Graz University of Technology

Faculty of Civil Engineering Sciences
Institute of Hydraulic Engineering and
Water Resources Management

Faculty of Bioengineering Sciences
Department of Mathematical Modelling,
Statistics and Bioinformatics

Computational fluid dynamics modelling of large, shallow water reservoirs

Master's Thesis

Kristina Maierhofer

Graz University of Technology
Promoter · Univ.-Prof. Dipl.-Ing. Dr.techn. Gerald Zenz
Tutor · Univ.-Prof. Dr.-Ing. Dirk Muschalla
Tutor · Dipl.-Ing. Wolfgang Richter

Ghent University
Ass.Prof. Dr. ir. Ingmar Nopens · *Promoter*
ir. Daan Van Hauwermeiren · *Tutor*

Submitted in partial fulfilment of the requirements for the degree of
Master of Science in Civil Engineering - Geotechnics and Hydraulics

Graz, May 2015

The author and promoter give the permission to use this thesis for consultation and to copy parts of it for personal use. Every other use is subject to the copyright laws, more specifically the source must be extensively specified when using results from this thesis.

June 2015

The promoter,	The promoter,	The author,
Univ.-Prof. Gerald Zenz	Ass.Prof. Ingmar Nopens	Kristina Maierhofer

Statutory declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources or resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, June 2015

The author,

Kristina Maierhofer, BSc

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen oder Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Graz, Juni 2015

Die Autorin,

Kristina Maierhofer, BSc

Für Lukas

Acknowledgement

Thanks!

Abstract

The objective of this Master's thesis is the development of a computational fluid dynamics (CFD) model of a water reservoir being part of the drinking water production center *De Blankaart*. Since, the water treatment plant does not meet any longer the current technical requirements, it will undergo renovation. As part of the regeneration, an improvement of the flow pattern inside the reservoir is being considered. At present, non-uniform discharge, dead zones and partly long residence time cause deterioration of the water quality. A possible structural measure is the relocation of the outlet structure, if this were to improve the general flow behaviour in the reservoir.

In the course of this work, a computational fluid dynamics model is developed serving as a planning basis for the renovation of the water storage reservoir. By means of the CFD model the main water flow is identified and further analyses were investigated to demonstrate whether the relocation of the outlet has a beneficial effect on the water flow.

As a first step, a structured mesh is designed for a simplified geometry of the reservoir. The mesh quality and mesh independence are checked by comparison of sampled velocity data. Then, a computational fluid dynamics model is developed applying the finite volume method to determine the present flow pattern. The results are in accordance with the expectations. The flow pattern is characterised by a main stream around the partition wall and large areas of dead zones.

Based on the developed model, different scenarios are simulated concerning improvement measures in order to achieve a more uniform flow. In a first case scenario, the outlet is relocated to a neighbouring wall with the aim of improving the flow pattern. Unfortunately, the relocation causes additional dead zones and does not show any improvement.

With respect to the results obtained, another improvement measure is investigated. In a second scenario, an internal wall is located in the center of the modelled reservoir in order to prevent the "shortcut" of the main flow around the partition wall. The results show a general improvement of the flow pattern. The internal wall acting as a barrier causes mixing and circulation of the water. Furthermore, the occurrence of dead zones decreases significantly. According to the results received, the location of an internal wall is proved to have a positive impact on the flow pattern.

Zusammenfassung

Das Ziel dieser Masterarbeit ist die Entwicklung eines "computational fluid dynamics" (kurz CFD -) Modell eines Wasserreservoirs, das Bestandteil der Trinkwasseraufbereitungsanlage *De Blankaart* (Belgien) ist. Auf Grund veralteter Ausstattung und Bausubstanz wird die gesamte Anlage erneuert. Im Zuge der Erneuerungsarbeiten wird eine Verbesserung des Strömungsverhaltens angestrebt. Derzeit verursachen ungleichförmige Strömungsverteilungen, Totzonen, und lange Aufenthaltszeiten schlechte Wasserqualität, und fördern die Entstehung von Algen und anaeroben Bedingungen. Eine mögliche konstruktive Maßnahme ist die Versetzung des Auslaufbauwerks, falls dies eine Verbesserung hervorrufen würde.

Im Rahmen dieser Arbeit wird ein numerisches Modell erstellt, das als Planungsgrundlage für die Renovierung des Reservoirs dienen soll. Mit Hilfe des entwickelten Modells werden die gegenwärtigen Strömungen bestimmt und analysiert. Darüber hinaus wird untersucht inwieweit die Lagenveränderung des Auslaufes die Strömungsbedingungen im Reservoir positiv beeinflusst.

Als erstes wird ein strukturiertes Rechennetz für ein vereinfachtes Modell des Wasserreservoirs generiert. Die Qualität und die entsprechende Gitterweite des Netzes werden durch Datenvergleiche überprüft. Mit dem Netz wird unter Anwendung der Finiten-Volumen-Methode ein CFD - Modell erstellt um das Strömungsbild zu bestimmen. Das Ergbnis der Simulation zeigt einen Hauptstrom entlang der Trennwand fließen sowie zwei weitflächige Totzonen.

Nach erfolgreicher Modellierung des ursprünglichen Reservoirs, werden verschiedene Szenarien simuliert um geeignete Maßnahmen zur Verbesserung des Strömungszustandes zu entwickeln. In einer ersten Fallstudie, wird der Auslauf, mit dem Zweck eine gleichmäßige Strömungsverteilung zu erhalten, versetzt. Jedoch zeigen die Ergebnisse eine generelle Verschlechterung des Strömungsbildes.

In Anbetracht dieser Erkenntnis wird eine weitere Untersuchung durchgeführt. In einer zweiten Fallstudie, wird eine zusätzliche interne Wand konstruiert. Das Ergebnis zeigt, dass diese Maßnahme zu einer Verbesserung führt. Die interne Wand verursacht eine Durchmischung und Zirkulation des Wassers, und verkleinert somit die Totzonen im gesamten Reservoir. Auf Grund dieser Ergebnisse, kann die Konstruktion einer zusätzlichen Wand im Reservoir als Verbesserungsmaßnahme in Betracht gezogen werden.

Contents

List of Figures

List of Tables

Preface

The present Master's thesis was realised in cooperation with *Ghent University* and the public water supplier of the Flemