f}_i$ are calculated from the old ones by applying the collision operator. In the streaming step, each of the post collision distribution functions is then moved one cell in its direction.

$$f_i(\vec{x}_k + \vec{e}_i \delta t, t_n) = \underbrace{\Omega_i(f(\vec{x}_k, t_n)) + f_i(\vec{x}_k, t_n - \delta t)}_{\widetilde{f}_i(\vec{x}_k, t_n)} \tag{2.20}$$

The simplest collision operator is the BGK collision operator (Bhatnagar et al., 1954). The distribution functions are relaxed with a viscosity dependent relaxation time τ to the Maxwell equilibrium function.

$$\Omega_i(f(\vec{x}_k, t_n)) = \frac{1}{\tau}\left(f^{\mathrm{eq}}(\vec{x}_k, t_n) - f(\vec{x}_k, t_n)\right) \tag{2.21}$$

The equilibrium function is obtained by developing the Maxwell distribution for low Mach numbers and discretizing it. The constants ω_i and c_s are dependent on the number m of distribution values.

$$f_i^{\mathrm{eq}}(\rho, \vec{u}) = \omega_i \rho \left(1 + \frac{\vec{e}_i \cdot \vec{u}}{c_s^2} + \frac{(\vec{e}_i \cdot \vec{u})^2}{2c_s^4} - \frac{\|u\|^2}{2c_s^2}\right) \tag{2.22}$$

To increase the stability of LBM, other collision operators like the multi relaxation time operator (d'Humières et al., 2002) or the cumulant collision operator (Geier et al., 2015) have been developed.

2.5 Boundary Conditions

At the boundaries of the calculation domain unknown distribution functions are introduced to the domain by the streaming step. Therefore, further models to determine those values according to boundary conditions specified in macroscopic quantities are necessary. In the context of LBM, it is helpful to differentiate between boundary conditions for curved boundaries like the surface of a geometry and those for the flat sides of the Cartesian grid. Here, only models used in this work are presented. An overview of other methods can be found in Guo and Shu (2013, 40 ff.).

2.5.1 Boundary Conditions for Flat Walls

To obtain a symmetry or slip boundary, the distribution functions from the nearest cell can simply be copied and mirrored. As then exactly the same distribution functions travel from both sides over the boundary, the velocity normal to the boundary will be zero. If the distribution functions are not mirrored but reversed, this leads to a no-slip boundary condition for flat walls.

For an inlet or outlet, the simplest approach is to set the unknown distribution functions to the values of the equilibrium function calculated with the prescribed macroscopic boundary conditions. If only density or macroscopic velocity are prescribed, the other quantity can be extrapolated from the interior. An improvement first presented by Guo et al. (2002) is to decompose the distribution functions into an equilibrium and a non-equilibrium part $f = f^{\text{eq}} + f^{\text{neq}}$. The equilibrium part can be directly computed as before, whereas the non-equilibrium part is copied from the interior. For a cell at the boundary at $\vec{x}_b$ and a neighbored cell inside the domain at $\vec{x}_f$, this can be written as:

$$f_i(\vec{x}_b, t) = f_i^{\text{eq}}(\rho_b, \vec{u}_b) + \underbrace{f_i(\vec{x}_f, t) - f_i^{\text{eq}}(\vec{x}_f, t)}_{f_i^{\text{neq}}(\vec{x}_f, t)} \tag{2.23}$$

2.5.2 Boundary Conditions for Curved Walls

For curved walls the problem arises that the wall often is not exactly between two cells. If the simple bounce-back rule is used under such conditions, the distribution function would then move more or less than one space unit. This problem can be solved by interpolating a position so that the movement is exactly one space unit. The position of the wall between the fluid cell next to the wall x_f and the first cell in the wall x_w will be denoted as q whereas $q = 1$ means the wall is at the wall cell and $q = 0$ that the wall is at the fluid cell. Both situations are illustrated in figure 2.1.

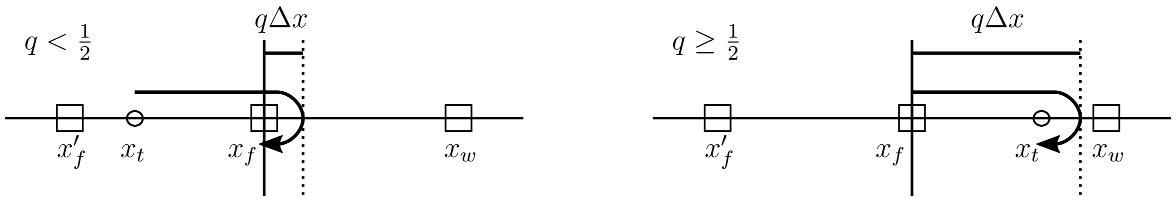


Figure 2.1: Illustration of the linear interpolated bounceback

For $q < \frac{1}{2}$ the distribution function in left direction arriving at the fluid cell x_f at time step $t + \delta_t$ is the function that bounced on the wall and started at the fictive point x_t. The distribution function can therefore be calculated by interpolating the distribution function in right direction at time step t for the point x_t.

For $q > \frac{1}{2}$ the distribution function in left direction for point $\vec{x}_t$ at time step $t + \delta_t$ is that in right direction that originated from $\vec{x}_f$ at time step t and therefore known. The unknown distribution function in left direction for $\vec{x}_f$ at time $t + \delta_t$ can thus again be calculated by interpolation between $\vec{x}'_f$ and $\vec{x}_t$.

When used with linear interpolation this scheme is known as the linear interpolated bounce-back and was first introduced by Bouzidi et al. (2001). The resulting formulas for direction $\vec{e}_i$ and the opposite direction $\vec{e}_{\bar{i}}$ are given in equation 2.24. Note that both cases reduce to the simple bounce-back scheme for $q = \frac{1}{2}$.

$$f_{\bar{i}}(\vec{x}_f, t + \delta_t) = \begin{cases} 2q f_i(\vec{x}_f, t) + (1 - 2q) f_i(\vec{x}'_f, t) & q < \frac{1}{2} \\ \frac{1}{2q} f_i(\vec{x}_f, t) + \frac{2q-1}{2q} f_{\bar{i}}(\vec{x}_f, t) & q > \frac{1}{2} \end{cases} \tag{2.24}$$

One problem with the simple bounceback as well as the interpolated bounceback rule is that the assumed wall position depends on the viscosity (Pan et al., 2006). This leads to the effect that two flows with the same Reynolds number can be calculated differently depending on the set viscosity. For most problems this error is negligibly small, but for some cases like porous media it can be of importance. For this case Ginzburg and d'Humières (2003) proposed the multireflection boundary condition which in combination with the two-relaxation time collision operator (Ginzburg, 2006) leads to viscosity independent results.

2.6 Grid Refinement

Often the same grid resolution is not needed everywhere and therefore a Cartesian grid can lead to a high number of unnecessary cells. Grid refinement can be used to overcome this issue. For the sake of simplicity only a refinement by a factor of two is considered here. In order to keep the speed for the distribution of information consistent on all refinement levels, the time step length on a finer level has to be half the one of a coarser level, meaning that for each coarse grid iteration two iterations on a finer grid level have to be performed. At the boundaries between two different levels of refinement, interpolation is needed to transfer the streamed distribution functions. This level transfer is usually done by the addition of "ghost cells", cells which are not actually part of the mesh but are added for easier implementation. In order to circumvent the need for temporal interpolation, two layers of ghost cells should be used around a finer level. At the beginning of a coarse step, both layers are interpolated and are used for streaming and collision in the fine level steps. After the first update on the fine level, the outer layer becomes invalid, but the inner layer still contains valid values that can be used to finalize the update of the actual grid cells. In order to better align the cells for interpolation, three layers of ghost cells should be used in practice. This is sketched in figure 2.2.

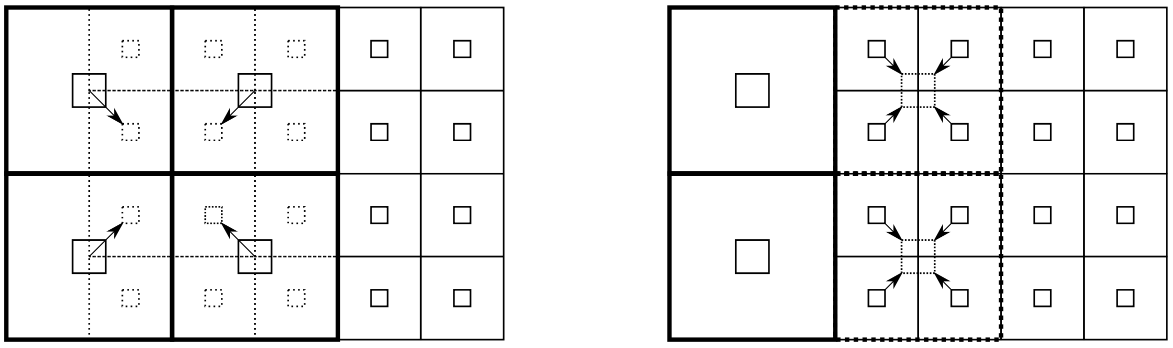


Figure 2.2: Illustration of transfer between levels by interpolation. Ghost cells are drawn with dashed lines. Three layers of ghost cells are drawn for the coarse-to-fine transfer. Note that in the coarse-to-fine step all four coarse cells are needed to interpolate each of the fine level cells.

The interpolation between the grid levels is done by transferring equilibrium and non-equilibrium parts separately. Macroscopic velocity and density are calculated and interpolated. These are then used to calculate the equilibrium part of the unknown distribution. The non-equilibrium part is interpolated directly. Note that in order to maintain continuity, the non-equilibrium part has to be rescaled according to $\omega_{\text{coarse}} f^{\text{neq}}_{\text{coarse}} = \omega_{\text{fine}} f^{\text{neq}}_{\text{fine}}$. The advantage of this separation is that the compact interpolation scheme (Geier et al., 2009) can be used to achieve second-order accuracy for the velocity interpolation with the same small stencil as used by linear interpolation by using high-order moments containing information on the derivatives of velocity.

2.7 Unit Conversion

Calculations are usually done in lattice Boltzmann units. However, specification of initial and boundary conditions, as well as evaluation are preferred in SI units. Therefore, conversions for the units for length, mass and time are required. A LBM unit can be written as the SI unit times a conversion factor c, e. g. for the length:

$$l_{\text{LBM}} = c_L \cdot l_{\text{SI}} \Longrightarrow c_L = \frac{l_{\text{LBM}}}{l_{\text{SI}}} \tag{2.25}$$

During grid generation the number of cells per meter has to be chosen, thus the coefficient c_L can be calculated.

The density for LBM calculations is usually initialized to a value of $\rho_{\text{LBM}} = 1 \frac{\text{lm}}{\text{lu}^3}$. The fluid density ρ_{SI} is known as well as the conversion factor c_L from the previous equation leading to the conversion factor for the mass.

$$\rho_{\text{LBM}} = \frac{c_M}{c_L^3} \cdot \rho_{\text{SI}} \Longrightarrow c_M = \frac{\rho_{\text{LBM}}}{\rho_{\text{SI}}} \cdot c_L^3 \tag{2.26}$$

The remaining conversion factor is the one for the time. The speed of sound v is a model dependent constant in LBM and therefore known as well as the speed of sound for the fluid in physical units. The conversion factor for time is chosen so that the Mach numbers in both frameworks are equal. For incompressible flows the Mach number and therefore speed of sound has no influence on the results and thus is only a scaling parameter. Choosing a lower speed of sound will increase the time step length but also the Mach number and therefore the speed of sound must be kept sufficiently high so that the flow is still incompressible.

$$v_{\text{LBM}} = \frac{c_L}{c_T} \cdot v_{\text{SI}} \Longrightarrow c_T = \frac{v_{\text{SI}}}{v_{\text{LBM}}} \cdot c_L \tag{2.27}$$

Note that there are also different possibilities for the unit conversion. As an example, it would also be possible to choose the time step length directly. However, the constraints for the Mach number still apply and therefore the conversion with the speed of sound was chosen as it is easier to estimate the Mach number in physical units.

Conversion of units other than mass, time and length are not needed in this work and are therefore omitted.

3 Implementation Highlights

3.1 General Goals

There are many free[1] LBM implementations like Palabos[2] or waLBerla[3] available. However, none of these could provide all the needed features like GPU support, grid refinement and specific LBM models like the cumulant collision operator. For this reason, a new LBM implementation, *lbm4e*, was created with the following design goals:

The code should deliver high performance: This is achieved by using a compiled language, C++, in contrast to the typically slower interpreted languages. Furthermore, this implementation should allow the usage of accelerator cards like GPUs. A theoretical performance analysis of LBM is done first (section 3.3.1) and benchmarking is used to find individual bottlenecks in the application.

The code should be portable: For high performance computing sometimes different hardware architectures than x86 like SPARC, POWER or ARM and different operating systems than GNU/Linux like IBM AIX are used. A portable build system is used to ease compilation on and for different systems. Furthermore, no direct calls to system functions should be used and instead if possible free platform agnostic libraries. The usage of free libraries also enhances portability as free libraries can be recompiled for different systems and are not dependent on vendor provided binaries.

The code should be maintainable: Code duplication should be avoided as it lengthens the code and requires fixing the same bug at multiple lines. Encapsulation should be used so that parts of the code or external libraries can easily be exchanged. Furthermore, "reinventing the wheel" should also be avoided and a library function used instead.

Some of these goals are somewhat mutually exclusive: Optimizations are sometimes hardware specific and thus can limit portability. Code duplication can sometimes increase performance. In such cases a balance has to be found between the conflicting goals.

[1]In this thesis, "free" is mostly used in the meaning of free software. For a introduction to free software see: www.gnu.org/philosophy/free-sw.en.html
[2]https://palabos.unige.ch/
[3]https://www.walberla.net/

3.2 Software Architecture

3.2.1 Build System

For building non-trivial software usually build automation is used to simplify the task of compiling the source code and packaging the resulting binaries. To compile the source code, one must invoke the compiler for each source file with the compiler dependent options and provide the system dependent paths to external libraries. Finally the generated object files have to be linked together. Build automation tools can greatly simplify these tasks as they allow setting of options in a compiler agnostic manner and automatically check for needed libraries and detect the necessary include paths.

CMake is used as the build tool here. It is available for many platforms and configured by configuration files written in a CMake specific scripting language. CMake does not directly compile the source code but instead creates the files necessary for the actual build system like Make. In order to simplify the build process a superbuild scheme is used that can also download and install necessary external libraries.

3.2.2 External Libraries

In order to not “reinvent the wheel”, a multitude of external free libraries is used:

- Libconfig allows parsing and writing of configuration files.
- Boost is used for platform independent access of the file-system. File-system support was also included in the C++ standard library since the 2017 revision. Since compiler support of the newer language standards is sometimes limited, boost is used instead. Both libraries are very similar, making future transition to the standard library very easy. Boost’s thread pool is also used for the implementation of the later presented co-processing.
- Hdf5 is a file format and a library for reading and writing files in this format. It is designed for saving large files in the scientific context. There is also a mpi version which allows parallel writing of single files. The library is used for writing checkpoints of the simulation.
- VTK is used for post-processing as well as reading a variety of file formats containing triangle meshes used as input for grid generation.
- OpenCL and HIP are used to program accelerator cards like GPUs.

3.2.3 Pimpl Idiom

One of the challenges when designing C++ code is the necessity of including the member variables in the header files, which has several drawbacks: It conflicts with the principle of encapsulation, it can enforce the inclusion of more header files, increasing the compilation time and if one of the member variables is changed, all files that include the header have to be recompiled, further increasing the compilation time. In order to overcome this issue, an opaque pointer to a struct containing the internal variables and providing internal methods (the implementation) can be used. This concept is generally known as the pointer-to-implementation or Pimpl idiom.

In the header file a nested struct is declared, but not defined, making it an incomplete class. The only member variable is then a pointer to this struct.

```
/* header file A.hpp */
class A{
public:
        A();

private:
        struct Impl;
        std::unique_ptr<Impl> impl_;
};
```

The struct is defined inside the implementation file of the class and therefore can only be accessed from within the file. It has to be created while construction of the class and destroyed upon its destruction. A smart pointer like the unique_ptr from the standard library can be used to simplify this.

```
/* implementation file A.cpp */
struct A::Impl{
        /* implementation */
}

A::A() : impl_(std::make_unique<Impl>()) {
}
```

Although this concept solves the mentioned challenges, it should be noted that it also has several drawbacks:

Templates can sometimes only meaningfully be implemented in the header file and then have no access to the internal implementation.

It complicates inheritance as inheriting classes cannot directly access the implementation. This can to some extent be eased by providing protected getter and setter methods for the

	Hardware	Mem. GB	Bandwidth GB/s	Performance TFLOPS SP	Performance TFLOPS DP	Arithm. Int. FLOP/byte SP	Arithm. Int. FLOP/byte DP
GPU	AMD Radeon VII	16	1024	13.8	3.46	13.48	3.38
GPU	AMD RX 6900 XT	16	512	18.69	1.168	38.46	2.28
GPU	NVIDIA RTX 3090	24	936	35.58	0.556	38.02	0.59
GPU	NVIDIA A100	40	1555	19.5	9.7	12.54	6.24
CPU	Intel Xeon 9242		262	7.06	3.53	26.95	13.47
CPU	Intel i9-11900		50	1.28	0.64	25.6	12.8

Table 3.1: Overview of amount of memory, bandwidth of memory, theoretical peak performance for double (DP) and single (SP) precision as well as the threshold arithmetic intensity for common hardware. No amount of memory is given for CPUs as it is mainly dependent on the fitted modules. The arithmetic intensity given is the threshold above which an algorithm is compute bound instead of memory bound, i. e. the ratio of peak performance to memory bandwidth.

internal variables.

3.3 GPU Focused Implementation

3.3.1 Performance Analysis

In order to analyze the achievable performance of a kernel, the roofline model (Williams et al., 2009) can be used. First the *arithmetic intensity*, the number of (floating point) operations, FLOP[4], needed to execute the kernel divided by the amount of bytes read and written, is calculated. This value is then multiplied by the memory bandwidth of the used hardware. If the result is lower than the peak compute performance of the hardware, the kernel is memory bound and compute bound otherwise. The number of FLOP can be roughly estimated by counting the number of arithmetic operations like additions and multiplications in the source code. As an example, the D3Q15 MRT collision operator uses approximately 160 FLOP per cell (d'Humières et al., 2002) and needs to read and write 15 distribution functions per cell, a transfer of 240 bytes when double precision is used. Therefore, this operator has an arithmetic intensity of $0.66\,\frac{\text{FLOP}}{\text{byte}}$. Other collision operators and models with a lower or higher number of distribution functions will have an arithmetic intensity of a similar order of magnitude.

Table 3.1 gives an overview of the specification data for some common hardware. As can be seen, the calculated arithmetic intensity is significantly lower than the threshold

[4]Not to be confused with floating point operations per second, FLOPS.

between memory and compute bound for most listed hardware, making the LBM kernel per se memory bound. Therefore, optimization should be focused on memory bandwidth. The only exception are GPUs intended for graphical purposes like the RX 6900 XT and RTX 3090 as they usually deliver poor double-precision performance. It can also be seen, that the device memory of GPUs - although limited in size - has much higher bandwidth than the RAM used on CPU systems. Therefore a speed-up is to be expected when using GPUs instead of CPUs. The table ignores that CPUs have multiple layers of cache with much higher bandwidth. However in LBM, each distribution function is usually read and written exactly once, thus LBM does not benefit from caching.

3.3.2 Interfaces

There are multiple platforms available for programming GPUs or - more general - accelerator cards with specific advantages and disadvantages:

- CUDA is Nvidia's proprietary framework. It offers many advanced features like unified host-device memory and the possibility to use most C++ features for kernel programming. A drawback is that it is only available for Nvidias's GPUs due to its proprietary nature.

- HIP is a framework by AMD and very similar in concept to CUDA. For most CUDA functions a direct HIP equivalent exists making ports very easy. Code written with HIP can also be compiled using CUDA as a header which maps to the analogous CUDA functions is included. Since currently AMD and Nvidia are the main vendors of GPUs, HIP allows to use all common GPUs.

- OpenCL is an open standard for executing across several platforms and is not only limited to GPUs. Multiple implementations can be installed and used simultaneously by using a driver (OpenCL ICD) that forwards the function calls to the selected implementation. Portability is achieved by providing a compiler that is used for run-time compilation similar to OpenGL. There are multiple implementations available by the Hardware vendors like Intel (GPUs and CPUs), Nvidia and AMD (GPUs only). Furthermore, a free implementation, Portable Computing Language (Pocl), exists which is mostly aimed at CPUs. Kernels are written in OpenCL C which is based on C99. C++ is specified in newer standards, but only supported by very few implementations. HIP and CUDA offer the possibility to use "device aware MPI", i. e. sending and receiving GPU buffers directly via MPI without the necessity of an intermediate CPU buffer. OpenCL currently does not have this feature.

Macro	OpenCL	HIP/CUDA
___GLOBAL	___global	(not needed)
___LOCAL	___local	___shared___
___PRIVATE	___private	___locale___
___CONSTANT	___constant	___constant___
___KERNEL	___kernel	___global___
___DEVICE	(not needed)	___device___

Table 3.2: Overview of some equivalent keywords in OpenCL and HIP/CUDA and the macro used in *lbm4e*

The concepts of the kernel languages used by OpenCL and CUDA/HIP are very similar. Most keywords used in the kernel language exists with the same function in the other languages as well. In order to allow usage of all presented frameworks, these keywords are not used directly but instead a macro is used that translates to the keyword depending on the desired kernel language (see table 3.2). The framework specific host code is wrapped into classes implementing a common interface. The implementation currently supports only OpenCL and HIP but as already mentioned, it should also be possible to compile the HIP code using CUDA. One challenge with the implementation was that OpenCL is run-time compiled which is used to compile options into the kernel directly, but HIP and CUDA are generally pre-compiled. CUDA provides a library for run-time compilation (NVRTC), which is not available in HIP. To circumvent the problem, a custom implementation of run-time compilation is provided that writes the code to a file at run-time, compiles the code to a shared library using a compiler invocation and then loads the created library using the dlopen function. This is limited to systems providing the dlopen function, namely UNIX like operating systems.

3.3.3 Data Structures

As stated before, LBM is bandwidth-limited and thus the implementation should be optimized for bandwidth. There are multiple requirements for the data structures used to store the distribution functions: It should be possible to merge the streaming and collision step in order to decrease the bandwidth requirements. Distribution functions should only be stored once as especially on GPUs the device memory is limited. The data structure must be suitable for GPUs and it should be possible to only save fluid nodes and omit wall nodes.

The so called Esoteric Twist (or short EsoTwist) algorithm by Geier and Schönherr (2017) solves all of these problems. Not all distribution values are directly stored in the cell they belong to. If a component of the direction vector is negative, they are saved to the cell in that direction (e. .g. the distribution function for $e_i = (-1, 1, -1)$ of cell (x, y, z) would actually be saved in cell $(x+1, y, z+1)$). For the combined collision and streaming step, the distribution functions of a cell are first read, new values are calculated according to the used collision operator and then saved at the position of the opposite distribution function. After the collision and streaming step is completed, the arrays for the distribution values f_i and $f_{\bar{i}}$ are swapped. If the distribution functions are saved in a structure of array arrangement, this can be achieved by simply swapping the pointers to the memory.

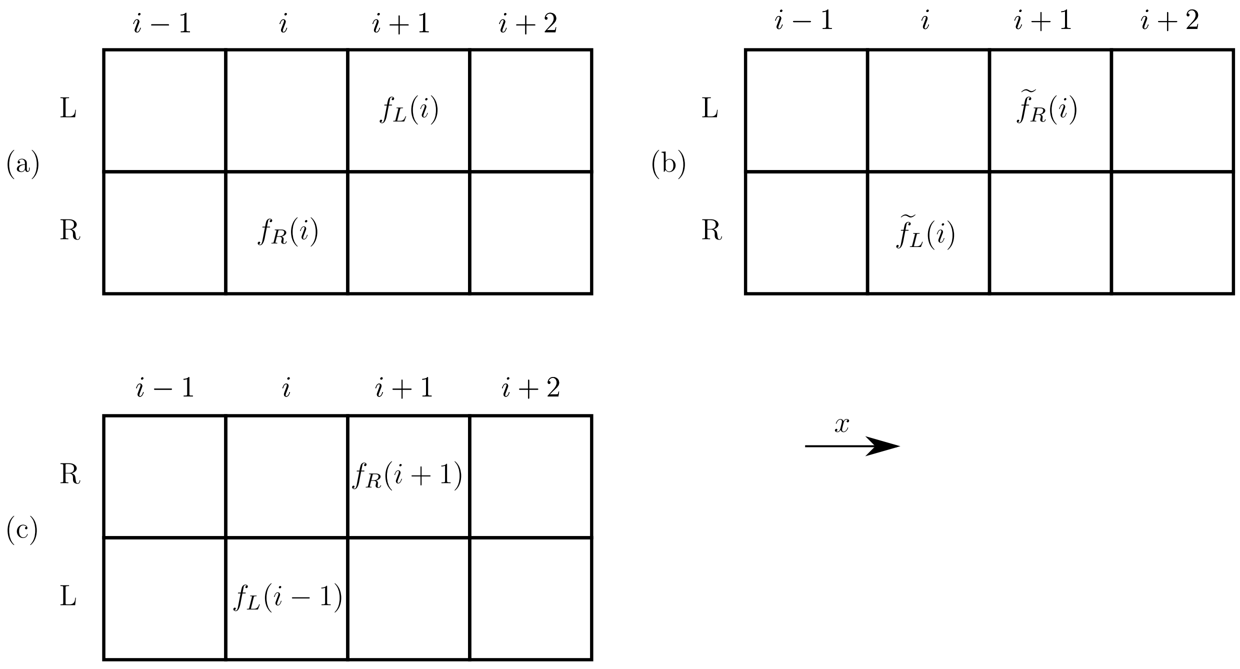


Figure 3.1: Illustration of the EsoTwist algorithm: (a) initial state; (b) state after collision; (c) state after streaming step

The whole process for a simple D1Q2-scheme is visualized in figure 3.1. The two possible directions are denoted as left (L) and right (R) with left being the negative direction. The function values are saved in a structure of arrays, meaning that all left values are saved in one array and all right values in a different one. As explained, the left value for the cell i, $f_L(i)$, is stored in its neighbor cell, here the cell $i+1$. This is shown in part (a) of the drawing. For the collision step, the distribution functions are read, the postcollision values $\widetilde{f}_L(i)$ and $\widetilde{f}_R(i)$ are calculated and then saved in the cell in reversed order, meaning that $\widetilde{f}_L(i)$ is saved in the array for the right values and vice versa. The state after collision is shown in part (b). The streaming step is then executed by switching the meaning of the arrays so that left becomes right and right becomes left. It can be seen (part (c) of the sketch), that the former post-collision function $\widetilde{f}_R(i)$ is now at position $i+1$ of the right array, which is the right function of cell $i+1$ and the former post-collision function $\widetilde{f}_L(i)$ in the left array at position i which due to the shift for negative pointing distribution functions belongs to cell $i-1$.

This algorithm has another important advantage: During the collision and streaming step, only the neighbored cells in positive direction are needed. Therefore, the connectivity can be saved by a linked list structure where for each cell a “pointer” (implementations with integer indices instead of pointers are also possible) to the cells in positive x, y and z direction is saved. Three “pointers” are sufficient as e. g. the neighbored cell in xy-direction can be retrieved as the neighbor in y-direction of the neighbor in x-direction. Using a linked list structure also has the advantage that only fluid cells plus an additional layer of cells in positive direction have to be saved which can significantly decrease the amount of memory needed. This is also useful as hierarchical grid refinement can be interpreted as multiple sparse grids connected by special boundary conditions, i. e. interpolation and averaging.

Another important aspect of data structures is the necessity to enable memory coalescing as only then the full memory bandwidth on GPUs can be utilized. Memory coalescing occurs when consecutive work-items[5] read or write on consecutive positions in memory. In order to achieve memory coalescing, a structure of arrays arrangement is used, furthermore the cells for averaging are placed together in memory as well as the cells for interpolation.

3.3.4 Asynchronous Processing

OpenCL offers the possibility to use an out-of-order queue, which executes the commands submitted to it in an arbitrary order, potentially in parallel. Dependencies between different kernels are handled by using events. Each function call returns an event object which can be used for creation of a list of events that models the dependencies of a function call. This allows to fully utilize the capabilities of the GPU as low intensity kernels can potentially be executed in parallel with other kernels. In contrast, HIP and CUDA only offer the possibility to use multiple command queues (called streams in CUDA terminology). A simple implementation that assigns kernels in a round-robin fashion to different streams and synchronizes according to the given dependencies could be used to allow an event based system for all interfaces. However, the performance of this simple implementation might be inferior to the native out-of-order queue of OpenCL.

[5]In this thesis the OpenCL termini *work-item* and *workgroup* are used instead of the CUDA/HIP equivalents *thread* and *block* as the CUDA/HIP usage of *thread* conflicts with the normal definition of threads being independently executable.

3.4 Grid Generation

This work was done in cooperation with Feng Gu of the research group for Graphical Data Processing and Multimedia. Feng Gu implemented the creation of the BVH on GPUs as well as the watertight intersection algorithm. He also proposed the alternative approach for the point-in-polygon test. I developed and implemented the CPU based meshing as well as the basic algorithm for the GPU based meshing.

3.4.1 General Requirements

The ability to easily generate meshes for LBM is often listed as a key advantage for LBM over classical CFD. Surprisingly, the amount of publications addressing meshing is relatively limited (Janßen et al., 2015; Pasquali et al., 2013), although meshing - especially for refined grids - is not trivial.

Meshing means to create a list of cells for a given specified computational domain, number of levels, cell size and input geometries, most of the time given as a triangle mesh. The wall position as described in section 2.5 must also be calculated and cells inside the geometry should be excluded. There are further restrictions for this implementation: Only a refinement factor of two is supported and cells can only be refined completely as otherwise averaging becomes difficult. This means that a refined cell (in three dimensions) must be replaced by eight finer cells. Furthermore, there must not be any multi-level jumps in the mesh, i.e. between each level there should be a minimal number (given by the user) of cells. Additionally, all cells neighbored to the geometry have to be on the finest level.

3.4.2 CPU Based Meshing

One option for meshing is to first calculate all wall distances by iterating over the triangles, generating their axis-aligned bounding box (AABB), enlarging this bounding box slightly so it aligns with the grid and then using the cells on the surfaces of the AABB as starting points for the intersection calculation. The calculated wall distances are then saved in an octree. In order to distinguish between in- and outside, the user must provide a point inside the mesh which cell is used to determine the cells of the mesh based on connectivity to this cell. Connectivity can be determined by using breadth-first search (BFS) (Cormen et al., 2009, 594 ff.) or depth-first search (DFS) (Cormen et al., 2009, 603 ff.). The minimum distance between the levels is then enforced by using the cells neighbored to the wall, i. e. cells having at least one wall distance saved, as an initial list of cells for breadth-first-search which can ensure one layer of cells per iteration. Since an octree is used, the criteria that cells must be fully refined or not refined at all can easily be ensured.

This algorithm has some drawbacks: Most ray-triangle intersection algorithms are not numerically robust meaning that in some cases they report actual intersection as misses. As the whole distinction between in- and outside is based on the correct calculation of the intersections, a single error here can be fatal. Another drawback is that even though both BFS and DFS can be parallelized to some extent; they are not very well suited for usage on GPUs.

3.4.3 GPU Based Meshing

A different approach is to use recursive refinement of an octree. Starting from a single cube as an initial cell, it is first checked if the cell is completely outside the domain. If it is completely outside the domain or completely inside the geometry, it is not processed further. If it intersects with the domain boundaries or with the geometry or if it is larger than the cells of the coarsest grid level, the cell is refined to 8 smaller cubes which are then processed recursively until the cell length matches the desired resolution for the finest level. To ensure that all cells neighbored to the geometry are on the finest level, the cell has to be enlarged by at least half of the length of the finest cells before checking for an intersection with the geometry. If the cell is enlarged further, this can be used to enforce the minimum distance between two levels. The whole process is illustrated in figure 3.2.

Although recursive calling of functions on GPUs is not possible, this algorithm can be modified to be suitable for GPUs: Instead of recursive refinement, which would lead to a DFS traversal of the tree, a BFS traversal can be used, meaning that with one kernel invocation all cells on the current level are checked and - if appropriate - refined. The number of cells on the next level can be accumulated by using atomic functions so that before the next kernel invocation the appropriate amount of memory can be allocated.

One drawback of this algorithm is, that the possible parallelism for the first tree levels is very small, which might not allow to fully utilize hardware capabilities at the beginning. Furthermore, this method also requires a robust point-in-polygon test. On the plus side, all children of a particular node can be processed in parallel. This makes parallelization to multiple GPUs possible or allows to generate the mesh piecewise if the device memory is too limited.

3.4.4 Watertight Triangle Intersection

A numerically robust ray-triangle intersection algorithm has to be used in order to generate correct meshes. Two algorithms from literature have been used to achieve this:

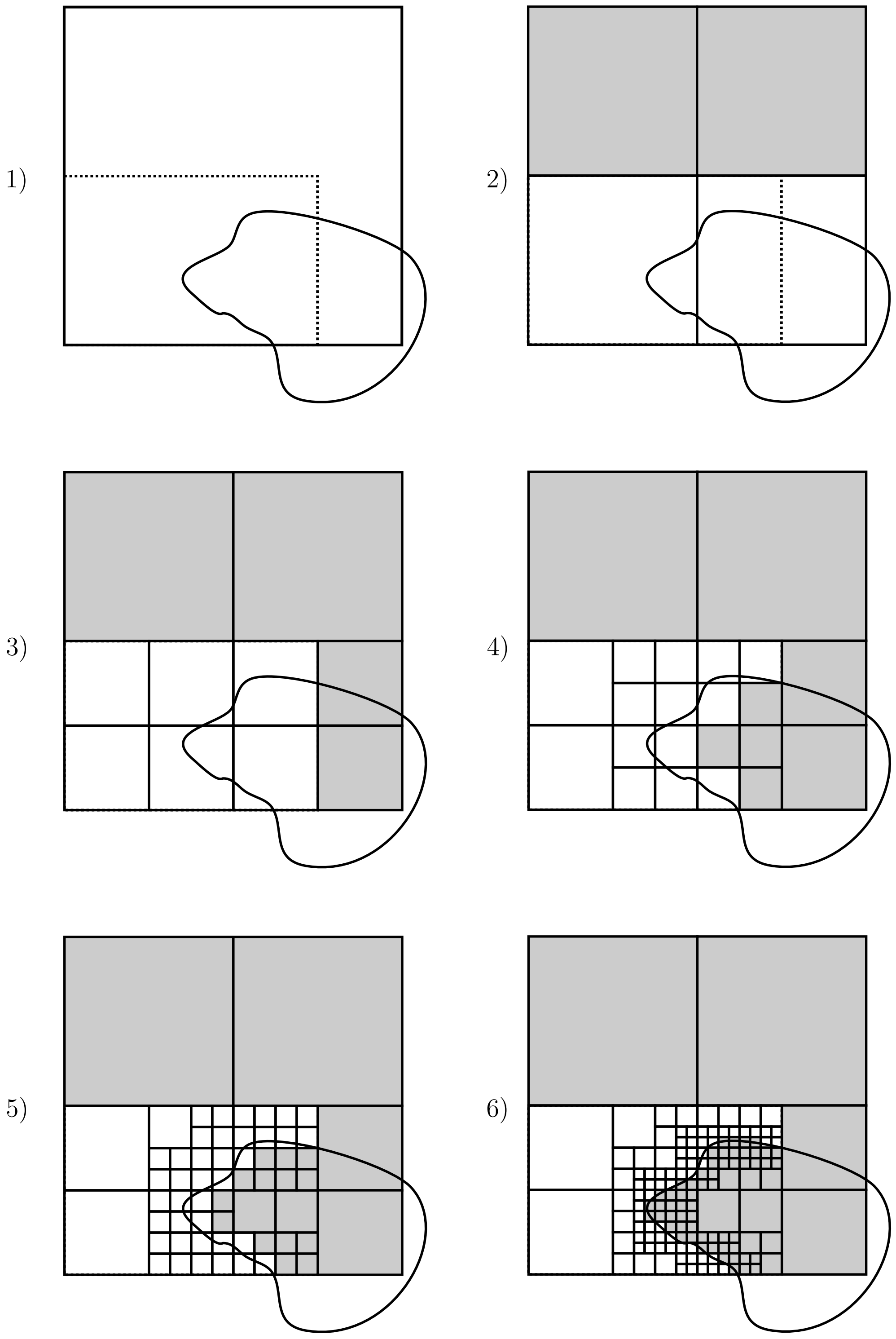


Figure 3.2: Illustration of the grid generation algorithm. The computational domain is drawn with a dotted line. The curved shape is the geometry. The gray cells are cells that only exist for the completeness of the octree and are not part of the mesh.

The algorithm by Dammertz and Keller (2006) does not perform a direct ray-triangle intersection. Instead a ray is tested against the AABB of the triangle, if there is an intersection the triangle is subdivided into four smaller triangles by halving each edge of the original triangle. The intersection with these triangles is then recursively checked until no further subdivision is possible because of limited floating point accuracy. Then, the result of the intersection test between the AABB and the ray is reported as the result of the ray-triangle intersection. One problem is that if the ray intersects the triangle in a parallel way, i. e. the result is a line instead of a single point, an arbitrary point on the triangle is given as intersection and not necessarily the closest point to the ray origin. This can be solved by processing sub-triangles that are closer to the origin first. Although this algorithm is numerically robust, it has two drawbacks: It is noticeably slower than other algorithms due to the high amount of recursion needed and due to its recursive nature it is not suitable for GPUs.

Woop et al. (2013) developed another watertight intersection algorithm. It performs calculations with single precision and falls back to double precision when single precision is insufficient. The performance is comparable to other non-robust intersection algorithms. It requires IEEE-754 floating-point standard confirming treatment of floating point operations in order to work correctly. Although this generally is mandatory for OpenCL implementations, some optimizations can break IEEE-754 conformance. The following code snippet illustrates a problem caused by optimization:

```
float a = 0.2;
float b = 0.3;
if( a*b - a*b == 0 ){
        printf("this␣should␣be␣true\n");
}
```

Normally the printf-statement should be executed as the product $a \cdot b$ is rounded to the same result both times. Nevertheless, the compiler could choose to optimize the calculation of $a \cdot b - a \cdot b$ by first calculating $c = -a \cdot b$ and then using a multiply-add (mad) instruction to calculate $a \cdot b + c$. As a mad instruction can be implemented with only one rounding, the result does not have to be zero. It was found that some OpenCL implementation use this sort of optimization although optimizations are disabled, leading to wrong ray-triangle intersections.

Generally, a bounding volume hierarchy (BVH) should be used to accelerate the calculation of intersections. A BVH is a tree structure of bounding volumes (usually axis aligned bounding boxes or spheres) so that for each node the bounding box of the node covers the bounding boxes of its children. Instead of testing the ray-triangle intersection with all triangles, the tree is recursively traversed starting from the root node. Children of a node have only to be processed if their bounding volume has an intersection with the ray. If multiple children have an intersection, the processing should be continued with the child having the nearest intersection first. There are multiple ways of building BVHs, usually divided into bottom-up and top-down approaches. Here, the method by Karras (2012) is used as it is suitable for implementation on GPUs.

3.4.5 Point-in-Polygon Test

There are multiple possibilities to determine whether a point is inside a polygon or not:

One can cast a ray from the far outside to the given point and count the number of intersections until the point is reached. If the number is odd, the point is inside the polygon and if the number is even, outside. Although simple in concept, this algorithm is difficult to implement in a robust way. First, it obviously requires watertight triangle intersection. Furthermore, there are some special cases that must be considered: If the ray hits the edge between two triangles, this must only be counted as a single intersection. Furthermore, if the ray lies in the same plane as the triangle and therefore the intersection is a line instead of a single point, this must be handled as one intersection at the beginning of the line and another one at the end.

An alternative approach is to cast a ray from the point away in an arbitrary direction and determine the first intersected triangle encountered. Then, it is checked whether the ray and the normal vector of the triangle point towards the same direction by checking if the dot product between the ray and the normal vector is positive (assuming that the normal vectors point to the outside). If this is the case, the point is inside the geometry. One problem arises if the triangle found is orthogonal to the ray as then no decision is possible. A pragmatic solution for this problem is to cast multiple rays in random directions from the point and use a majority vote for the result.

3.5 Co-Processing and Visualization

3.5.1 Motivation

Traditionally, evaluation of the results in CFD is done by post-processing, i. e. saving time steps during the calculation and then after the calculation reading the saved data again and processing it. This is a good process for steady-state simulations as only the last time step has to be evaluated, but for transient simulation the amount of data that can be analyzed might be limited by IO speed. As stated for the LBM method, approximately 300 bytes have to be read and written for a single cell and time step and four macroscopic values (e. g. the pressure and the velocity components) have to be written per cell for post-processing. Although the amount of data needed for the calculations is therefore approximately ten times higher, the device memory bandwidth (ca. 1 TB/s) is roughly by a factor of 1000 higher than IO bandwidth (around GB/s for SSD drives). Therefore, only about 0.1 % of the calculated data can be used for evaluation without delaying the simulation.

To overcome this issue, co-processing can be used. Instead of writing the data to the disk, the time step is immediately evaluated and only the result in a single value, a file containing parts of the mesh, an image or similar is saved. This requires the code to have the capabilities to perform the desired evaluation. In this work, the VTK and Paraview Catalyst are used for this purpose. Although VTK is mainly CPU based, this can be used as an advantage: If a GPU is used for the calculation, the CPU only needs to handle the small overhead for GPU management and therefore is free for co-processing. As long as the co-processing is not too computationally demanding, it can be performed without increasing the total simulation time. The downside of using co-processing is that if a different evaluation is desired, the whole simulation has to be repeated instead of just re-evaluating the generated data sets.

3.5.2 Data Structures

Using VTK for co-processing requires converting the data first to a VTK data structure. There are mainly two data structures available in VTK which are suitable for LBM data:

An unstructured grid (vtkUnstructuredGrid) is a collection of arbitrarily shaped cells without any regular topology. Note that due to the refinement, the grids used by LBM cannot be treated as structured grids. The LBM grid can be directly treated as an unstructured grid which is an easy to implement and straightforward approach. To decrease the data size, points belonging to multiple cells should be saved only once. The disadvantage of this approach is that for interpolation inside the cells, that is necessary for some filters and smooth display of the data, the macroscopic data has to be interpolated to the cell points. A further disadvantage is that the t-shaped connection at the level jumps can lead to visual artifacts (Harel et al., 2017).

Another approach is using the unstructured grid structure, but creating a dual mesh, i. e. creating a mesh in which cell points are the actual cell centers. The macroscopic data calculated by LBM at the cell centers of the LBM mesh can then be directly used as point data for the dual mesh. Furthermore, the wall positions can be used to create cells that match the geometries surface using an algorithm similar to the marching cubes algorithm by Lorensen and Cline (1987). Smooth transfer at level boundaries is also possible. The disadvantage is that the dual mesh approach is more difficult to implement and requires more conversion time.

Although the grids used by LBM are not structured grids, they do have some sort of structure which can be used for higher efficiency. In the later versions of VTK, a grid of hyper trees (octrees in 3D) has been introduced. Using this data structure allows for a much more memory efficient representation of the data. Nevertheless, the code in VTK still is not production ready and not all VTK filters, but only selected specialized ones can be used with this data structure.

3.5.3 GPU Filters

One problem when implementing co-processing with VTK is that VTK assumes that the data resides in RAM and not in device memory. Simply copying all macroscopic data to RAM each time step is not possible as the connection between GPU and RAM, usually PCI Express, only offers data transfer rates in the order of $10\,\frac{\mathrm{GB}}{\mathrm{s}}$. Therefore, custom filters that directly operate on device data such as a slicing, clipping or integrating were implemented. This allows to limit the data transferred to a subset of the original data.

Generally, filters form a tree structure. After the completion of a filter all its subsequent filters can be executed independently. In order to achieve maximum performance, the filters have been included in the event based asynchronous scheduling scheme. Filters processed on the CPU are parallelized with a thread pool.

3.5.4 In-Situ Visualization

Often, images or animations are desired as a result of the simulation. Paraview Catalyst can be used to generate those. The simulation needs to convert the data to one of the VTK data structures and call the Paraview Catalyst co-process function. Paraview Catalyst must be initialized with one or more python scripts that are then executed when the co-process function is called. These scripts can be created manually or with a graphical generator in Paraview. All functionality of Paraview is available to the script, also allowing for generation of images or videos in-situ. In addition, it is possible to connect with Paraview to a running simulation and process and visualize the results live.

3.6 Validation

The implementation was validated by Heng Li in his master's thesis "Validierung und Verifizierung des Lattice-Boltzmann-Codes lbm4e". The flow for three test cases was calculated and compared to experimental and numerical results:

The lid driven cavity is a square domain with one moving and three fixed walls leading to formation of a prime vortex and multiple secondary vortices. Position of the vertices as well as velocity profiles were compared to those by Ghia et al. (1982) and Hou et al. (1995).

In the backward-facing step flow, a channel suddenly widens causing flow de- and reattachment. De- and reattachment positions were studied, as well as velocity profiles and compared to experimental data by Armaly et al. (1983) and numerical data by Erturk (2008).

The third test case was the flow around a cylinder. Drag coefficient and detachment angle were evaluated for stationary flow and drag coefficient, lift coefficient and Strouhal number for transient flow. Results were compared to experimental data (Clift et al., 2005; Roshko, 1993) and numerical data (Dennis and Chang, 1970; Nieuwstadt and Keller, 1973; Fornberg, 1980; Saiki and Biringen, 1996).

Good agreement was achieved between results of *lbm4e* and data from literature in all three test cases.

4 Pressure Drop of Sphere Packed Beds

The work on particle packings is a continuation of my master's thesis "Skalenauflösende Berechnung der Permeabilität in inhomogenen, porösen Medien" and figure 4.1 is strongly based on a figure from that thesis. The results of this chapter have been published in the following papers:

1. Hofmann, S., Bufe, A., Brenner, G., and Turek, T. (2016). Pressure drop study on packings of differently shaped particles in milli-structured channels. *Chemical Engineering Science*, 155: 376-385.
2. Bufe, A., and Brenner, G. (2018). Systematic Study of the Pressure Drop in Confined Geometries with the Lattice Boltzmann Method. *Transport in Porous Media*, 123: 307-319

Section 4.3 describes the work published in the first paper. The other sections are strongly based on the second publication. All figures and their respective captions (except figure 4.1) have been taken from the second publication.

4.1 Introduction

Particle packings are of great importance for chemical engineering e. g. in catalyst beds. Thus, knowledge of flow characteristics such as pressure drop is crucial. Despite that importance and much research on this topic, a universal correlation of the pressure drop in packed beds is unknown. The pressure drop can be described in dimensionless form as the product of two dimensionless numbers: the friction factor f and a modified Reynolds number Re. For Stokes flow, the dimensionless pressure drop is usually expressed as $f \cdot Re = A' \cdot A_W$, where A' is the dimensionless pressure drop of an infinitely extended bed and A_W a wall correction term, usually depending on porosity and the ratio of reactor hydraulic diameter to sphere diameter.

The pressure drop of a particle packing confined by walls changes due to changes in the structure near to the wall. Figure 4.1 shows the porosity and velocity averaged for a slice as a function of height for a sphere packing between infinite extended plates. As it can be seen, the porosity next to the wall is basically one as the spheres only touch the walls.

Furthermore, the porosity over height fluctuates as the wall force the otherwise random packing into a structure. The pressure drop of a packing confined by wall is therefore different for two reasons: First, the regular structure close to the walls can create “channels” where the flow can go straight through the reactor and thus has a shorter path compared to a random packing. Second, the confining walls add surface and therefore friction.

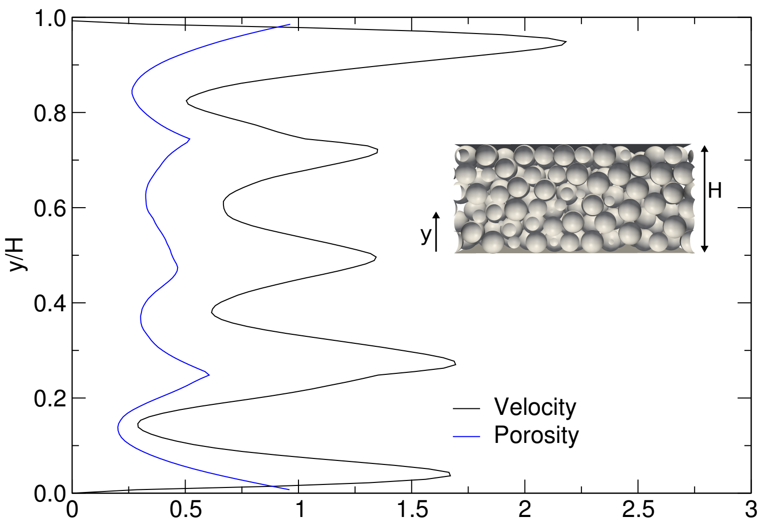


Figure 4.1: Averaged Porosity and Velocity (normalized by mean velocity) over height for a sphere packing between infinite extended plates. Note that the package drawing is only for illustration and not the packing used to create the profiles.

Even for the special case of spheres in an infinite extended bed, where $A_W = 1$, the experimental findings and derived correlations differ widely. For example, Ergun (1952) proposed a value of 150 for A', whereas the Carman-Kozeny equation (Carman, 1937) states a value of 180. Table 4.1 gives a short overview of some common correlations. As can be seen, the prefactor in the wall correction term also varies. The later introduced theory behind these correlations also leads to a dependence on tortuosity. For practical reasons, many correlations do not depend on tortuosity as it is difficult to be determined. In contrast, the porosity can be easily measured e. g. by weighing. Due to that reason, the pressure drop is treated as a function of easily measurable quantities alone.

There are different ways to obtain the geometrical information needed for a simulation. One can scan real particle packings using techniques like computer tomography. A different approach would be to use numerically generated packings. As experiments or scanning of particles is laborious, they only allow for analysis of few cases whereas numerically generated packings can easily be generated in large numbers. A drawback might be that these calculated packings must not necessarily be realistic and thus might give different results. Both approaches are explored in this chapter.

Author	A'	A_W
Ergun (1952)	150	-
Carman (1937)	180	-
Macdonald et al. (1979)	180	-
Mehta and Hawley (1969)	150	$1+\frac{2}{3}\frac{1}{\frac{D}{d}(1-\epsilon)}$
Liu et al. (1994)	$\frac{85{,}2}{\epsilon^{\frac{2}{3}}}$	$1+\frac{\pi}{6}\frac{1}{\frac{D}{d}(1-\epsilon)}$
Zhavoronkov et al. (1949)	163,35	$1+\frac{1}{2}\frac{1}{\frac{D}{d}(1-\epsilon)}$
Eisfeld and Schnitzlein (2001)	154	$1+\frac{2}{3}\frac{1}{\frac{D}{d}(1-\epsilon)}$

Table 4.1: Overview of some common correlations of nondimensional pressure drop in a particle filled tube

4.2 Pressure Drop and Porosity Correlations

To model the pressure drop in packings of particles, usually the concept of hydraulic diameter is used. The hydraulic diameter d_h is defined as four times the fluid volume divided by the packings surface (equation 4.1). The packings surface is the sum of particle surface, which can be expressed in terms of porosity ϵ, sphere diameter d and the surface of the reactor, which is a function of the hydraulic diameter of the reactor D (equation 4.2). Using this definition, one obtains equation 4.3.

$$d_h = 4\frac{V_{\text{fluid}}}{S_{\text{spheres}} + S_{\text{reactor}}} \tag{4.1}$$

$$= 4\frac{V_{\text{fluid}}}{\frac{6}{d}(1-\epsilon)V_{\text{reactor}} + \frac{4}{D}V_{\text{reactor}}} \tag{4.2}$$

$$= \frac{2}{3}d\frac{\epsilon}{(1-\epsilon)}\left(1+\frac{2}{3}\frac{d}{D}\frac{1}{(1-\epsilon)}\right)^{-1} \tag{4.3}$$

The packing is modeled as a bundle of unconnected pores, each having the hydraulic diameter d_h. As the exact form and length of that pores are unknown, a generalized form of the Hagen-Poiseuille equation (equation 4.4) is used.

$$\frac{\Delta P}{L}\frac{d_h^2}{\mu u} = k \tag{4.4}$$

Inserting equation 4.3 into equation 4.4 leads to the final equation 4.5 with the unknown parameter A, where $\Lambda = f \cdot Re$ denotes the dimensionless pressure drop. The parameter A'

is a function of geometrical properties of the packing. Carman (1937) obtained a value of 180 by using a value of 40 for the constant k, the mean between the solution for a pipe (32) and for a channel (48), and modified it by a factor of two, grounded in considerations on tortuosity. In other work, the parameter A' is determined by experiments. The numerous correlations available in literature not only differ in the parameter A', which is usually between 150 and 180, but also in the treatment of the wall correction term A_W as some use a factor different from $\frac{2}{3}$. A more detailed presentation and a good overview over the correlations available can be found in the paper by Eisfeld and Schnitzlein (2001).

$$\underbrace{\frac{\Delta P}{L}\frac{d}{\rho u_0^2}\frac{\epsilon^3}{(1-\epsilon)}}_{f} \cdot \underbrace{\frac{\rho u_0 d}{\mu(1-\epsilon)}}_{Re} = \underbrace{\overbrace{\frac{9}{4}}^{A'}k \cdot \Big(\overbrace{1+\frac{2}{3}\frac{d}{D}\frac{1}{(1-\epsilon)}}^{A_W}\Big)^2}_{A} \tag{4.5}$$

This equation can easily be extended to flows that are not in the limit of small Reynolds numbers. Since only the case of a negligibly small Reynolds number is investigated in this work, the extension shall not be further discussed.

The equation predicts an increase in the dimensionless pressure drop as the $\frac{d}{D}$ ratio increases and therefore the surface of the reactor increases, which can be explained by increased wall friction. Another adverse effect of confining walls is the increase of local porosity near the walls which decreases the dimensionless pressure drop. Eisfeld and Schnitzlein (2001) concluded that the first effect is predominant for small Reynolds numbers, whereas the latter occurs at high Reynolds numbers.

For the cases of simple cubic, body centered cubic and face centered cubic arrays of spheres, Sangani and Acrivos (1982) derived analytical solutions based on work by Hasimoto (1959) (table 4.2). Their solutions are given in the form of a dimensionless drag (equation 4.6), which can be converted into a dimensionless pressure drop by equation 4.7. The different results for the three packings also show that there is no universal parameter A' as this parameter is a function of geometrical properties like tortuosity.

$$K = \frac{F}{3\pi\mu u_0 d} \tag{4.6}$$

$$f \cdot Re = 18K\frac{\epsilon^3}{(1-\epsilon)} \tag{4.7}$$

4.3 Scanned Packings

This work was done in cooperation with Sebastian Hofman of the Institute of Chemical

	K	ϵ	$f \cdot Re$
simple cubic	42,1	0,4764	156
body centered cubic	163	0,3198	141
face centered cubic	438	0,2595	186

Table 4.2: Analytical solutions for regular packings in touching configuration. Values of K are taken from Sangani and Acrivos (1982).

and Electrochemical Process Engineering, who conducted the experiments. Packings were scanned at Bundesanstalt für Materialforschung (BAM), Berlin.

In order to quantify the wall effect beds of spherical and non-spherical particles with different diameters were created and the pressure drop for low Reynolds numbers measured. Particle beds were also scanned using micro computer tomography. The scans were binarized by a threshold procedure and directly used as grids for LBM with the simple bounce back rule. This allowed for direct comparison between numerical and experimental results. Good agreement was achieved for non-spherical particles whereas for spherical particles the pressure drop was overestimated by approximately 10 %. There are multiple possible reasons for deviations: The bed scanned was not the same that was used for the experiments, but a different one created with a same sized container and particles. The simple binarization scheme and limited grid resolution might lead to a further error. Some of the experimental results are also highly questionable: For the non-spherical particles Sauter diameter, volume equivalent diameter and sphericity have been measured. The sphericity Ψ of a particle (equation 4.8) is defined as the ratio of the surface of a volume equivalent sphere to the surface of the particle. It can also be interpreted as the ratio of Sauter diameter d_S to the volume equivalent diameter d_V. Since the surface to volume ratio is optimal for a sphere, the sphericity equals one for a sphere and is smaller for all other geometric shapes. Therefore, the Sauter diameter must always be smaller than the volume equivalent diameter. In contrast, it is the opposite case for the measured values as the Sauter diameter is always higher.

$$\Psi = \frac{\pi^{\frac{1}{3}}(6V_P)^{\frac{2}{3}}}{A_P} = \frac{d_S}{d_V} < 1 \tag{4.8}$$

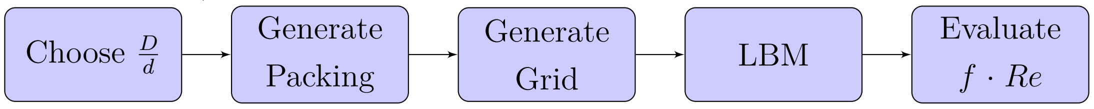
4.4 Synthetic Packings

4.4.1 Generation of Packings

Another approach is to obtain the needed geometrical information by simulating the packing generation. The generator usually takes the number of spheres and geometrical information of a surrounding box and delivers the positions and diameters of the spheres in a random close packing. Though these artificially created packings do not necessarily resemble all relevant geometrical characteristics of real packings, this approach has the advantage of fast generation of numerous packings so that parameters like the geometry of the surrounding box can be easily varied. To rule out artifacts created by the packing algorithm, two different algorithms were used, namely a modified Lubachevsky-Stillinger algorithm (Donev et al., 2005a,b) and the algorithm by Desmond and Weeks (2009).

The modified Lubachevsky-Stillinger algorithm uses event driven molecular dynamics to create random close packing. While the radius of the particles is slowly increased, they collide with the walls and other particles until a jammed state is reached. Details can be found in Donev et al. (2005a,b). The fortran implementation “packLSD” Donev et al. (2005b,a) which is freely available is used here, as well as a C++ implementation that can handle particles in a cylindrical box.

The algorithm by Desmond and Weeks initially places the particles at random positions and then slowly swells and contracts them until a jammed state is reached. Overlaps are handled by an energy minimization technique. Details can be found in Desmond and Weeks (2009). There is also an implementation freely available called “rcp generator”(Desmond and Weeks, 2009).



4.4.2 Used LBM Models

The D3Q19 model and the two-relaxation-time approach was used for the calculations (Ginzburg et al., 2008; d’Humières and Ginzburg, 2009). To satisfy the no-slip conditions at immersed boundaries, the multi-reflection boundary conditions developed by Ginzburg and d’Humières (2003) were applied. If there are not enough fluid nodes available to use these boundary conditions, the calculation fell back to the simple bounce back rule. As the multi-reflection boundary conditions and the two-relaxation-time model together lead to viscosity independent measurements, the viscosity was set to $\nu = 0.5$ in lattice Boltzmann units to speed up the calculations as Pan et al. (2006) showed that a higher viscosity leads to a smaller number of iterations required to reach a steady state. The flow through the porous media is driven by a constant pressure gradient which acts as a volume force in the momentum balance until a steady state solution is obtained. The pressure gradient was chosen, so that the flow is in the limit of a small Reynolds number. The steady flow state was considered as reached, if the change of the velocity in main flow direction between 100 iterations relative to the minimum of both velocities were less than 10^{-5} or mathematically if $\frac{|u_i - u_{i+100}|}{min(u_i, u_{i+100})} < 10^{-5}$. Subsequently, the mean velocity in the pore space obtained from the numerical solutions, together with the pressure gradient were evaluated. This setup is very similar to that of Pan et al. (2006), so that details may be found there. Since the multi-reflection boundary conditions increase the computational load, first the flow until steady state was calculated with the simple bounce back rule and then the simulation were continued with the multi-reflection boundary conditions until a steady state was reached again.

4.4.3 Influence of Grid Resolution

Regular Packings

In order to choose a sound grid resolution for the calculations, first the influence of the grid resolution for a regular packing was investigated. The cases of a simple cubic and a body-centered cubic array of spheres in touching configuration were chosen as their porosities are similar to the random close packing limit and an analytical solution is available(Sangani and Acrivos, 1982). Figure 4.2 shows the absolute value of deviation to the analytical solution in percent as a function of grid resolution and figure 4.3 the same for the body centered configuration. As it can be seen, the multi-reflection boundary conditions significantly increase the accuracy of the calculation. A resolution of approximately 30 cells per sphere diameter is sufficient to bound the error to less than one percent. In contrast, the simple bounce back rule converges extremely slowly. Even for the finest resolution, the error is greater than two percent. As no body-fitted, but a Cartesian grid is used, the representation of curved geometries is imperfect so that the error does not decrease monotonically with the increasing grid resolution. Note that the seemingly higher oscillations of the multi-reflection boundary scheme are caused by the logarithmic scale.

Irregular Packings

To ensure grid convergence, the results for the more realistic random close packings, the grid resolution was varied from 50^3 to 400^3 cells for a random close pack consisting of 125 spheres with all periodic boundaries. Since there is no reference value available, the value computed with the highest resolution and the multi-reflection boundary condition is used as a reference, which according to the previous calculations should be a good approximation. The results (figure 4.4) again show the superiority of the multi-reflection boundary conditions as even the calculation with the second coarsest grid of the multi-reflection boundary condition gives a higher accuracy than all other calculations with the simple bounce back.

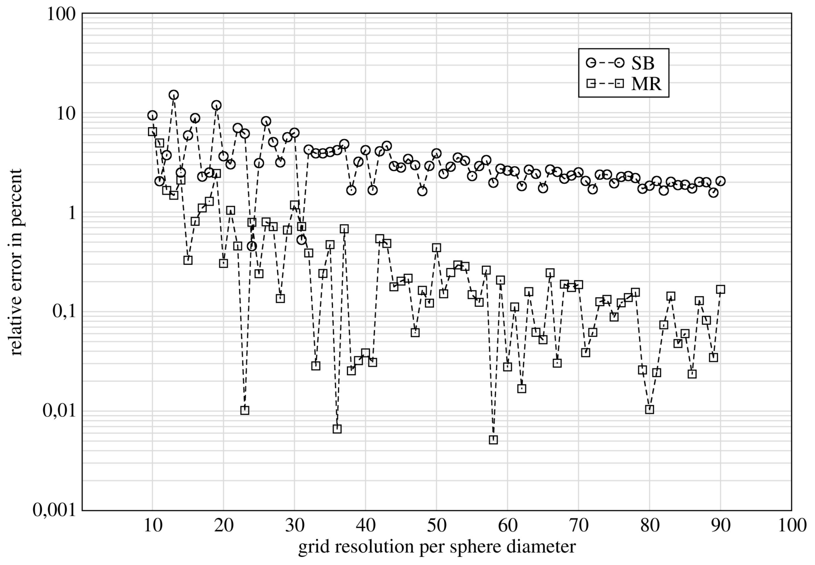


Figure 4.2: Absolute value of error (in percent) as function of grid resolution for a simple cubic array of spheres for simple bounce back (SB) and multi-reflection boundary conditions (MR). The analytical solution by Sangani and Acrivos (1982) is used as reference.

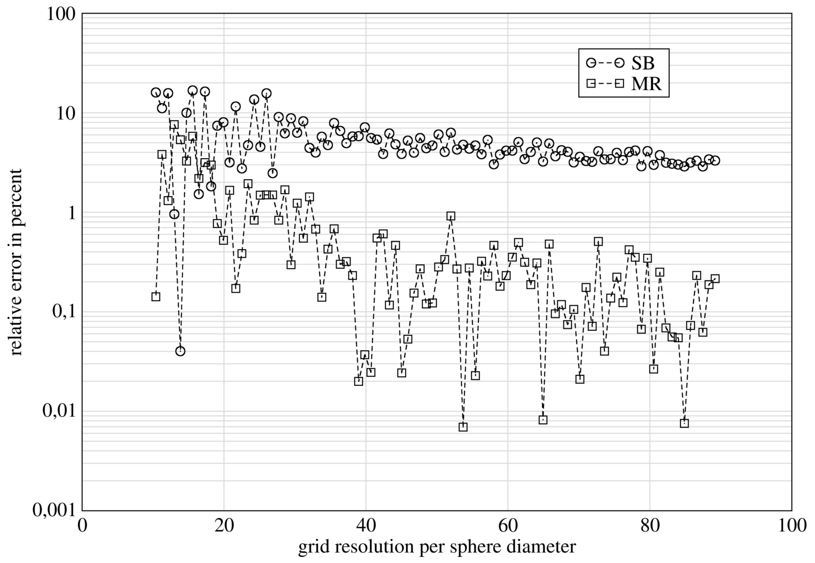


Figure 4.3: Absolute value of error (in percent) as function of grid resolution for a body centered cubic array of spheres for simple bounce back (SB) and multi-reflection boundary conditions (MR). The analytical solution by Sangani and Acrivos (1982) is used as reference.

For the following calculations, a resolution of 35 grid points per sphere diameter was chosen as this limits the deviation to the reference solution to less than one percent.

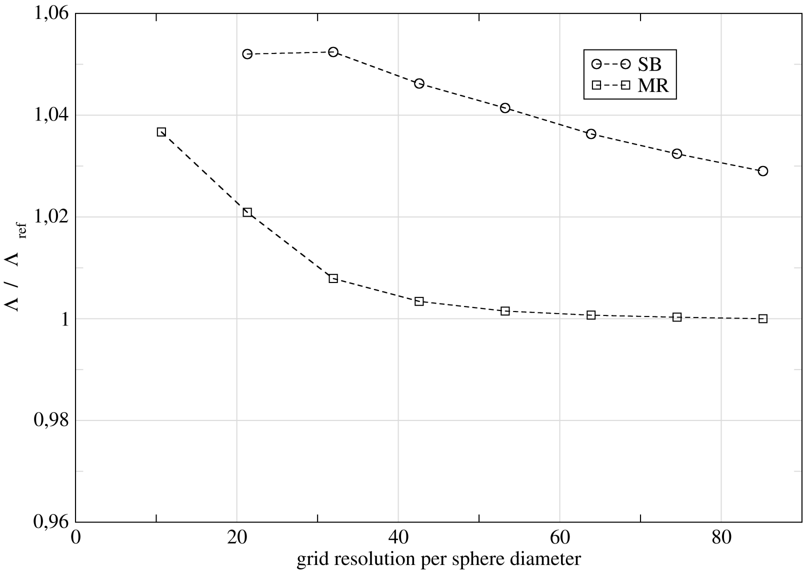


Figure 4.4: Relative error of dimensionless pressure drop as function of grid resolution for a packing of spheres for simple bounce back (SB) and multi-reflection boundary conditions (MR). The solution created with the finest grid and the multi-reflection boundary conditions is used as reference.

4.4.4 Pressure Drop Study

The pressure drop of four different configurations was studied: An infinite extended packing of spheres, which is modeled by periodic boundaries in all directions, spheres between two infinite extended parallel plates with distance H ($D_{hyd} = 2H$), spheres in a channel with square cross section and side length a ($D_{hyd} = a$) and a tube filled with spheres ($D_{tube} = D_{hyd}$). Sample packings of these configurations are visualized in figure 4.5. The packings are periodic in flow direction and the length was chosen so that the domain size in flow direction was at least ten times the sphere diameter.

Periodic Packing

First, the pressure drop of an infinite extended sphere packing (table 4.3) was simulated. Using both generators 10 packings each with a sphere diameter to packing length ratio of 10 were created. The achieved porosities are $\epsilon = 0.360$ for the packing generated with PackLSD and $\epsilon = 0.362$ for the packing generated with rcpgenerator. These values are very close to the random close packing limit of 0.36 (Song et al., 2008), corresponding to a densified

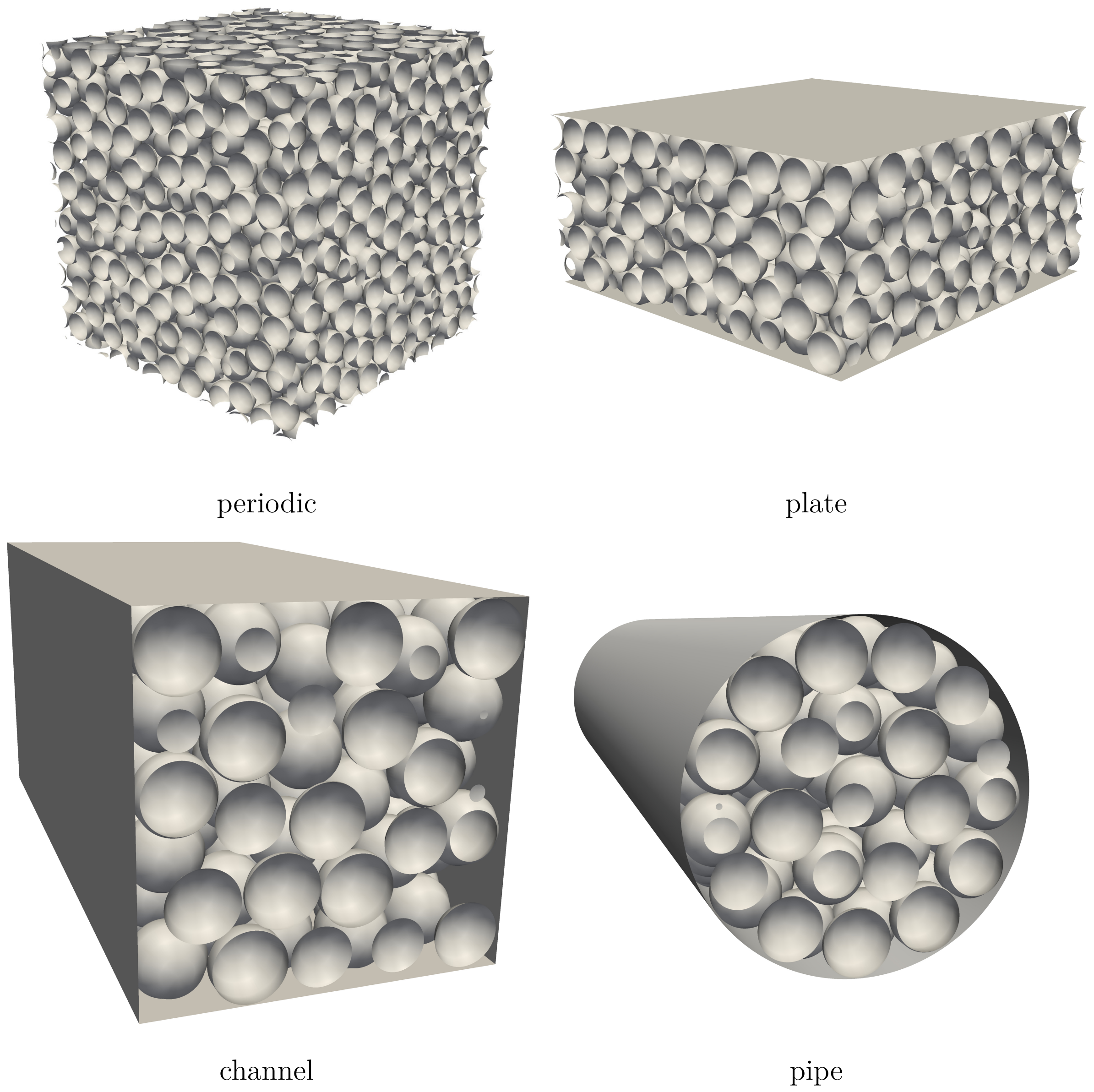


Figure 4.5: Sample packings of the different configurations.

	ϵ		$f \cdot Re$	
	mean	σ	mean	σ
rcp generator	0.3621	0.00109	172.37	1.90
packLSD	0.3598	0.00098	172.93	1.62

Table 4.3: Mean values and standard deviation (σ) of porosity and dimensionless pressure drop for periodic packings of spheres.

bed. The obtained averages of pressure drop are almost identical ($f \cdot Re = 172.9$ and $f \cdot Re = 172.4$ for rcpgenerator). These values are close to the value of the Carman-Kozeny equation (180).

Rectangular Configurations

For the plate geometry, numerous packings with different ratios of plate distance to sphere diameter ($\frac{H}{d}$) were created using both generators. Both curves (figure 4.6) show the same trend with packLSD creating slightly denser packings. The porosity increases with decreasing $\frac{H}{d}$, but tends to oscillate for small ratios. For certain ratios, the packing is no longer random but forced in an almost regular packing consisting of ordered sphere layers. A small increase in the ratio leads to an increase of porosity as the increase might be too small for creation of another layer of spheres thus forcing the spheres to a less dense configuration. A small decrease in the ratio can lead to a removal of a whole layer and thereby drastically increasing the porosity. These almost regular packings have a porosity lower than the random close packing limit. The form of the dimensionless pressure drop plot is similar. The slightly denser packings created by packLSD correspond to a slightly higher pressure drop. The fluctuations can be seen in pressure drop too, as the change of packing structure also affects pore geometry and tortuosity leading to a non monotonic behavior of the pressure drop as a function of $\frac{H}{d}$ ratio, which is in contrast to equation 4.5.

Since the results for both generators are very similar, only the "packLSD" generator was used in the following simulations.

The results for the channel with square cross-section are very similar to those of the plate configuration (figure 4.7). The porosities for same $\frac{H}{d}$-ratio are higher as the channel configuration is bounded in two dimensions. Therefore, the oscillations are also higher.

Pipe Configuration

Although porosity and dimensionless pressure drop also fluctuate for the pipe configuration (figure 4.8), these fluctuations are smaller and there are no super dense packings as the circular cross section prohibits those. Comparison of the calculated pressure drop to that

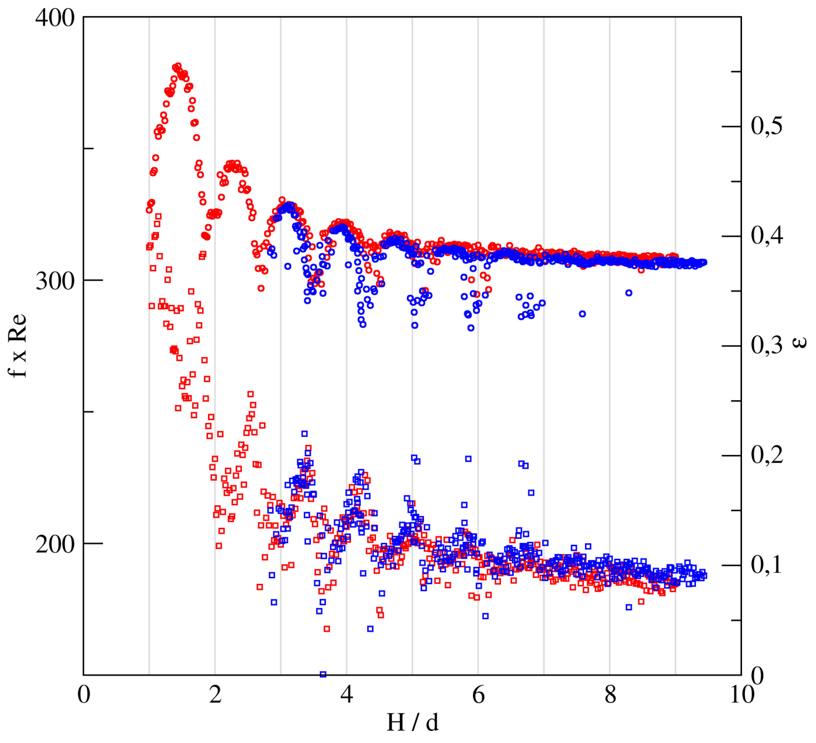


Figure 4.6: Top: porosity (circles), bottom: dimensionless pressure drop (squares) as function of plate distance H to sphere diameter ratio for plate configuration. Red: packings generated with rcpgenerator, blue: packings generated with packLSD

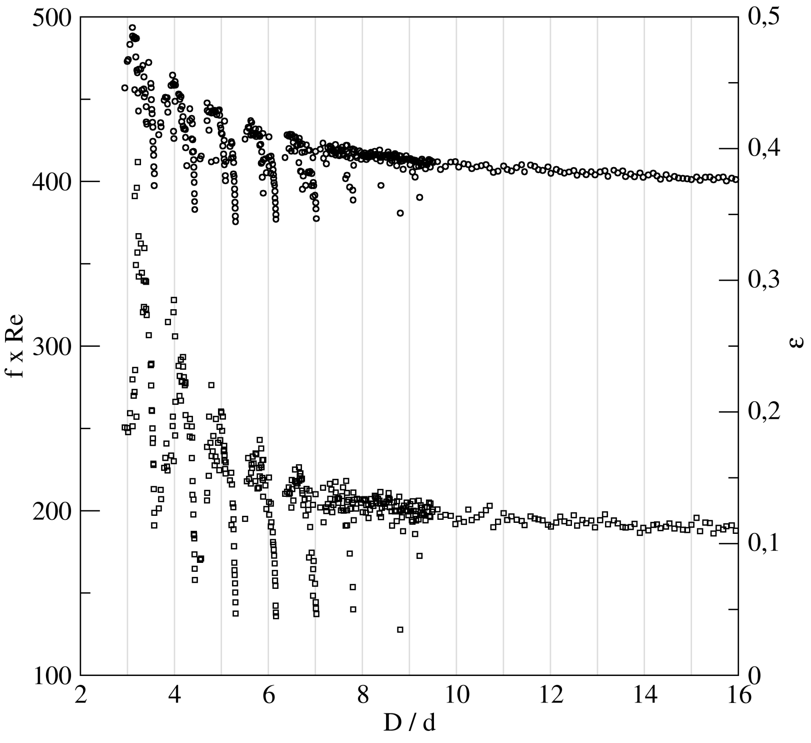


Figure 4.7: Top: porosity (circles), bottom: dimensionless pressure drop (squares) as function of hydraulic diameter D to sphere diameter ratio for channel configuration

predicted by the correlation of Eisfeld and Schnitzlein (2001) and that of Zhavoronkov et al. (1949) shows that the correlations average the fluctuations with good overall agreement. In this simulation, the correlation by Zhavoronkov fits better to the obtained values as the limit for infinite extended beds is closer to the result of the previous simulated periodic packings. The fluctuations of the pressure drop might also explain the differences in the pre-factor of the wall correction term as slight changes of the used $\frac{D}{d}$-ratio in the experiment might have a significant influence on the pressure drop obtained.

Comparison between the three configurations

Finally, the porosities (figure 4.9) and the dimensionless pressure drop (figure 4.10) of all three confined geometries are compared by using the concept of hydraulic diameter. The porosity profile of the pipe configuration shows smallest variations and the densest packings except for very small $\frac{D}{d}$ ratios. The channel and the plate configuration have the same limit, but the profiles vary greatly for small ratios due to oscillations.

The results for the dimensionless pressure drop are similar. For higher ratios, plate and channel configuration have similar values whereas the pipe configuration seems to have a lower dimensionless pressure drop. It is not clear if this smaller pressure drop is actually

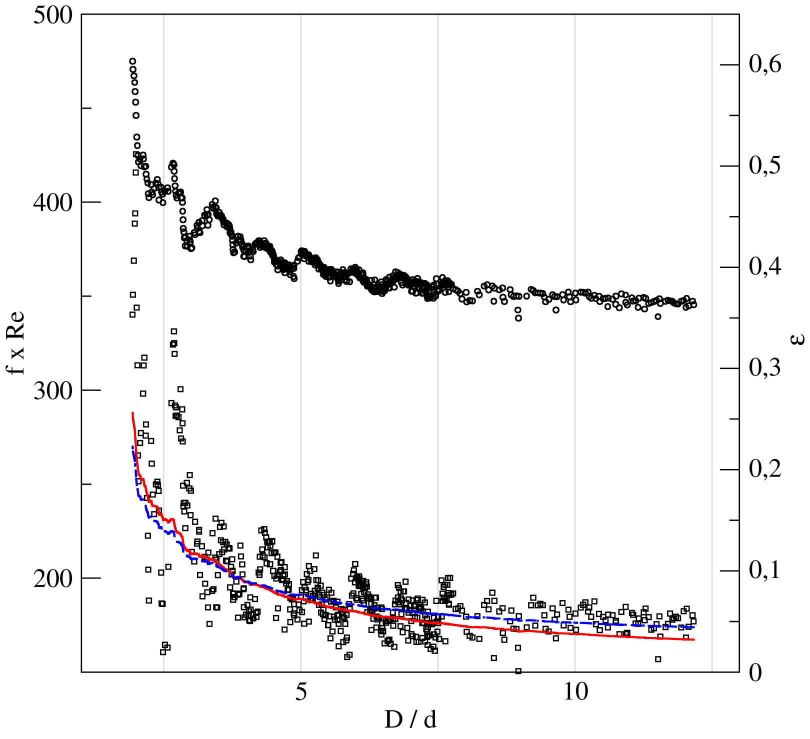


Figure 4.8: Top: porosity (circles), bottom: dimensionless pressure drop (squares) as function of hydraulic diameter D to sphere diameter ratio for pipe configuration, solid red line: correlation by Eisfeld and Schnitzlein (2001), dashed blue line: correlation by Zhavoronkov et al. (1949)

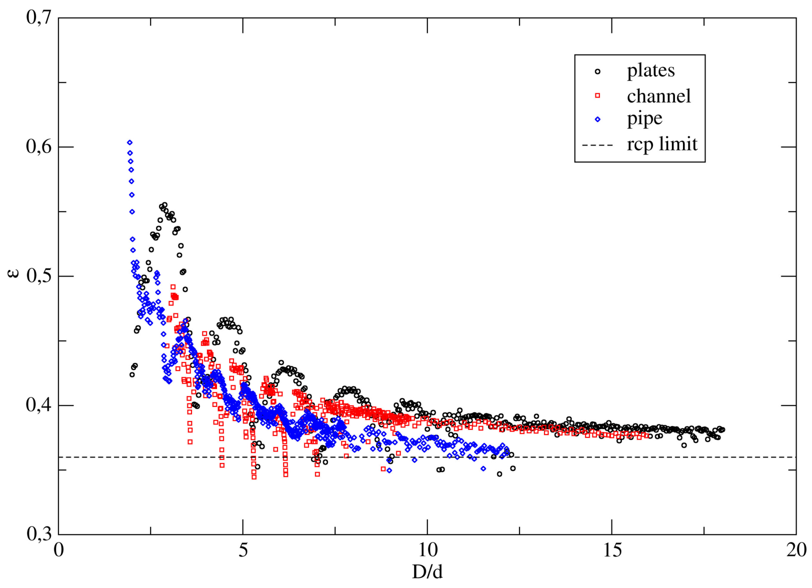


Figure 4.9: Porosity as function of hydraulic diameter to sphere diameter ratio for all three geometries. Dashed line: random close packing limit (0.36)

caused by the different configurations or by the different implementation of the packing generator. For small ratios there is a great deviation between all three configurations as in all three cases the dimensionless pressure drop changes drastically for only a slightly changed ratio. Another important difference between the channel configurations and the others is that in a channel configuration an area in the corners of the channel will always be empty, which might significantly effect the tortuosity, especially for small $\frac{D}{d}$ ratios.

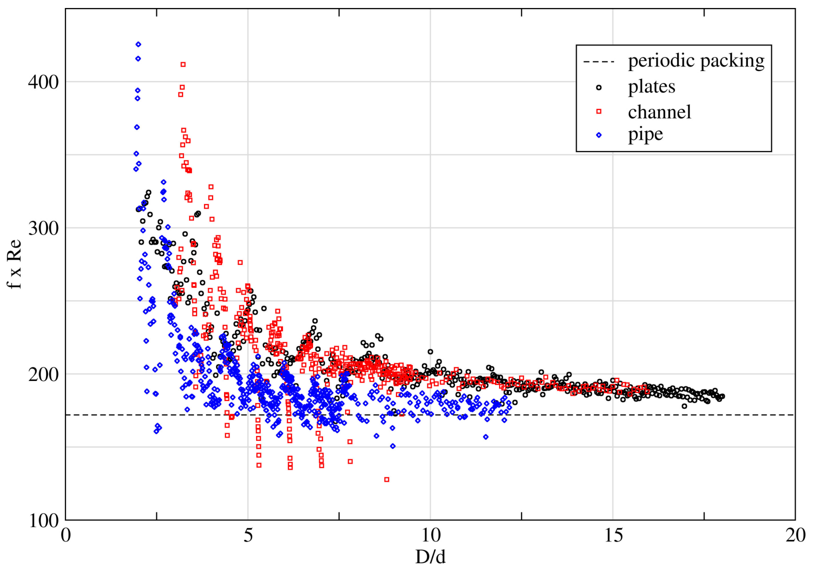


Figure 4.10: Dimensionless pressure drop as function of hydraulic diameter to sphere diameter ratio for all three geometries. Dashed line: dimensionless pressure drop of periodic packing (172)

4.5 Conclusions

In this chapter, the pressure drop in particle packed beds was calculated both for real packings, which were scanned using computer tomography as well as for computer generated packings.

It was shown that computational methods could be used to predict the pressure drop in scanned packings sufficiently well. Differences between measurement and simulation can be explained by limited scanning resolution and the fact that both beds were not identical, as well as measurement errors for the experiments. The drawback of these scanned packings is that only a few packings can be analyzed.

In contrast, numerical packing generation can be used for a fully automatic pipeline that can calculate the pressure drop for given geometric parameters as diameter to wall ratio. Using this method the wall effect on pressure drop in packed beds of spherical pellets was investigated and precisely quantified by calculation of the flow through systematically varied, synthetically generated packing configurations. In total, approximately 2 000 different packings of spheres were created and the flow through the packing was calculated. It was shown that the pressure drop is a non monotonic function of the wall distance to sphere diameter ratio especially for small ratios which is in contrast to common correlations. In general, the presence of walls was found to increase the pressure drop. For the pipe configuration, the results agree well with the correlation by Zhavoronkov et al. (1949). Furthermore, the concept of hydraulic radius to generalize the cross-section of the reactor was evaluated. Due to the geometry dependent changes of the packings for low wall distance to sphere diameter ratios, this generalization is only valid for high $\frac{D}{d}$ ratios. The oscillations of the pressure drop are caused by changes in the structure of the packings. For a more detailed analysis the influence of other quantities besides the porosity like tortuosity and pore size distribution should also be studied in the future work.

5 Transport Pores Optimization for Fischer-Tropsch Synthesis

This work was a collaboration with the Institute of Chemical and Electrochemical Process Engineering, where the one-dimensional model was developed, which is extended to three dimensions. The results of this chapter have also been published in the following research paper from which the figures and the respective captions were also taken:

- Bufe, A., Klee, M., Wehinger, G., Turek, T., Brenner, G. (2017). 3D Modeling of a Catalyst Layer with Transport Pores for Fischer-Tropsch Synthesis. *Chemie Ingenieur Technik.* 89; 1385-1390. Copyright Wiley-VCH GmbH. Reproduced with permission..

5.1 Introduction

The Fischer-Tropsch synthesis (FTS) is a process to convert carbon monoxide and hydrogen into a mixture of hydrocarbons that can be used as fuel or base compounds for the chemical industry. The reactants can be produced from sources like coal, natural gas and biomass by processes like steam reforming. For the production of alkanes, the reaction equation is as follows with $\nu_{\mathrm{H_2}}$ and ν_{Cn} being the stoichiometric coefficients for hydrogen and carbon monoxide:

$$\mathrm{CO} + \nu_{\mathrm{H_2}}\,\mathrm{H_2} \longrightarrow \nu_{\mathrm{C}n}\,\mathrm{C}_n\mathrm{H}_{n+2} + \mathrm{H_2O}$$

Historically, most FTS plants were built by states with large resources of coal in order to increase independence from oil imports like Nazi Germany and South Africa during the Apartheid. Nowadays, FTS is again of increasing interest as it can be used to generate carbon-neutral fuel from biomass that can be used for applications like aviation that require a high energy density fuel.

Usually, a porous catalyst is used for the process. As the pores are filled by the liquid product, the gaseous reactants have to dissolve in the liquid product and diffuse to the catalyst surface where they react. Not only is diffusion generally quite slow, but the molecules also have a longer path to travel through the porous catalyst that can be seen as a labyrinth. Due to the slow transport of the reactants, the possible height for the catalyst layer is limited. Furthermore, the concentrations of carbon monoxide and hydrogen

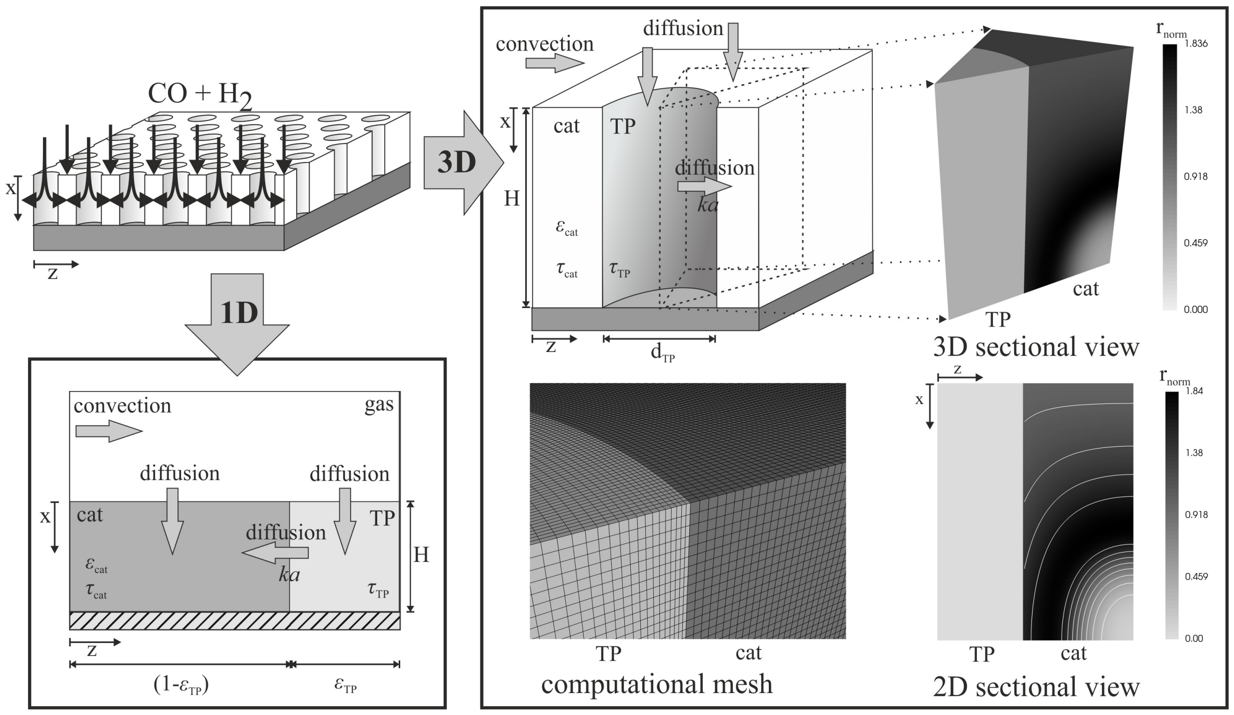


Figure 5.1: Outline on the derivation of the 1D and 3D diffusion and reaction model for a catalyst layer with transport pores, picturing concentration profiles and the computational mesh (TP: transport pore)

determine the reaction rate as well as the composition of the product. A method to improve the transport is the introduction of “transport pores”, cylindrical empty spaces in which diffusion is faster since the reactants can take the direct path.

Becker et al. (2014, 2016) presented a one dimensional model for a catalyst layer with transport pores which can be used to optimize catalyst layers. However, it is only valid for the limit of small transport pore diameters. To overcome this limitation, the model is extended to three dimensions here.

5.2 Modeling

At a gas-liquid boundary, the partial pressure p_i of a species i can be converted to a concentration c_i by using Henry’s law (eq. 5.1) where H is the substance dependent Henry constant and ν_L the molar volume of the liquid.

$$p_i = H_i \nu_L c_i \tag{5.1}$$

The transport pore porosity ϵ_{TP} is defined here as the ratio of the transport pore volume V_{TP} to the overall volume of the catalyst layer V_{Layer}:

$$\epsilon_{\mathrm{TP}} = \frac{V_{\mathrm{TP}}}{V_{\mathrm{Layer}}} \tag{5.2}$$

Fick's law is used for the formulation of a transport equation in the transport pores where no reaction occurs (eq. 5.3) and in the catalyst phase with reaction (eq. 5.4). The diffusion coefficient D_i depends on the species. Inside the catalyst the diffusion coefficient has to be modified by the porosity and the tortuosity τ, which describes by how much a path through the catalyst is longer than the direct path (eq. 5.5). The reaction rate is denoted by r and ν_i is the stoichiometric coefficient of species i.

$$\nabla \cdot D_i \nabla c_i(\vec{x}) = 0 \tag{5.3}$$

$$\nabla \cdot D_{i,\mathrm{eff.}} \nabla c_i(\vec{x}) - \nu_i(\vec{x}) r(\vec{x}) = 0 \tag{5.4}$$

$$D_{i,\mathrm{eff}} = \frac{\epsilon_{\mathrm{cat}}}{\tau_{\mathrm{cat}}} D_i \tag{5.5}$$

The reaction kinetics by Yates and Satterfield (1991) are used (eq. 5.6). As it can be seen, the reaction is slowed down by high concentrations of carbon monoxide. The partial pressure needed for the calculations can be obtained with Henry's law.

$$r = \rho_{\mathrm{cat}} \frac{a p_{\mathrm{H_2}} p_{\mathrm{CO}}}{(1 + b p_{\mathrm{CO}})^2} \tag{5.6}$$

The Anderson-Schulz-Flory distribution is applied to determine the stoichiometric coefficients for hydrogen (5.8) and the formed hydrocarbons (5.8). It uses a chain growth probability α (5.9) to describe of which length the formed hydrocarbons are.

$$\nu_{\mathrm{C}n} = \alpha^{n-1}(1-\alpha)^2 \tag{5.7}$$

$$\nu_{\mathrm{H_2}} = \alpha - 3 \tag{5.8}$$

$$\alpha = \frac{1}{1 + k_{\mathrm{a}} \left(\frac{c_{\mathrm{H_2}}}{c_{\mathrm{CO}}}\right)^{\beta}} \tag{5.9}$$

In order to unite both transport equations 5.3 and 5.4 into a single one, a marker function δ is introduced that differentiates between transport pore and catalyst:

$$\delta(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \text{ in catalyst phase} \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

With the help of this function, the transport for a species i can then be described with a single equation (eq. 5.11). Note that this equation only has to be solved for carbon monoxide and hydrogen as reaction rate and chain growth probability only depend of the concentration of these two compounds.

$$\nabla \cdot \left\{ \left[\delta(\vec{x}) \frac{\epsilon_{\text{cat}}}{\tau_{\text{cat}}} + (1 - \delta(\vec{x})) \right] D_i \nabla c_i(\vec{x}) \right\} + \delta(\vec{x}) \nu_i(\vec{x}) r(\vec{x}) = 0 \tag{5.11}$$

For the evaluation, the selectivity of the undesired methane (eq. 5.12) and the desired longer chained hydrocarbons (eq. 5.13) is calculated.

$$S_{\text{CH}_4} = (1 - \alpha)^2 \tag{5.12}$$

$$S_{\text{C}_{5+}} = 1 - \sum_{n=1}^{4} n \alpha^{n-1} (1 - \alpha)^2 \tag{5.13}$$

Finally, the area time yield (ATY), the amount of created hydrocarbons with a chain length greater than 5 per time and area is used as an integral evaluation parameter for the catalyst layer:

$$\text{ATY} = \frac{1}{A_{\text{Surface}}} \int_{V_{\text{cat}}} r(\vec{x}) S_{\text{C}_{5+}}(\vec{x}) \mathrm{d}\vec{x} \tag{5.14}$$

5.3 Numerical Setup

OpenFOAM (version 4.1) is used to discretize the model using the finite volume method and to solve the resulting system of equations. The linear contributions of the concentrations to eq. 5.6 are implicitly discretized, whereas the non-linear terms are treated explicitly. Due to the non-linear terms in the reaction rate and the stoichiometry, an iterative solution scheme has to be utilized. Instead of alternatingly solving the transport equations for both species, for each species the equation is first solved multiple times until the residuum is reduced by a factor of ten before switching to the other species as this was found to improve convergence. The calculations are terminated if both residua in one iteration are below $1 \cdot 10^{-7}$. After convergence, the ATY is calculated according to equation 5.14.

Parameter	Value		Source
p	21	bar	assumed
ϵ_{cat}	0.4		assumed
τ_{cat}	3		assumed
ρ_{cat}	1000	$\mathrm{kg\,m^{-3}}$	assumed
a	0.088 853 3	$\mathrm{mol\,kg^{-1}\,s^{-1}\,bar^{-2}}$	Maretto and Krishna (1999)
b	2.226	$\mathrm{bar^{-1}}$	Maretto and Krishna (1999)
β	1.76		Vervloet et al. (2012)
k_{α}	0.0567		Vervloet et al. (2012)
D_{CO}	$14.30 \cdot 10^{-9}$	$\mathrm{m^2\,s^{-1}}$	Erkey et al. (1990)
D_{H_2}	$36.05 \cdot 10^{-9}$	$\mathrm{m^2\,s^{-1}}$	Erkey et al. (1990)
H_{CO}	363.8	bar	Marano and Holder (1997)
H_{H_2}	456.6	bar	Marano and Holder (1997)
ν_{L}	0.5818	$\mathrm{m^3\,mol^{-1}}$	Erkey et al. (1990)

Table 5.1: Simulation parameters and their sources

Hexaedral, structured meshes that utilize the symmetry were created with the OpenFOAM tool blockMesh for the calculation. The meshes were created so that the interface between transport pore and catalyst fell on a cell boundary. As there is no reaction inside the transport pores, the changes in concentration are lower there and therefore the mesh was chosen to be less dense there with approximately twice the cell length as in the catalyst. A grid convergence study was conducted in order to assure sufficiently fine grids. An exemplary grid is shown in figure 5.1.

On top of the mesh, a fixed value for the concentrations was set and a zero-gradient condition was applied to the bottom. The other sides of the mesh are symmetry planes. The parameters used for the calculations can be found in table 5.3.

5.4 Results

5.4.1 Verification

For verification, the concentration, reaction rate and specificity profiles were calculated for a plain catalyst layer without any transport pores. As the concentrations orthogonal to the catalyst layer are assumed constant, the profiles are solely dependent on the height coordinate and therefore should be identical to those calculated with the one dimensional model by Becker et al. (2016).

Figure 5.2 compares the profiles for the one dimensional and the three dimensional model. As it can be seen, there is basically no deviation between the two models, proofing correct

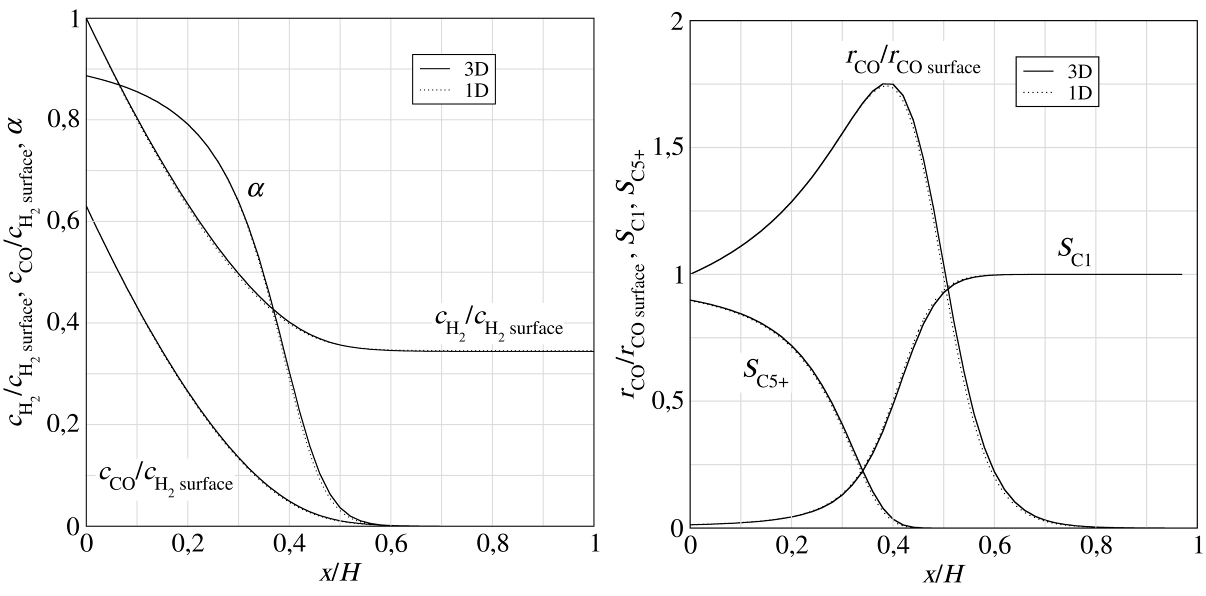


Figure 5.2: Comparison of results of 1D (dashed lines) and 3D model (solid lines) for a 300 µm thick dense layer. a) Concentrations of H_2 and CO, normalized by surface concentration of H_2 and chain growth probability α. b) Reaction rate of CO, normalized by surface reaction rate, selectivity of methane and higher (C_{5+}) hydrogen carbons.

implementation. The results also show the problems of a plain catalyst layer without transport pores. As the diffusion of carbon monoxide is slower than that of hydrogen, its concentration decreases quite fast. Although this leads to a peak in reaction rate at 40 % of the height, the selectivity for the desired products is almost zero. Furthermore, the last 20 % of the height is basically wasted as there is no reaction due to the lack of carbon monoxide.

5.4.2 Results for Constant Catalyst Layer Thickness

Next, the influence of the porosity and transport pore diameter on the ATY is investigated for a layer thickness of 300 µm. Although it might seem that porosity and transport pore diameter are coupled, they can be varied independently by changing the distance between transport pores, i. e. the total number of transport pores. The results are plotted in figure 5.3. As it was to be expected, the results match the results by the one dimensional model for the smallest used transport pores of 10 µm as the one dimensional model can be seen as the limit of very small pore size. The profiles show a peak as for large transport pore porosity only more and more catalyst is removed. With higher transport pore size the achievable gain in ATY decreases and is shifted to higher transport pore porosities. This can be explained by the fact that for a fixed porosity but larger transport pore size, the

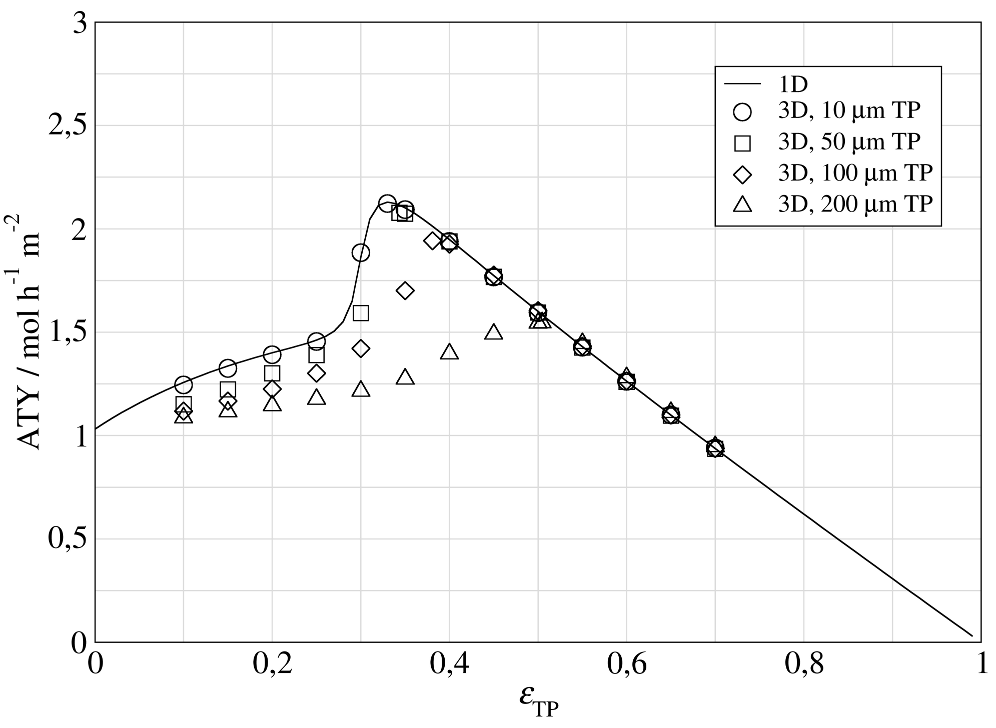


Figure 5.3: Areal time yield (ATY) as function of transport pore porosity for the 1D and the 3D model with different transport pore diameters. Note that the 1D model has also transport pores (of infinitesimal diameter).

distance between the transport pores is higher and therefore the catalyst layer will contain more inactive zones.

Figure 5.4 shows 2D sectional views for a transport pore porosity of 0.3 and a transport pore diameter of 100 µm. It can be clearly seen that the quantities cannot be treated as a function of height alone anymore. Similar to the case of a dense layer, a "reaction front" with high reaction rates can be seen. Due to the transport pore, this front is bent to a curve and stopped at the bottom of the catalyst suggesting that the lower ATY of such high transport pore diameters can be compensated by a greater catalyst layer height which will be investigated next.

5.4.3 Algorithmic Optimization for Maximum ATY

For the purpose of designing a chemical reactor, catalyst layer height and transport pore porosity have to be optimized to achieve maximum ATY for a given transport pore diameter. The transport pore diameter is not a parameter of optimization as smaller transport pores will always lead to high ATY and the transport pore diameter cannot be chosen freely because it is limited by manufacturing capabilities.

Optimization with the one dimensional model is somewhat more straightforward as the

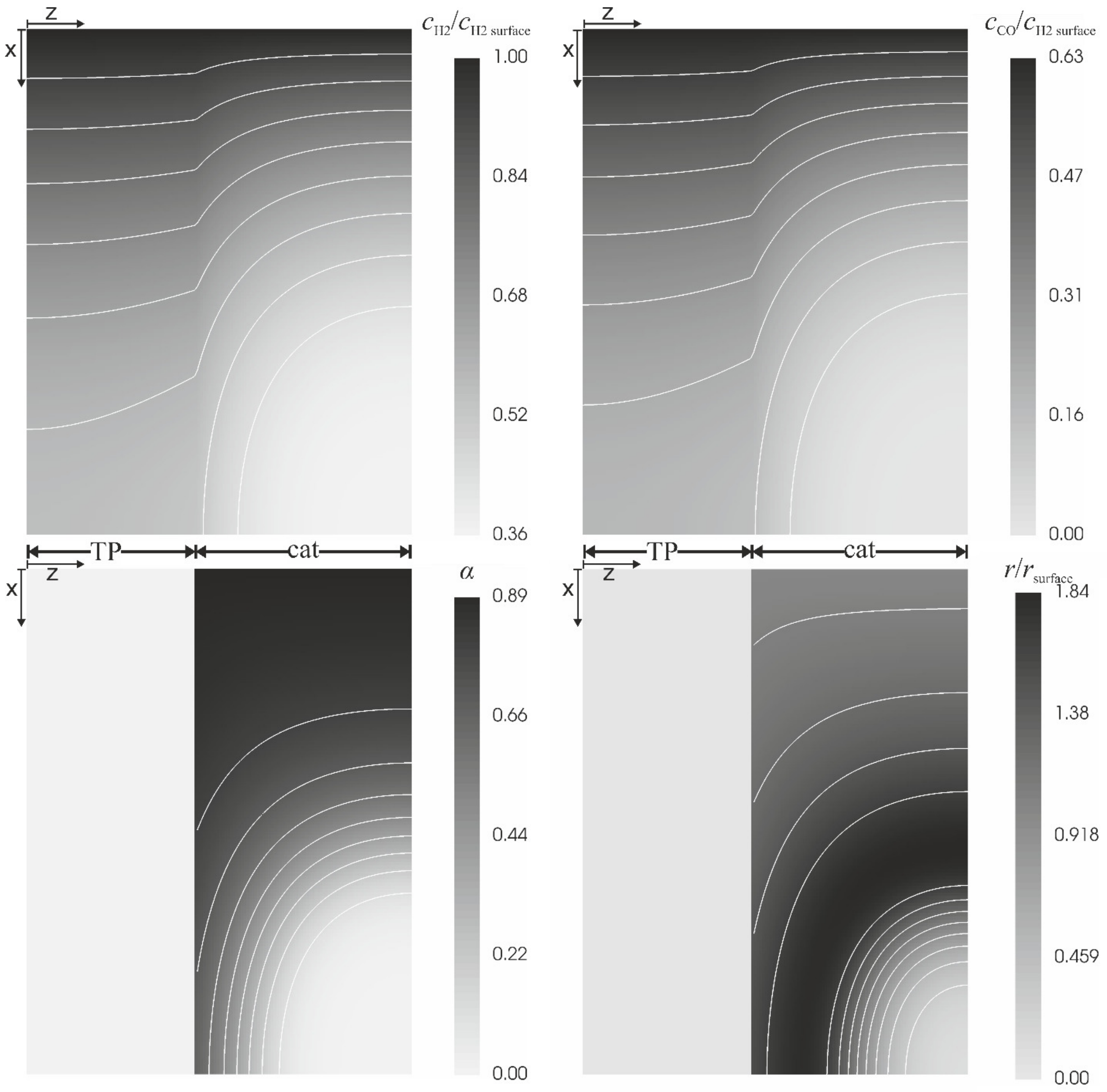


Figure 5.4: 2D sectional view of transport pores and catalyst. Transport pore porosity is 0.3, transport pore diameter 100 µm and the layer thickness 300 µm. a) concentration profile of H_2, b) concentration profile of CO, c) chain growth probability α, and d) reaction rate of CO. Concentrations are normalized by external surface concentration ($x = 0$) of H_2 and the reaction rate by the surface reaction rate. The white lines are isolines.

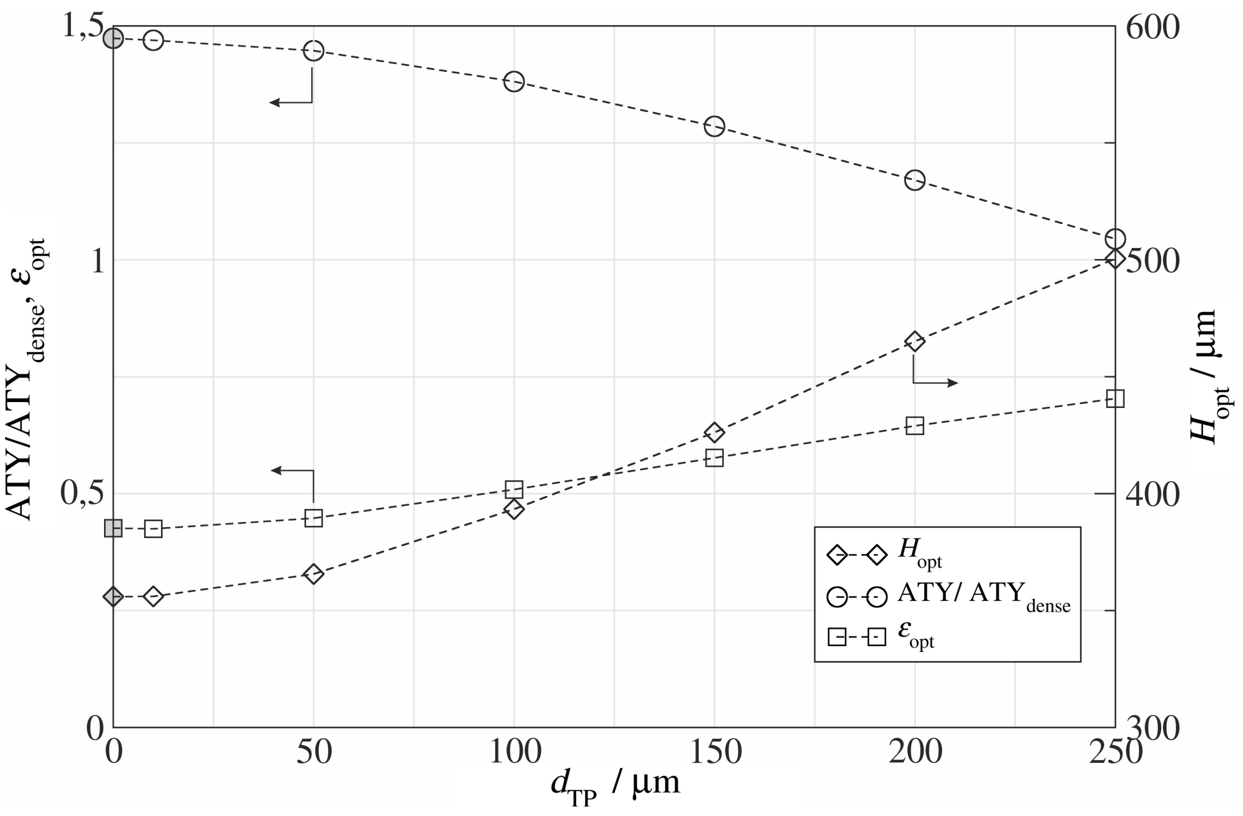


Figure 5.5: Achievable ATY, optimal transport pore porosity $\epsilon_{TP,opt}$ (both left axis) and optimal layer thickness H_{opt} (right axis) as function of transport pore diameter. The gray marked symbols are the results of the 1D model.

model consists only of an ordinary differential equation that has to be solved, whereas the three dimensional model uses the finite-volume method and requires creation of a grid. As structured grids are used, the creation of those is reliable. Furthermore, the solver was found to be robust and did not require any "tweaking" of numerical parameters. The whole process of grid creation and calculation can thus be automatized.

The optimization is handled as a two-stage optimization where the ATY for a catalyst layer height with optimal transport pore diameter is optimized in the outer stage and for each optimization step another optimization of the transport pore diameter is performed. The algorithm by Brent (2002) for finding the minimum or the maximum of an arbitrary function in the implementation of the GNU Scientific Library (version 1.11) was used as it is a fast and robust method that does not require the knowledge of derivatives.

The results are shown in figure 5.5. For small transport pore diameters, a gain in ATY of 50 % compared to that of an optimal dense layer ($1.5\,\mathrm{mol\,h^{-1}\,m^{-2}}$) can be achieved. For pores larger than 50 µm the gain starts to reduce. This in agreement with the results by Johannessen et al. (2007) who developed a theoretical criterion of whether or not the gradients orthogonal to the layer can be neglected. The results also show that for higher transport pore diameters the loss in ATY can partly be compensated by employing a higher catalyst layer and an increased transport pore porosity. Transport pores, however, have to be smaller than 250 µm to obtain a noticeable gain.

5.5 Conclusions

A three dimensional model for diffusion and reaction inside a catalyst layer with transport pores for FTS was presented. In contrast to the one dimensional model by Becker et al. (2014, 2016) on which it was based, it is also valid for large transport pores. Larger transport pores than 50 µm generally reduce the achievable gain of ATY which can somewhat be compensated by thicker catalyst layers and more transport pores.

Another advantage of the three dimensional model is that the influence of the transport pore shape can also be studied as it can handle arbitrarily shaped transport pores which might be of interest as the manufacturing of perfectly cylindrical shaped transport pores is difficult. The drawback however is that it requires the generation of a computational mesh and is generally more computationally expensive.

Although – strictly speaking – no "flow parameters" were calculated as only diffusion was considered, this example also shows the advantage of an automatic processing because it allowed using algorithmic optimization. Since the creation of the used structured meshes is straightforward, the finite-volume based OpenFOAM was used here instead of the LBM solver.

6 Nasal Cavity Flow

This work was done in cooperation with Nataša and Aleksa Janović, University of Belgrade, who created the database of CT scans as well helped with medical questions.

6.1 Introduction

The nasal septum is the part of the nose that divides the nasal cavity into a left and a right airway. Nasal septum deviation (NSD) means the displacement of this septum (figure 6.1) and *can* lead to symptoms like facial pain, headaches and sleep apnea (WebMD, 2020). However, although the prevalence of NSD is very high with up to 92.7 % in general population (Mladina et al., 2008; Janovic et al., 2020), many do not have any symptoms and are not even aware of their deviated septum. It is possible to surgically correct the nasal septum, but since the connection between clinical findings and symptoms is unclear, it is challenging to decide whether a patient will actually profit from surgery.

Many attempts have been made to connect patient's symptoms to objective quantities like the pressure drop of the nose. There have also been numerous studies - the review paper by Quadrio et al. (2013) alone lists 19 different articles - of the flow changes caused by NSD with CFD. For a traditional CFD analysis of the flow multiple steps like surface extraction, mesh generation and evaluation have to be performed manually leading to a high amount of work per case and thus limiting the number of analyzable cases. This is problematic as analyzing only a few cases limits the usefulness of a study. There are many different types of NSD and therefore one has to analyze a high number of cases and use statistical methods to find correlations for sound results. For these reasons, the focus here is not on studying a few cases in detail, but on providing a method that can determine fluid dynamic parameters as pressure drop and the distribution of the flow between the two sides in a fully automated fashion. Such an approach has two advantages: First, it can be used to extract fluid dynamic parameters from a complete database of CT scans. Second, it could be used as an analysis tool for surgery planning as it does not require deep knowledge in CFD and therefore could be used in medical practice.

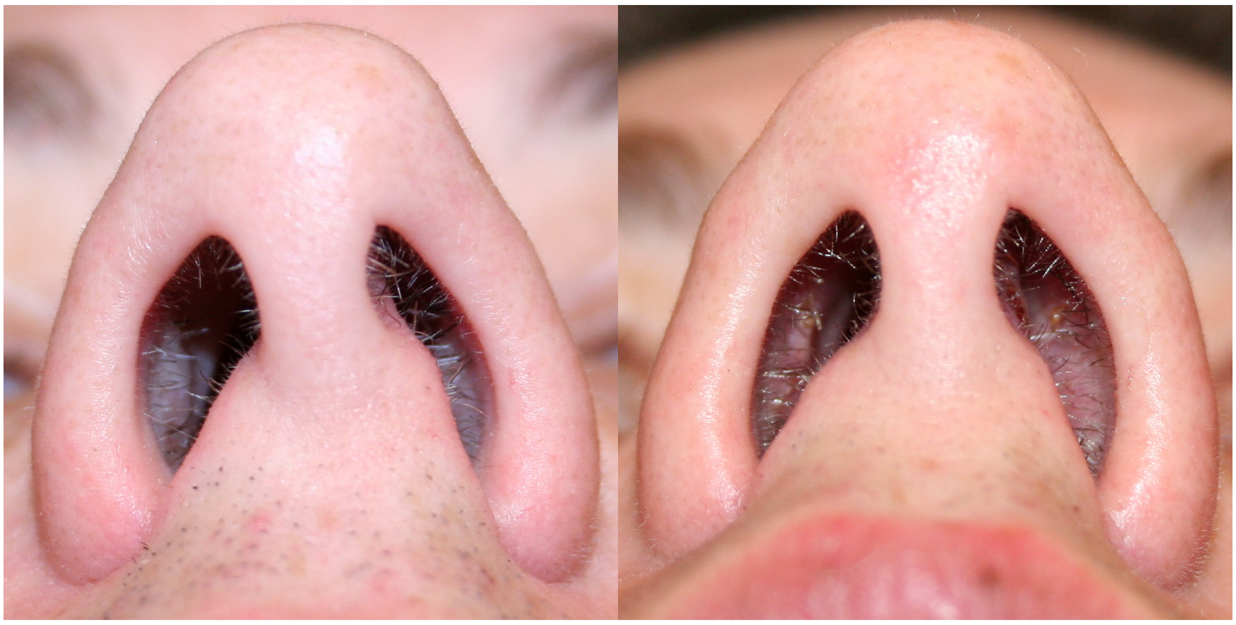


Figure 6.1: Example of a person with NSD before (left) and after (right) surgery. The pictures are based on photographs by "Jeff and Mandy G" licensed under CC BY-SA 2.0.

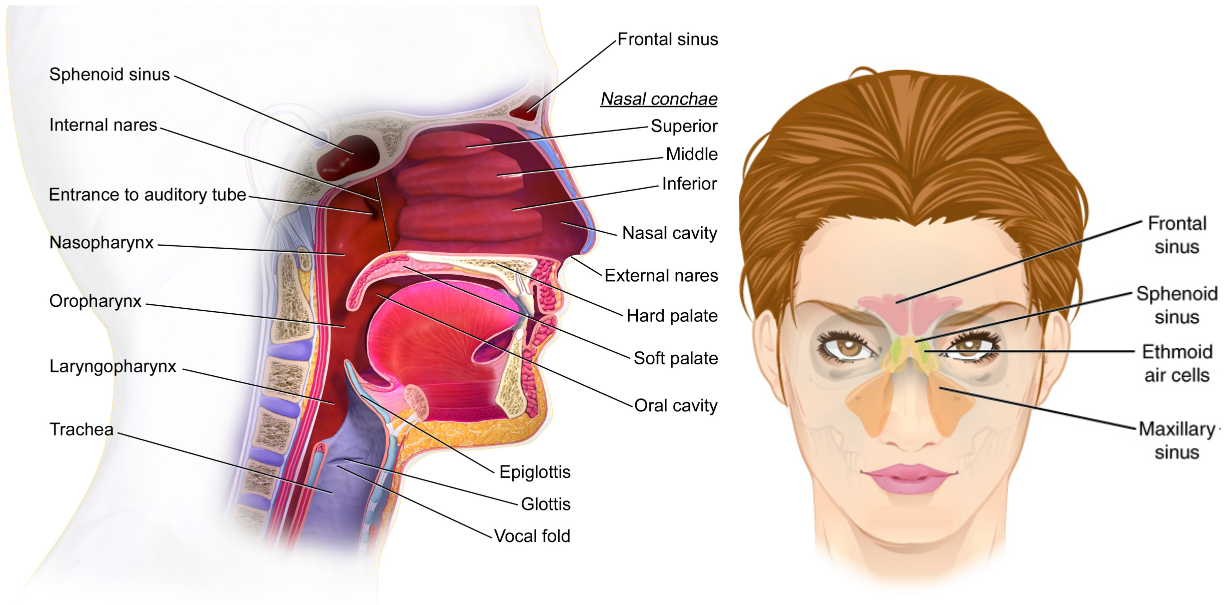


Figure 6.2: Illustration of the upper respiratory system and the paranasal sinuses (Source left: Blausen.com staff (2014). "Medical gallery of Blausen Medical 2014". WikiJournal of Medicine 1 (2). DOI:10.15347/wjm/2014.010; Source right: Image by OpenStax licensed under CC BY 4.0)

6.2 Background

Figure 6.2 shows a sketch of the upper respiratory tract. The air enters the nose through the nostrils into the nasal cavity, which is divided by the nasal septum into two separate airways. At the end, both airways unite through the choanae into the nasopharynx.

The nasal cavity is surrounded by a group of air-filled spaces, the paranasal sinuses, which are connected to it. The paranasal sinuses can be distinguished as maxillary, frontal, ethmoidal and sphenoidal sinuses. Being dead ends, they are unimportant for breathing. Their exact purpose is unknown (Standring, 2016).

The rate of breathing, the respiratory rate, for adults at rest is usually around 12-16 per minute and the minute ventilation between 5 and 8 liters per minute. The distribution of the airflow between the two sides of the nasal cavity is usually not symmetrical as the sides partially congest and decongest over time, which is called the nasal cycle.

Computed tomography can be used to generate 3D imagery by measuring x-rays attenuation from different directions and using a reconstruction algorithm. The resulting imagery can be seen as a three-dimensional regular grid with an attenuation value given for each of its voxels. The attenuation is usually given in the Hounsfield scale, which is based on the attenuation of water ($\mathrm{HU} = 0$) and air ($\mathrm{HU} = -1000$).

In order to quantify the severity of patient's symptoms, the Nasal Obstruction Symptom Evaluation scale (NOSE) was developed. It is a survey with five questions evaluating the severity of different conditions, which is used to calculate a scoring in the range of 0-100. To classify different types of nasal septum deviation, multiple schemes have been proposed (Teixeira et al., 2015).

6.3 Automatic Processing

There are multiple steps necessary that all have to be performed in a fully automated fashion for calculation of fluid parameters. First, the CT-scan is preprocessed, meaning artifacts have to be removed and the scan has to be trimmed to the area of interest. Then, a triangle mesh describing the surface is extracted from the scan. This triangle mesh is used as input for the meshing tool to generate the computational meshes for CFD. Boundary conditions have to be set for the computational mesh and the flow is then calculated. After the calculation, the flow has to be evaluated. In the following sections each step of our pipeline will be described.

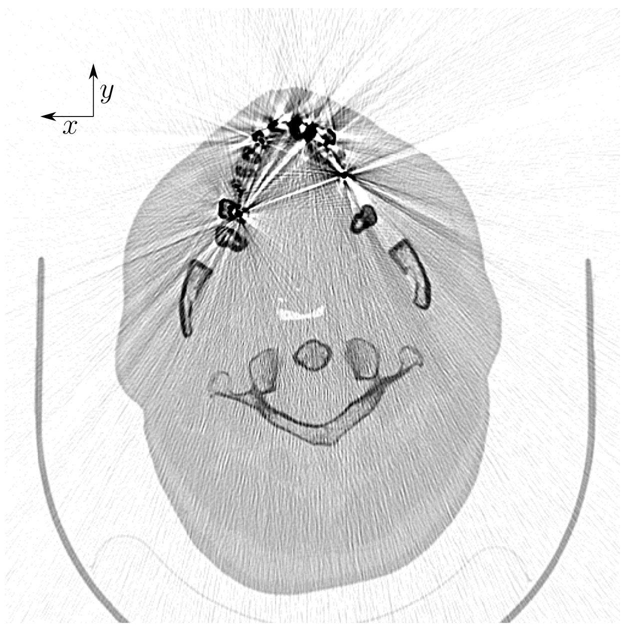


Figure 6.3: Example for artifacts caused by dental inlays. Furthermore, the headholder used during the examination is visible.

6.3.1 Preprocessing of the CT Scans

The scans can be seen as a three-dimensional grid of uniform cells with the radiodensity given in the cell center. A simple threshold is used to decide whether a cell is an empty, fluid or a solid, wall cell. The axes are defined as follows: The patient is looking in y-direction; The z-axis is upwards from patients view and the x-axis pointing to the left. Since the patient's head will not always be at the same position in the scans, the nose tip is used as a point of reference. Presuming that there are no artifacts in the scan, the nose tip can be determined by taking the wall cell with the maximum y-value.

Artifact Removal

The first step is to remove artifacts from the imaging process. Parts of the CT scanner can appear on the scan. Dental inlays or crowns can cause further artifacts. Examples are given in figure 6.3. In order to remove such artifacts, the geometry is binarized to solid and fluid voxels. Then, connected component labeling is performed and the largest connected component determined. Although there are specialized algorithms available for connected component labeling (Ohira, 2017), a BFS is used here as a simple and easy to implement approach as this step is not time critical. After the connected component analysis, only solid voxels which are part of the largest connected component are kept.

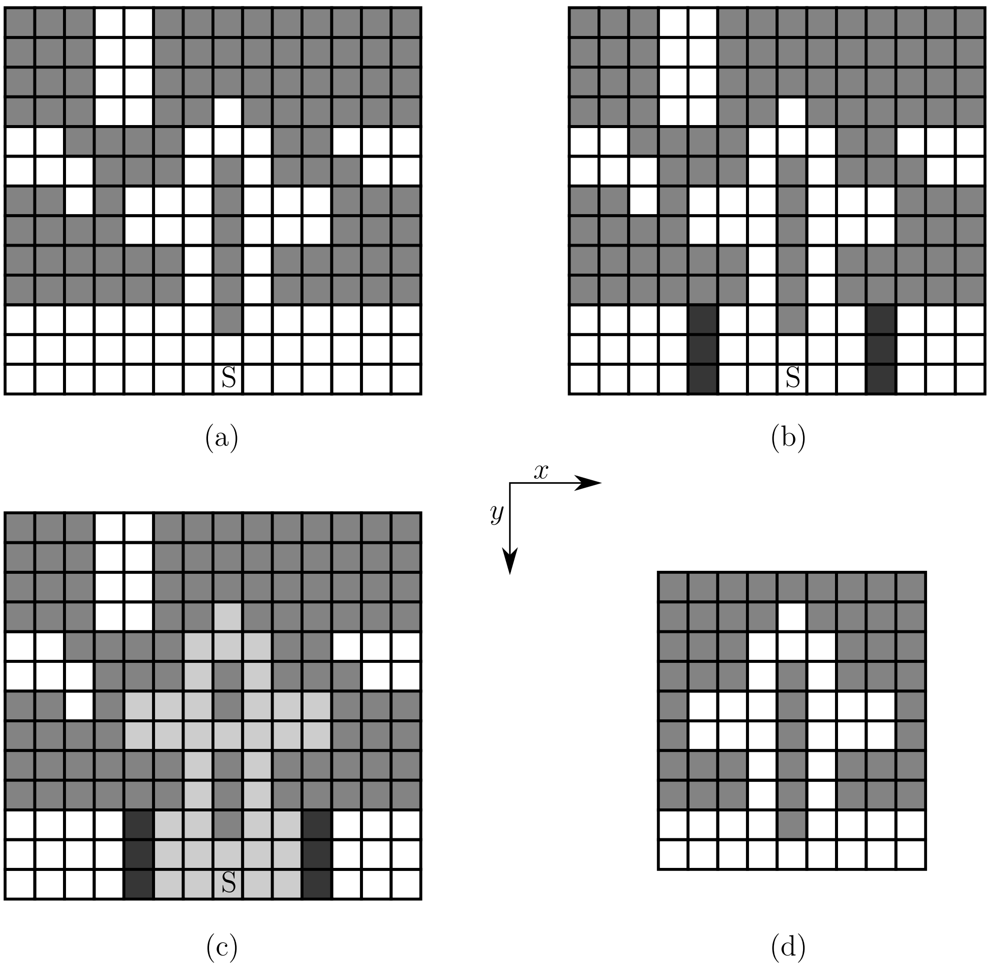


Figure 6.4: Illustration of the removal of unconnected cavities: (a) start point (S) in front of nose tip marked; (b) barriers added; (c) all cells connected to start point marked; (d) trimmed scan

Removal of Unconnected Cavities and Trimming

The scan is usually larger than needed and thus should be trimmed in order to save processing time. Furthermore, it may contain cavities that have no connection to the respiratory system. These would also appear in the meshes for CFD calculation and increase the computational time as well as complicate the setting of boundary conditions. In order to determine the cells of the scan that need to be kept, a "molding" of the airways is created: First, a barrier around the nose that limits the region outside the head is set. This barrier is an axis-aligned channel around the nose tip with open ends in y-direction. It starts from sufficiently far away in y-direction and ends on the first contact with the head. The x- and z-extent of the channel are chosen based on heuristical values. Then, all fluid cells that are connected to a fluid cell in front of the nose tip are marked by using a breadth-first search.

Fluid cells of the scan which are not marked are set to wall cells. This step is similar to the CPU based meshing (section 3.4.2). Furthermore, the scan is trimmed to an axis aligned bounding box around the marked cells. This procedure is sketched in figure 6.4.

Closure of Clipped Paranasal Cavities

In some CT scans, the extension of the scans in z-direction is not sufficient so that frontal or maxillary sinuses are not fully included, leading to additional openings in the later generated triangle mesh. These additional openings are marked red in Figure 6.5. In order to detect and remove these openings, the xy-slices with highest and lowest z-values are taken and a connected component analysis for each of the two slices is performed. The slice with high z-values should only have one component, i. e. the region in front of the head. In the case of multiple components, a frontal sinus is clipped and only the component with the highest y-values is kept. The other components are set to wall nodes. Similarly, the slice with the lowest z-values should have two components: one outside the head and the other being the nasopharynx. All other components are again closed and the positions of the two components are saved as they are needed to differentiate between outlet and inlet.

If the distance between the nose tip and the lowest most slice is less than 2 cm, the scan is artificially enlarged by copies of this slice translated downwards. This is necessary as otherwise there might be very little space under the nostrils, which could artificially disturb the airflow.

Surface Extraction

As mesh generators for CFD software usually require a triangle mesh as input, the marching cubes algorithm (Lorensen and Cline, 1987) for extracting an isosurface is used. After the extraction, a smoother is applied to remove sharp edges. Since this might influence the results, the effect of the smoother as well as the influence of the threshold used for extraction on the calculated results will be evaluated later (sections 6.4.1 and 6.4.2). The surface extraction step is implemented using the VTK library (Schroeder et al., 2006).

6.3.2 Mesh Generation

The meshes for OpenFOAM are created using the OpenFOAM tools blockMesh and snappyHexMesh, whereas the meshes for the LBM calculations are generated with an in-house meshing tool which is based on recursive refinement of an octree (see section 3.4.3). In order to decrease the number of cells outside the nose, a box around the nose tip is used to limit the mesh outside the head. For OpenFOAM, a base mesh with cells having a

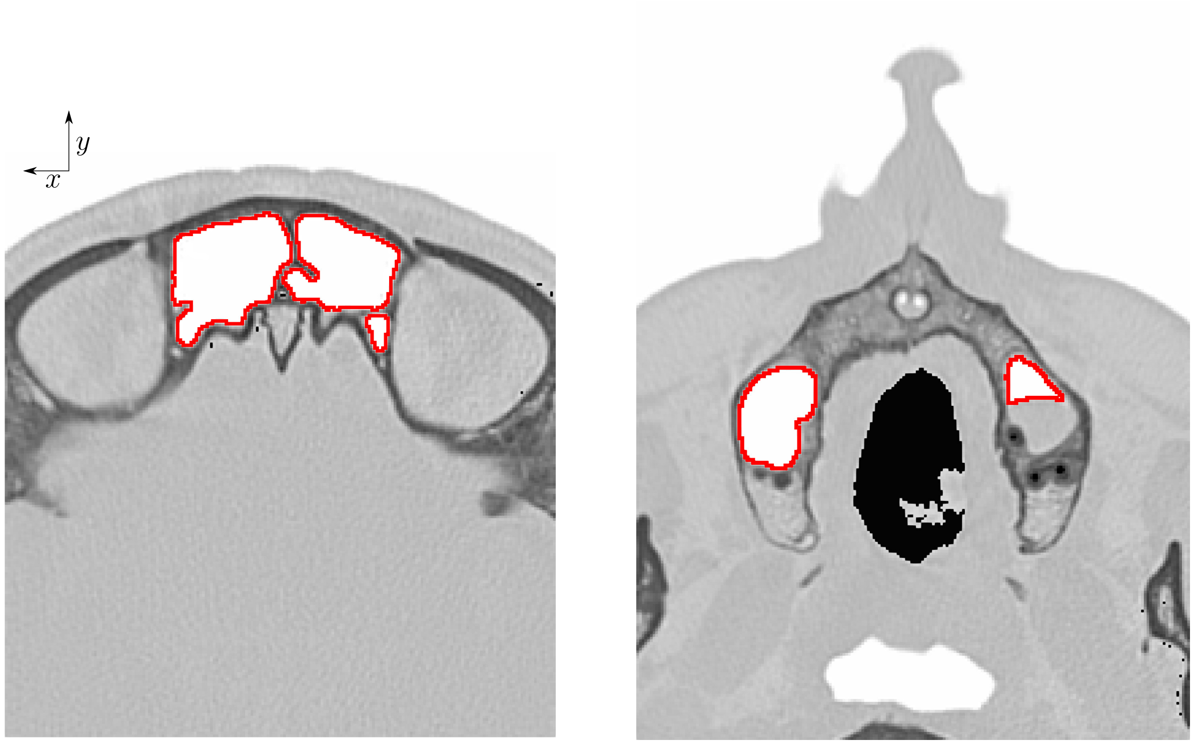


Figure 6.5: Example of a scan with insufficient extent in z-direction. In the top most slice (left) the frontal sinus is clipped and in the bottom most slice (right) the maxillary sinus. Blacked areas are removed unconnected cavities.

side length of ca. 0.8 mm and three levels, so that the finest cells have an approximate side length of 0.2 mm is generated. The meshes for the LBM consist of two levels and use a cell length of 0.25 mm on the coarse, respectively 0.125 mm on the fine level.

6.3.3 Flow Calculation

The inlet is the XZ-plane in front of the head for which a constant flow rate of $7.5\,\frac{\mathrm{L}}{\mathrm{min}}$ is set. The outlet is in the nasopharynx and a relative pressure of 0 Pa is applied there. For both solvers, no turbulence modeling is used as Hörschler et al. (2006) showed that the flow is most likely laminar for such flow rates.

The OpenFOAM calculations are performed using the steady state incompressible solver simpleFoam. The termination criterion on the residual for the pressure is set to $5 \cdot 10^{-5}$ and that for the velocity to $1 \cdot 10^{-5}$. The relaxation factors are set to 0.75 for the velocity field and 0.25 for the pressure field. Each case is run on a single compute node consisting of two Intel Skylake 6148 CPUs and 186 GB of RAM.

The LBM calculations are run with *lbm4e*. For better stability, the cumulant collision operator (Geier et al., 2015) were used. Interpolation between different grid levels is performed with the compact interpolation scheme (Geier et al., 2009). The linear interpolated bounce back (Bouzidi et al., 2001) is used for the curved walls and Guo (Guo et al., 2002) boundary conditions for the inlet where a constant velocity is set as well as the outlet where a constant pressure is set. Calculations are run for 25 000 coarse-grid time steps. Each case is run on a single AMD Radeon VII graphics card with 16 GB of device memory.

6.3.4 Automatic Evaluation

Although there are multiple possibilities of integral values for evaluation like maximum velocities in each of the two airways separated by the nasal septum, wall shear stress etc., only pressure drop and flow partitioning are considered here for the sake of simplicity.

The pressure drop can be easily determined by taking the difference between the pressures of the inlet and the outlet.

Determining the flow partition i. e. the percentage of the flow rate through one airway relative to the total flow rate requires information on whether a cell belongs to the right or left side. This can be determined using the following algorithm, which is illustrated in figure 6.6:

First, a starting point left of the nose tip is selected (the red cell in part (a)). Then, the shortest distance to every other cell is determined (part (b)). The distance between two neighboring cells is generally defined as the distance between their cell centers. Dijkstra's algorithm can be used to efficiently calculate the shortest paths. In the illustration, all neighbored cells are considered to have a distance of one for the sake of simplicity. The same is repeated for a point slightly right to the nose tip (part (c)). Last, for each cell the distances are compared. If a cell is closer to the starting point left of the nose tip, it belongs to the left side and to the right side otherwise (part (d)). An example of a final partitioning is shown in figure 6.7.

The flow rate through each airway can then be determined by taking a xz-slice through the respiratory tract at an arbitrary position and integrating the velocity for both sides separately over the obtained cross-section. Due to the mass conservation, the exact position of the slice does not matter.

6.4 Evaluation

6.4.1 Influence of the Threshold Value

The threshold value chosen for surface extraction (section 6.3.1) has a potential impact on the obtained results and is therefore studied before choosing a final value. Figure 6.8 shows

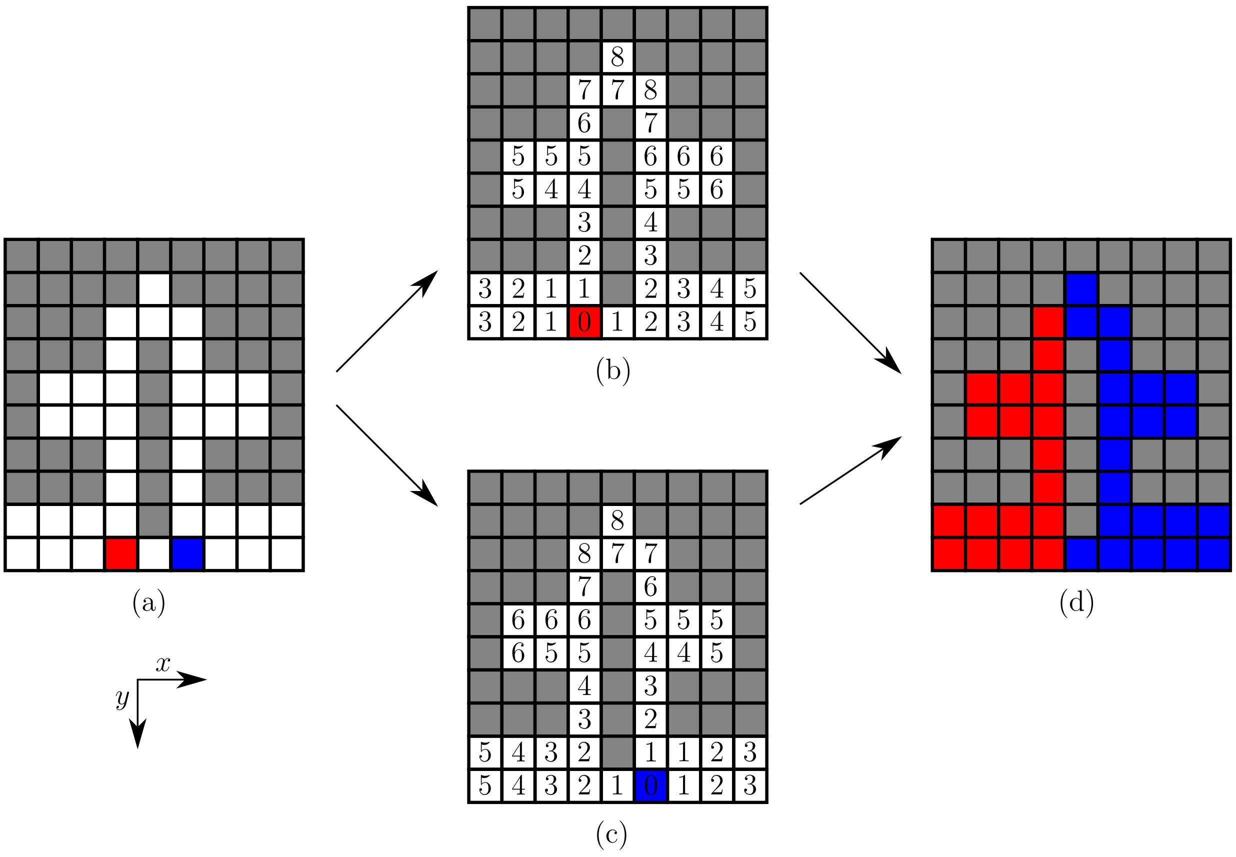


Figure 6.6: Illustration of the partitioning algorithm: (a) Start positions left and right of the nose tip; (b) distances to the left start point; (c) distances to the right start point; (d) cells marked as left and right

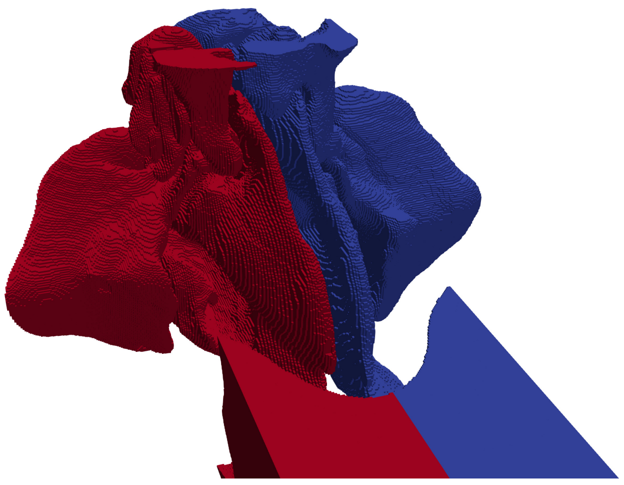


Figure 6.7: Mesh used for LBM calculations with the identification of the left and right upper respiratory tract

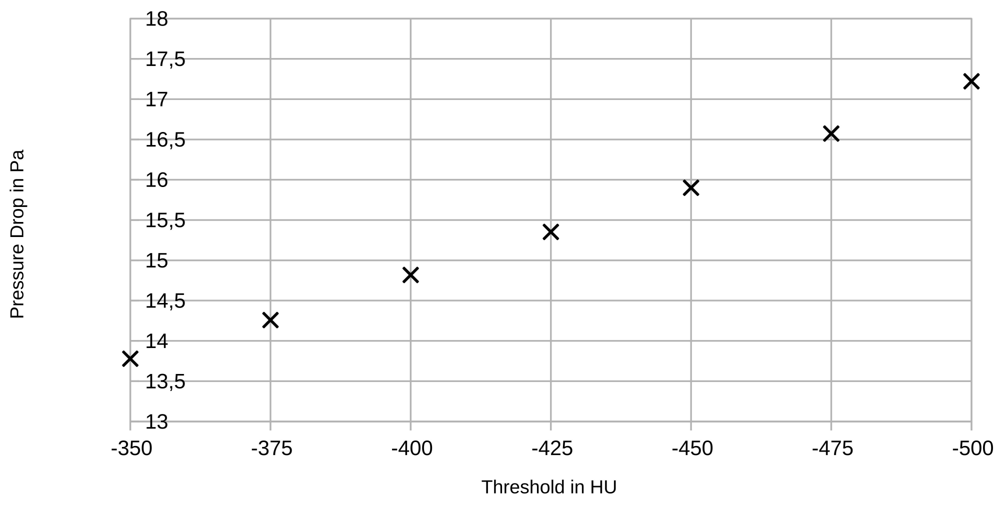


Figure 6.8: Pressure drop as a function of the threshold value used for isosurface extraction

the influence of the chosen threshold value on the calculated pressure drop. As it can be seen, the pressure drop changes almost linearly with the threshold parameter. This is in accordance with the results by Zwicker et al. (2018) and can be explained by the linear interpolation used in the marching cubes algorithm for the surface extraction. The small gaps which probably have a strong influence on the pressure drop are resolved only by a few voxels in the CT-scan. Thus, the width of those gaps is highly dependent on the threshold parameter. In the following, a threshold of -450 HU is used based on the results by Zwicker et al. (2018).

6.4.2 Influence of Smoothing

The effect of smoothing on the results is also investigated. Smoothing can help to remove artifacts in the scan but also has the risk of removing real features. Since the results (figure 6.9) show that the influence is minimal, a moderate smoothing of 10 iterations is applied.

6.4.3 General Success Rate

A selection of 104 scans is used to evaluate the automatic method. 10 scans were sorted out after preprocessing, as they contained one or multiple artifacts that made flow calculation impossible. Of these 10 scans, the most common problem was that the nasopharynx was closed due to the patient swallowing during examination. In two cases, the scanned area was too small. In one case, the patient's head was rotated so that the nose tip was not the point with the maximum y-coordinate. It might seem that these problems question the

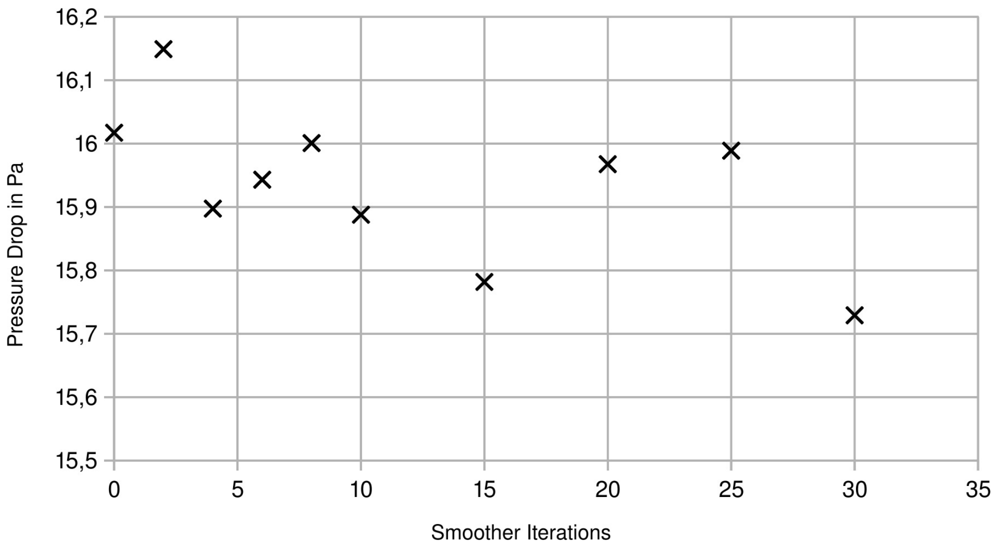


Figure 6.9: Pressure drop as a function of the smoother iterations used

generality of the presented method, but it should be kept in mind that the database used was never intended for this application and some problems could have been solved during the scanning process or by post-processing the scans.

For the remaining 94 scans, OpenFOAM mesh creation failed for two cases. In 15 cases snappyHexMesh completed with errors that the targeted mesh quality was not achieved. These errors were ignored as in most cases the solver converged regardless. In two cases no usable mesh for calculation could be achieved. Of the 92 scans which were used for the calculations the solver failed in one case to converge within the given time limit.

As it was expected, mesh generation for LBM was successful in all cases. Furthermore, no numerical instabilities were encountered when calculating the flow so that calculation of the flow for all 94 cases was possible.

Detailed analysis of the cases with a very high pressure drop revealed that in those cases the nasopharynx was partially closed by artifacts that were unfiltered and undetected during preprocessing. Furthermore, in some cases the scanning resolution was too coarse to resolve the nasal septum leading to connections between the left and right airway which might influence the results. Figure 6.10 shows an example of this.

6.4.4 Result Comparison

An overview of the results for the pressure drop is depicted in figure 6.11 and for the flow rate partitioning in figure 6.12. Some cases show a very high pressure drop of over 15 Pa, which was caused by undetected artifacts in the nasopharynx. The results for the flow partitioning are almost identical for OpenFOAM and *lbm4e* with a maximum deviation

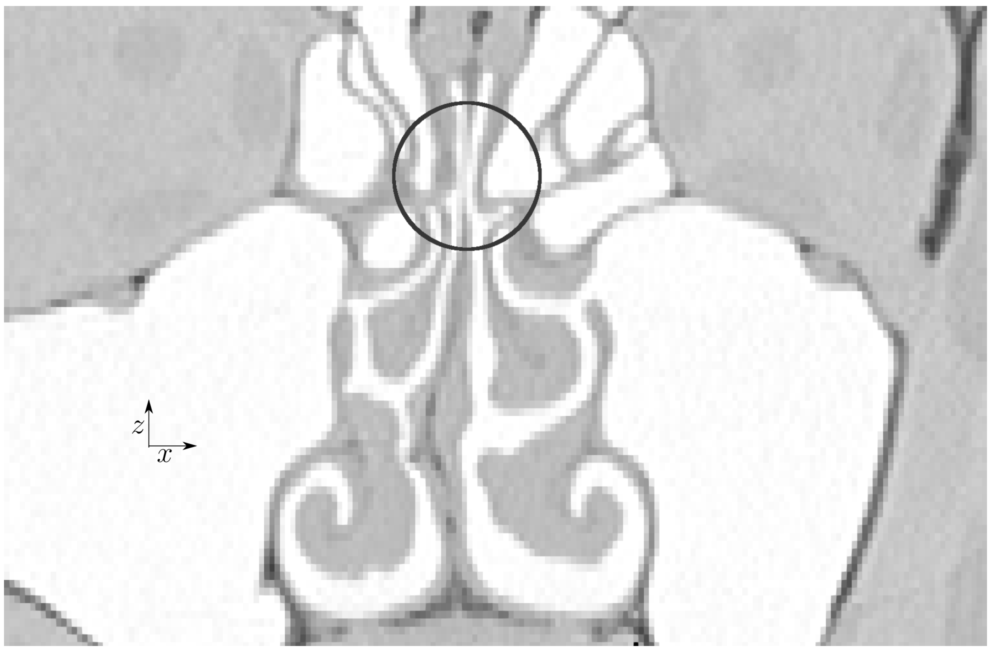


Figure 6.10: Example of a very coarse resolved septum leading to holes in the extracted surface

of 3.5 % and a mean deviation of 1.0 %. The differences for the pressure drop are slightly higher, with a maximum deviation of 33.3 % and a mean deviation of 8.6 %, where the maximum deviation occurs in the cases with artifacts. A finer grid for the LBM method could probably decrease the deviation further but would not fit into the 16 GB device memory of the used GPUs.

6.4.5 Performance Considerations

Generally, the aim of this work was only to prove the feasibility of a fully automatic approach; thus performance considerations were secondary. A direct performance comparison between *lbm4e* and OpenFOAM is difficult due to the different hardware used. One case calculated with LBM took approximately an hour on a single GPU, whereas a single OpenFOAM calculation takes approx. 10 minutes. Besides the different hardware, there are multiple reasons for these differences: The OpenFOAM calculations are done with a steady-state solver whereas the LBM calculations use a transient solver that is run until the solution converges. Multiple methods to decrease the simulation times for steady-state case have been proposed (Guo and Shu, 2013, 103 ff.). Geier et al. (2015) proposed a well-conditioned variant of their cumulant collision operator for single precision calculations, which should roughly halve the computation time. There are regions in the CT scan which are unimportant for the flow but are included in the flow calculations like the paranasal sinuses. Furthermore,

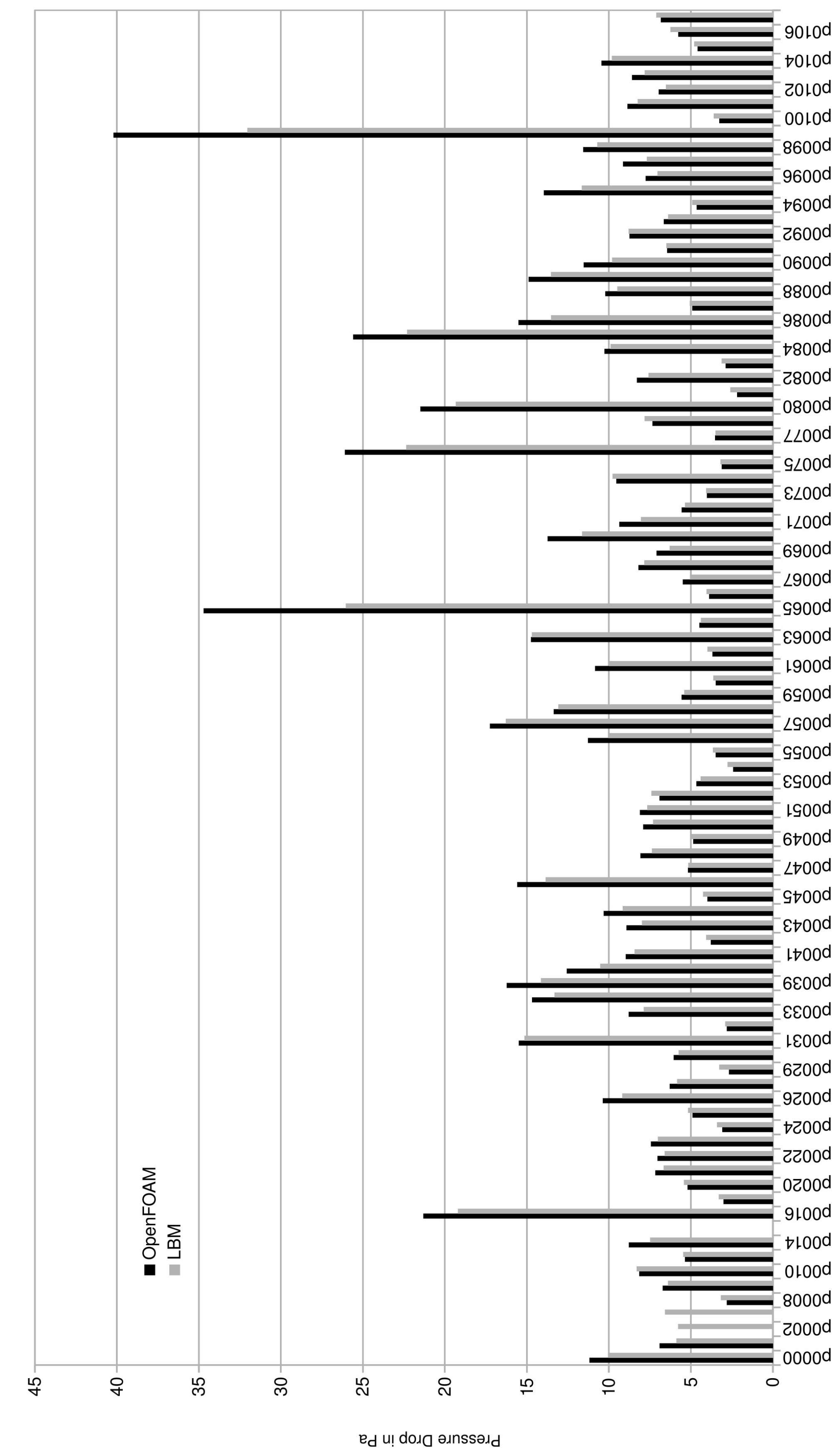


Figure 6.11: Comparison of calculated pressure drops between OpenFOAM and *lbm4e*

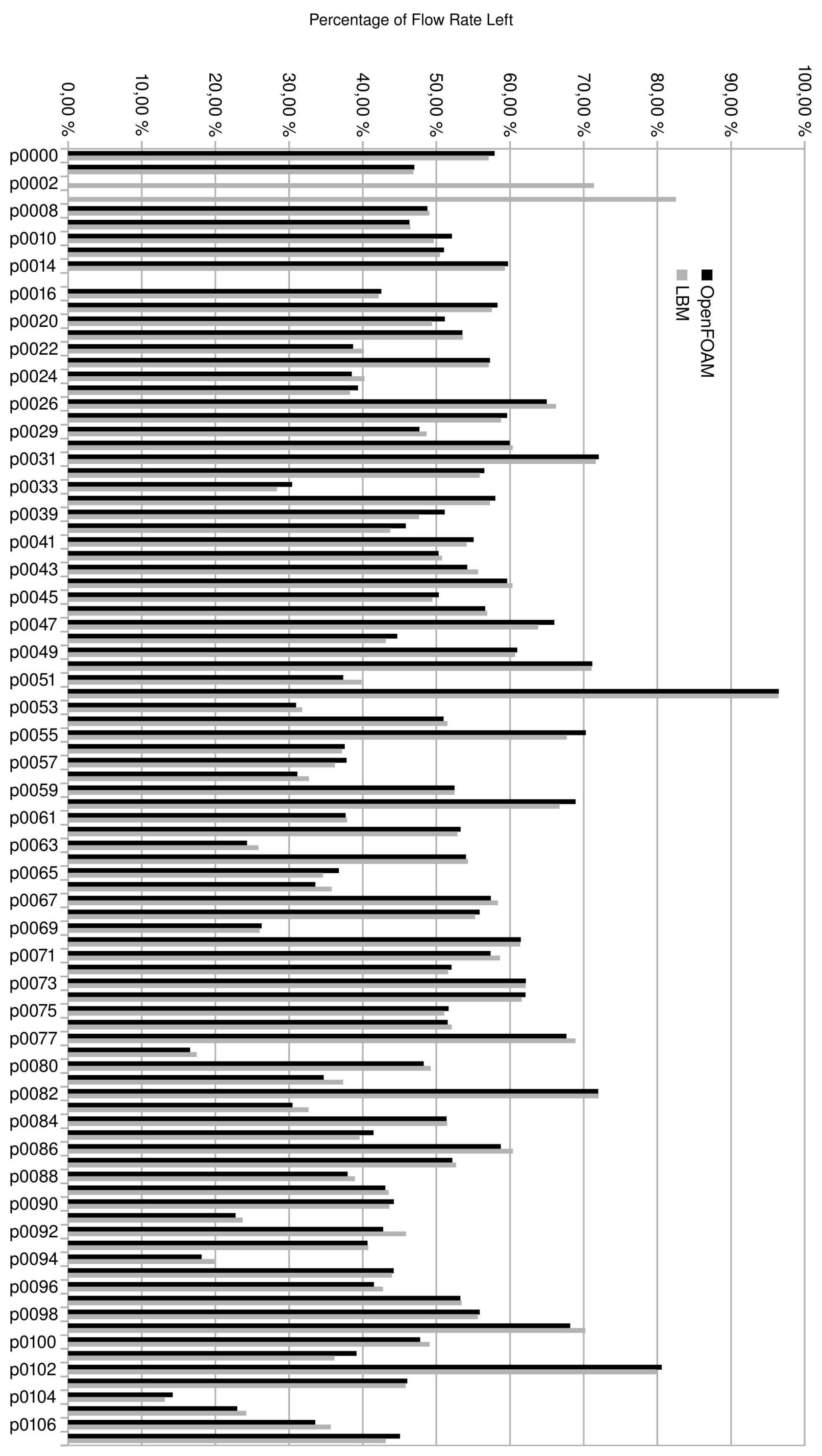


Figure 6.12: Comparison of calculated flow rate partitioning between OpenFOAM and *lbm4e*

the inlet should be placed directly at the nostrils to save the cells outside the nose.

6.5 Removal of Paranasal Sinuses

As stated before, the paranasal sinuses are unimportant for the nasal flow and therefore should be excluded to save computational time. Here, a two stage procedure is proposed to reliably mark cells belonging to a paranasal sinus in a scan: First, the scan is analyzed and all "cavities" are detected. Then in the second stage, these detected cavities are classified as paranasal sinuses or other nasal cavities using machine learning. This method was evaluated by recalculating the flow parameters with the paranasal sinuses removed and comparing to the previous results.

6.5.1 Cavity Detection

The algorithm to detect the cavities is illustrated in figure 6.13. The scan is first binarized (part (a)) and the solid voxels at the air-solid boundary are then enlarged by using a BFS (b). As cavities are only connected to other areas by small passages, the latter may vanish and the scan decomposes into multiple regions, which are then labeled using connected component analysis (c). The enlarged voxels are removed and in order to compensate for the enlarged solid voxels, the labeled regions are also enlarged as much as the solid voxels were enlarged before (d). There might be small cavities of unmarked voxels which were previously overlaid by the enlarged solid voxels. The regions are marked by another connected component analysis (e). If a newly marked region has only a single neighboring region that was labeled in the first step, both regions are merged (f).

6.5.2 Cavity Classification using Machine Learning

The next step is to classify the cavities for which a k-nearest neighbor (Fix et al., 1985) based algorithm is used. A k-nearest neighbor generally predicts the class of an object by determining the k most similar objects from a sample set, where k is usually a small number that can be used to tune the algorithm, and using the most common class of these k objects as the prediction. The algorithm therefore does not have any training phase in which parameters are learned from the sample set, but the time needed for a prediction is linearly dependent on the sample set size. An advantage is that each of its steps has an interpretable meaning whereas a neural network for example is more of a black box.

First, a similarity function has to be defined. The scans can be interpreted as a collection of voxels. A function g^A is defined that denotes whether or not a voxel of scan A with

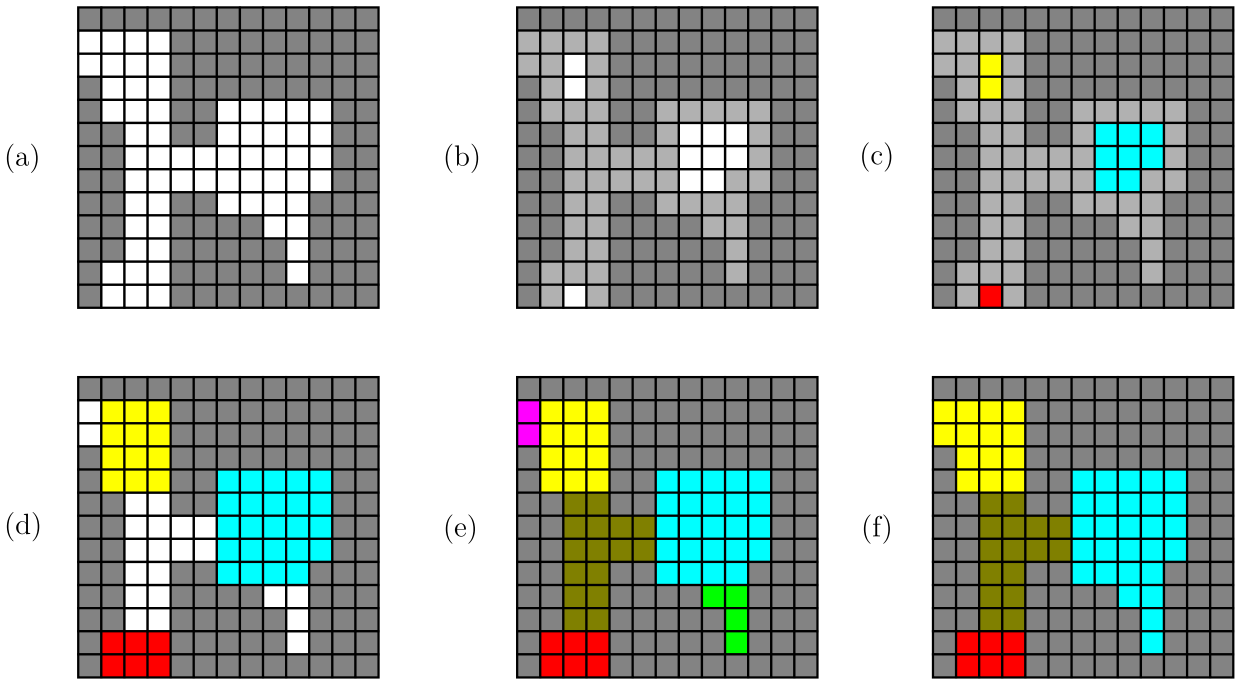


Figure 6.13: Illustration of the cavity detection: (a) initial state; (b) solid voxels at the air-solid boundary are enlarged; (c) connected components marked; (d) connected components enlarged; (e) connected component analysis of unmarked cells; (f) neighboring regions were merged

a certain index belongs to an air filled part of the nose (the nasal cavity or one of the paranasal sinuses):

$$g : \mathbb{N}^3 \to \{0,1\} \quad g(i,j,k) := \begin{cases} 1 & \text{voxel (i,j,k) belongs to nasal cav. or paran. sinus} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

The similarity $sim(A, B)$ between two scans A and B is then computed as the number of voxels with different values for g. Two very similar scans will have a very low similarity score. As the scans might not be aligned to each other, the minimum of all possible translations may be used as the final result. Rotations and different scaling are not considered here as the scans used generally have similar resolution and rotation.

$$sim(A,B) := \min_{i',j',k'} \sum_{i,j,k} |g^A(i,j,k) - g^B(i+i', j+j', k+k')| \tag{6.2}$$

As the function g outputs binary values, these can be saved memory efficient as a bit-array. Comparison of two scans can then be implemented efficiently with a bitwise XOR-operator and the sum accelerated with a population count hardware instruction.

Although it is possible to compute the minimum value and thereby the best overlap by brute-forcing all possible translations in a reasonable time, this leads to an extended evaluation time for many scans. Therefore, a better approach is needed: The similarity values for translations that are a multiple of 3 are calculated first and the 10 translations yielding the lowest similarity values are then used as starting points for neighboring translations. Although this is not guaranteed to give the optimal solution, the solution generally is sufficiently close to the optimal.

After the k most similar scans are computed, a prediction for the cavities has to be made. For each cavity, that has to be classified, it is iterated over all of its voxels. For each voxel is checked to which class the voxel belongs in the translated similar scans and the vote count for this class is incremented. The class with the most counts is the final prediction for the cavity.

One might also not match the whole nasal cavity for labeling and only compare a cavity to a sample set of different paranasal sinuses. However, matching the whole nasal cavity is probably superior as it also makes use of the relative positions of the nasal cavities.

6.5.3 Evaluation

To evaluate the presented algorithms, the 10 largest cavities for 91 scans have been manually categorized. Since the data set is relatively small, cross-evaluation is used, i. e. each scan is evaluated against all other scans.

There are multiple possible errors that can be categorized into three types: A cavity can be wrongfully labeled as a paranasal sinus (I), a paranasal sinus can be wrongfully labeled as not being a paranasal sinus (II) and a paranasal sinus can be wrongfully labeled as a different paranasal sinus (III) (e. g. a maxillary sinus is labeled as a frontal sinus). If reduction of the computational time for CFD calculations is the goal, then the last type of error is unimportant. The second type will only extend the computational time but should give accurate results. The first type might influence the results and is therefore the most critical type.

One can either differentiate between the different paranasal sinuses or use just “nasal cavity” and “paranasal sinus” as two categories since for the sake of removing the paranasal sinuses the actual classification is unimportant.

Table 6.1 shows the number of error types for the sample. The critical type I error is very rare for k higher than one. The higher number of type II errors can be explained by the incomplete labeling of the sample set. As stated, only the largest cavities have been

	Diff.			N. C./P. S.		
	k			k		
Error Type	1	3	5	1	3	5
N. C. as P. S. (I)	3	1	1	3	1	1
P. S. as N. C. (II)	44	27	30	44	25	27
P. S. as P. S. (III)	24	17	17		(NA)	

Table 6.1: Amount of errors according to type for a subset of 20 scans and different values of k. Diff.: with differentiation between different paranasal sinuses; N. C./P. S with differentiation only between nasal cavity (N. C.) and paranasal sinuses (P. S.).

classified. Therefore, some smaller cavities that are actually paranasal sinuses are counted as nasal cavity leading to more errors of this type. The results also show that a higher value of k like five can lead to worse results as also non-similar data sets are used for the prediction.

To quantify the savings in computational time as well as a possible change in the results, flow parameters are calculated for a randomly chosen subset of 20 samples. The meshes for LBM generated without the predicted paranasal sinuses had between 16 % and 33 % fewer cells, on average 25 % less, than those generated with the full geometry. A direct comparison in computational time cannot be made as the calculations were made with different GPU driver versions, which lead to an additional speedup. The results were very similar with an average deviation of 0.51 % for the pressure drop (2.31 % maximum deviation) and 0.17 % for the flow distribution (1.54 % maximum).

Even if some parts of the nasal cavity are classified as paranasal sinus and therefore removed, this does not necessarily lead to a significant error. For the one case in which this error occurred, the deviation was less than 1 %.

6.5.4 Nasopharynx Reconstruction

As artifacts in the nasopharynx were the most common type of artifacts found in the database, a method to remove those is proposed here for future work. A variant of the presented method for cavity detection and classification can possibly be used to accomplish this: The cavity detection also detects the nasopharynx as a cavity and they were labeled as such in the sample set. One could correct an artifact in the nasopharynx by computing the most similar scan in the database and then copy the nasopharynx from that scan to the scan with the artifact. As the pressure drop is mainly caused by the narrow positions in the nasal valve, the exact form of the nasopharynx should have little impact on the flow. Evaluation of this method could be achieved by artificially closing the nasopharynx for some scans and comparing the results of this method to those for the intact nasopharynx.

6.6 Summary

A fully automatic approach for the calculation of flow parameters for flow through the nasal cavity was presented. The method preprocesses CT scans and extracts a surface mesh which is then used as input for FVM or LBM flow calculations. For the flow calculation two methods, the FVM based solver OpenFOAM and LBM based solver *lbm4e* were compared. The FVM based method required less time in our setup for the calculations but was slightly less reliable than LBM. Furthermore, an algorithm to decide if cells of a mesh belong to the left or the right side of the nasal cavity was introduced. The method was evaluated with 94 different cases. A good agreement between OpenFOAM and LBM was achieved.

As paranasal sinuses form dead ends, they can be excluded without loss of accuracy. A two stage algorithm that first detects cavities and then labels them using machine learning was also introduced. The method lead to an average cell reduction of 25 % while changing the results only slightly. The cell reduction can possibly be increased by using a larger and completely labeled sample set.

The most common artifact in the scans preventing successful calculation was soft tissue movement caused by the patient swallowing during the examination. A method to correct these artifacts by a similar algorithm as used for paranasal sinus removal was also outlined for future work.

7 Summary

In this work, a novel LBM implementation was presented. It features good portability to various systems and is able to utilize various hardware like GPUs and CPUs. Furthermore, an algorithm for grid generation suitable for massively parallel hardware was introduced. To overcome the bottleneck of slow IO, in-situ visualization was used. This implementation was then used to calculate flow parameters for two different problems fully automatically: the pressure drop in particle packings and the flow in the human nasal cavities.

The accurate prediction of pressure drop in porous media is still challenging as there are various correlations, which differ from each other within a span of approximately 20 %. The presented method was used to calculate the pressure drop of about 2000 sphere packings with different geometric constraints and by doing so generating a larger database than experimentally possible. The results generally agree well with common correlations, but it has been shown that for small hydraulic diameter to sphere diameter ratios the pressure drop is a non-monotonic function. This work was done in the limit of zero Reynolds number but should be extended to all ranges of Reynolds numbers in the future.

Furthermore, a method to automatically calculate nasal cavity flow was presented, including algorithms for pre-processing of the CT scans, as well as partitioning the mesh into left and right nasal cavity for evaluation. The method works for both LBM as well as classical, FVM based CFD solvers, like OpenFOAM. The influence of parameters like smoothing and the threshold value for binarization was investigated. The results for the novel LBM implementation and OpenFOAM agreed well with a mean deviation of 1.0 % for the flow partitioning and 8.6 % for the pressure drop. The LBM based flow calculation had a slightly higher rate of successful calculations. Moreover, a method based on machine learning to automatically detect and remove paranasal sinuses was proposed which leads to a speed-up for the calculations while giving basically the same results (mean deviation of 0.51 % for the pressure drop). In future work, this method should be used to automatically calculate flow parameters for whole databases of CT scans and search for correlations between those parameters and severity of symptoms as quantified by the NOSE scale.

Another problem for which an automatic setup was developed is the Fischer-Tropsch synthesis. Here, a three dimensional model for diffusion and reaction in catalyst was presented based on a model by Becker et al. (2014, 2016). Using this model, it could be shown that for transport pores smaller than 50 µm, no gradients orthogonal to the catalyst layer occur and the problem can therefore be treated as one-dimensional. Furthermore, optimal catalyst layer heights and transport pore porosities were calculated for different transport pore diameters. It was shown that for large pores, the decrease in ATY can partly be compensated by a higher catalyst layer and a higher transport pore porosity. Transport pores, however, have to be smaller than 250 µm to achieve a gain over an optimal dense catalyst layer. Since the used structured grids used in the calculations are easy to create, the finite volume method was used to calculate the solution.

These three examples also show the three main advantages of full automation:

1. Full automation allows to generate **more data points** than with manual processing or experimentally for the same amount of effort. This allows to create more accurate models for the pressure drop in sphere backed beds or to use statistical methods to understand NSD better.

2. Full automation allows using **CFD as an element of a more extensive simulation**. This was used to optimize catalyst layers with transport pores. Instead of calculating the ATY for a few selected parameters, algorithmic optimization could be used that utilized a finite-volume calculation to assess each optimization stage.

3. Full automation allows a **broader applicability of CFD**. Usually, the application of CFD requires specialized knowledge e. g. for grid generation, but with automatic calculation CFD could be used in clinical practice to assess severity of NSD or for virtual surgery planning.

For future work, automation techniques like those presented in this thesis can certainly be adapted to other applications. Generally, LBM is a promising method for automated calculations due to the easy mesh generation. For FVM based approaches, the development of reliable mesh generation and robust solvers should be focused.

Bibliography

Armaly, B. F., Durst, F., Pereira, J., and Schönung, B. (1983). Experimental and theoretical investigation of backward-facing step flow. *Journal of fluid Mechanics*, 127:473–496.

Becker, H., Güttel, R., and Turek, T. (2014). Optimization of catalysts for fischer-tropsch synthesis by introduction of transport pores. *Chemie Ingenieur Technik*, 86(4):544–549.

Becker, H., Güttel, R., and Turek, T. (2016). Enhancing internal mass transport in fischer–tropsch catalyst layers utilizing transport pores. *Catal. Sci. Technol.*, 6:275–287.

Bhatnagar, P. L., Gross, E. P., and Krook, M. (1954). A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Phys. Rev.*, 94:511–525.

Bouzidi, M., Firdaouss, M., and Lallemand, P. (2001). Momentum transfer of a boltzmann-lattice fluid with boundaries. *Physics of Fluids*, 13(11):3452–3459.

Brent, R. P. (2002). *Algorithms for minimization without derivatives.* Dover Publications, Mineola, N.Y.

Carman, P. C. (1937). Fluid flow through granular beds. *Transactions of the Institution of Chemical Engineers*, 15:150–166.

Clift, R., Grace, J. R., and Weber, M. E. (2005). *Bubbles, drops, and particles.* Courier Corporation.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition.

Dammertz, H. and Keller, A. (2006). Improving ray tracing precision by object space intersection computation. In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 25–31.

Dennis, S. and Chang, G.-Z. (1970). Numerical solutions for steady flow past a circular cylinder at reynolds numbers up to 100. *Journal of Fluid Mechanics*, 42(3):471–489.

Desmond, K. W. and Weeks, E. R. (2009). Random close packing of disks and spheres in confined geometries. *Physical Review E*, 80(5):051305.

d'Humières, D. and Ginzburg, I. (2009). Viscosity independent numerical errors for lattice boltzmann models: From recurrence equations to "magic" collision numbers. *Comput. Math. Appl.*, 58(5):823–840.

d'Humières, D., Ginzburg, I., Krafczyk, M., Lallemand, P., and Luo, L.-S. (2002). Multiple-relaxation-time lattice boltzmann models in three dimensions. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 360(1792):437–451.

Donev, A., Torquato, S., and Stillinger, F. H. (2005a). Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. *J. Comput. Phys.*, 202(2):765–793.

Donev, A., Torquato, S., and Stillinger, F. H. (2005b). Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. i. algorithmic details. *J. Comput. Phys.*, 202(2):737–764.

Eisfeld, B. and Schnitzlein, K. (2001). The influence of confining walls on the pressure drop in packed beds. *Chemical Engineering Science*, 56(14):4321 – 4329.

Ergun, S. (1952). Fluid flow through packed columns. *Chemical Engineering Progress*, 48(2):89–94.

Erkey, C., Rodden, J. B., and Akgerman, A. (1990). A correlation for predicting diffusion coefficients in alkanes. *The Canadian Journal of Chemical Engineering*, 68(4):661–665.

Erturk, E. (2008). Numerical solutions of 2-d steady incompressible flow over a backward-facing step, part i: High reynolds number solutions. *Computers & Fluids*, 37(6):633–655.

Fix, E., Hodges, J., and of Aviation Medicine, U. S. (1985). *Discriminatory Analysis: Nonparametric Discrimination, Consistency Properties*. Number Bd. 1-2 in Discriminatory Analysis: Nonparametric Discrimination, Consistency Properties. USAF School of Aviation Medicine.

Fornberg, B. (1980). A numerical study of steady viscous flow past a circular cylinder. *Journal of Fluid Mechanics*, 98(4):819–855.

Geier, M., Greiner, A., and Korvink, J. G. (2009). Bubble functions for the lattice boltzmann method and their application to grid refinement. *The European Physical Journal Special Topics*, 171(1):173–179.

Geier, M. and Schönherr, M. (2017). Esoteric twist: An efficient in-place streaming algorithmus for the lattice boltzmann method on massively parallel hardware. *Computation*, 5(2).

Geier, M., Schönherr, M., Pasquali, A., and Krafczyk, M. (2015). The cumulant lattice boltzmann equation in three dimensions: Theory and validation. *Computers & Mathematics with Applications*, 70(4):507 – 547.

Ghia, U., Ghia, K. N., and Shin, C. (1982). High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411.

Ginzburg, I. (2006). Variably saturated flow described with the anisotropic lattice boltzmann methods. *Computers & Fluids*, 35(8–9):831 – 848. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

Ginzburg, I. and d'Humières, D. (2003). Multireflection boundary conditions for lattice boltzmann models. *Phys. Rev. E*, 68:066614.

Ginzburg, I., Verhaeghe, F., and Dhumieres, D. (2008). Study of simple hydrodynamic solutions with the two-relaxation-times lattice boltzmann scheme. *Communications in Computational Physics*, 3:519–581.

Guo, Z. and Shu, C. (2013). *Lattice Boltzmann Method and Its Applications in Engineering*. World Scientific.

Guo, Z.-L., Zheng, C.-G., and Shi, B.-C. (2002). Non-equilibrium extrapolation method for velocity and pressure boundary conditions in the lattice boltzmann method. *Chinese Physics*, 11(4):366–374.

Harel, G., Lekien, J.-B., and Pébaÿ, P. P. (2017). Visualization and analysis of large-scale, tree-based, adaptive mesh refinement simulations with arbitrary rectilinear geometry.

Hasimoto, H. (1959). On the periodic fundamental solutions of the stokes equations and their application to viscous flow past a cubic array of spheres. *Journal of Fluid Mechanics*, 5:317–328.

Hou, S., Zou, Q., Chen, S., Doolen, G., and Cogley, A. C. (1995). Simulation of cavity flow by the lattice boltzmann method. *Journal of computational physics*, 118(2):329–347.

Hänel, D. (2004). *Molekulare Gasdynamik : Einführung in die kinetische Theorie der Gase und Lattice-Boltzmann-Methoden*. Springer, Berlin [u.a.].

Hörschler, I., Brücker, C., Schröder, W., and Meinke, M. (2006). Investigation of the impact of the geometry on the nose flow. *European Journal of Mechanics - B/Fluids*, 25(4):471 – 490.

Janovic, N., Janovic, A., Milicic, B., and Djuric, M. (2020). Relationship between nasal septum morphology and nasal obstruction symptom severity: computed tomography study. *Brazilian Journal of Otorhinolaryngology.*

Janßen, C. F., Koliha, N., and Rung, T. (2015). A fast and rigorously parallel surface voxelization technique for gpu-accelerated cfd simulations. *Communications in Computational Physics*, 17(5):1246–1270.

Johannessen, E., Wang, G., and Coppens, M.-O. (2007). Optimal distributor networks in porous catalyst pellets. i. molecular diffusion. *Industrial & Engineering Chemistry Research*, 46(12):4245–4256.

Karras, T. (2012). Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, page 33–37, Goslar, DEU. Eurographics Association.

Liu, S., Afacan, A., and Masliyah, J. (1994). Steady incompressible laminar flow in porous media. *Chemical Engineering Science*, 49(21):3565–3586.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.

Macdonald, I. F., El-Sayed, M. S., Mow, K., and Dullien, F. A. L. (1979). Flow through porous media-the ergun equation revisited. *Industrial & Engineering Chemistry Fundamentals*, 18(3):199–208.

Marano, J. J. and Holder, G. D. (1997). Characterization of fischer-tropsch liquids for vapor-liquid equilibria calculations. *Fluid Phase Equilibria*, 138(1-2):1–21.

Maretto, C. and Krishna, R. (1999). Modelling of a bubble column slurry reactor for fischer–tropsch synthesis. *Catalysis Today*, 52(2):279–289.

Mehta, D. and Hawley, M. C. (1969). Wall effect in packed columns. *Industrial & Engineering Chemistry Process Design and Development*, 8(2):280–282.

Mladina, R., Čujić, E., Šubarić, M., and Vuković, K. (2008). Nasal septal deformities in ear, nose, and throat patients: An international study. *American Journal of Otolaryngology*, 29(2):75–82.

Nieuwstadt, F. and Keller, H. (1973). Viscous flow past circular cylinders. *Computers & Fluids*, 1(1):59–71.

Ohira, N. (2017). Memory-efficient 3d connected component labeling with parallel computing. *Signal, Image and Video Processing*, 12(3):429–436.

Pan, C., Luo, L., and Miller, C. (2006). An evaluation of lattice boltzmann schemes for porous medium flow simulation. *Computers and Fluids*, 35(8-9):898–909.

Pasquali, A., Schönherr, M., Geier, M., and Krafczyk, M. (2013). Lbmhexmesh: an openfoam based grid generator for the lattice boltzmann method (lbm).

Quadrio, M., Pipolo, C., Corti, S., Lenzi, R., Messina, F., Pesci, C., and Felisati, G. (2013). Review of computational fluid dynamics in the assessment of nasal air flow and analysis of its limitations. *European Archives of Oto-Rhino-Laryngology*, 271(9):2349–2354.

Roshko, A. (1993). Perspectives on bluff body aerodynamics. *Journal of Wind Engineering and Industrial Aerodynamics*, 49(1):79–100.

Saiki, E. and Biringen, S. (1996). Numerical simulation of a cylinder in uniform flow: application of a virtual boundary method. *Journal of computational physics*, 123(2):450–465.

Sangani, A. and Acrivos, A. (1982). Slow flow through a periodic array of spheres. *International Journal of Multiphase Flow*, 8(4):343 – 360.

Schroeder, W., Martin, K., Lorensen, B., and Kitware, I. (2006). *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Kitware.

Slotnick, J. P., Khodadoust, A., Alonso, J. J., Darmofal, D. L., Gropp, W. D., Lurie, E. A., and Mavriplis, D. J. (2014). Cfd vision 2030 study: A path to revolutionary computational aerosciences.

Song, C., Wang, P., and Makse, H. A. (2008). A phase diagram for jammed matter. *Nature*, 453:629–632.

Standring, S. (2016). *Gray's anatomy: the anatomical basis of clinical practice*. Elsevier Limited.

Teixeira, J., Certal, V., Chang, E., and Camacho, M. (2015). Nasal septal deviations: A systematic review of classification systems. *Plastic Surgery International*, 2016.

Vervloet, D., Kapteijn, F., Nijenhuis, J., and van Ommen, J. R. (2012). Fischer–tropsch reaction–diffusion in a cobalt catalyst particle: aspects of activity and selectivity for a variable chain growth probability. *Catalysis Science & Technology*, 2(6):1221.

WebMD (2020). Deviated septum. https://www.webmd.com/allergies/deviated-septum. last checked on 2021-06-18.

Williams, S., Waterman, A., and Patterson, D. A. (2009). Roofline: An Insightful Visual Performance Model for Multicore Architectures.

Woop, S., Benthin, C., and Wald, I. (2013). Watertight ray/triangle intersection. *Journal of Computer Graphics Techniques (JCGT)*, 2(1):65–82.

Yates, I. C. and Satterfield, C. N. (1991). Intrinsic kinetics of the fischer-tropsch synthesis on a cobalt catalyst. *Energy & Fuels*, 5(1):168–173.

Zhavoronkov, N., Aerov, M., and Umnik, N. (1949). Hydraulic resistance and density of packing of a granular bed. *J. Phys. Chem.*, 23:342–261.

Zwicker, D., Yang, K., Melchionna, S., Brenner, M. P., Liu, B., and Lindsay, R. W. (2018). Validated reconstructions of geometries of nasal cavities from CT scans. *Biomedical Physics & Engineering Express*, 4(4):045022.

Short Manual

1 Building and Installing

The LBM solver *lbm4e* uses the free building tool CMake[1]. At least version 3.12 is required. CMake does not directly build the software but only generates the necessary configuration files for the actual build tool like Ninja or Make. There are three possible ways to use CMake: the command line application cmake, the terminal application ccmake and the GUI application cmake-gui. In addition to the standard CMake variables, *lbm4e* provides additional variables that control which parts of lbm4e are built, if optional external libraries are used and if external libraries are built or searched for an installed version. The variables which control what parts will be built can be found in the following table:

Variable	Description	Default Value
BUILD_DOC	Whether or not a Doxygen documentation is built (requires doxygen installed)	OFF.
BUILD_SHARED_LIBS	If set, shared linking is used and static linking otherwise	ON.
BUILD_gui	Whether to build the graphical tool lbmGui for setting options and running cases	OFF.
BUILD_lbm4e	Whether to build the solver application	ON.
BUILD_lbmMesh	Whether to build a meshing tool based on section 3.4.2	OFF.
BUILD_lbmMeshGPU	Whether to build a meshing tool based on section 3.4.3	OFF.
BUILD_lbmUtilities	Whether to build utility applications	OFF.

The variables which control which optional external libraries will be used can be found in the following table:

[1]https://cmake.org/

Variable	Library	Default Value
ENABLE_openmp	OpenMP	ON.
ENABLE_openCL	OpenCL	ON.
ENABLE_hip	HIP	OFF.
ENABLE_catalyst	Paraview Catalyst	OFF.

As mentioned, some of the external libraries can be built by the superbuild scheme, whereas some have to be installed and some are optional:

Library	Is Optional?	Can Be Built?
OpenMP	yes	no
Paraview Catalyst	yes	no
Libconfig	no	yes
HDF5	no	yes
OpenCL	yes	no
HIP	yes	no
QT	no	yes
VTK	no	yes

Note that some of these libraries are only required to build certain parts and are therefore not required when these parts are not built.

The code can be built with a variety of C++ compilers which support the C++11 standard. The code was mainly tested on GNU/Linux. However, as cross-compilation to Windows was tested once successfully, native compilation under Windows should be possible. It is however unclear, whether the superbuild scheme supports building of external libraries in this case.

2 Running the Code

The workflow of running the code is similar to that of OpenFOAM. Each case has its own directory with certain subdirectories with a specific purpose:

Directory	Content
config	configuration files
mesh	generated mesh
geometry	surface meshes for mesh generation
data	time step (checkpoint) data

vtk	vtk files with macroscopic quantities
log	duplicate of the terminal output of applications

Such a case directory can be created by using the lbmCreateCase utility. The configuration is distributed over different configuration files:

File	Content
mesh.cfg	Overall domain, surface mesh and resolution of the mesh
general.cfg	Which device to use for calculation, number of iterations to run/convergence criteria
properties.cfg	Material properties of the fluid, boundary conditions, initial conditions, collision model
coprocessing.cfg	Co-processing pipeline and number to threads to use for co-processing

After setting up and configuring a case, the mesh can be created with one of the mesh utilities. It can be converted to a vtk file with the lbmConvert utility. The calculation is then started by invoking the solver lbm4e.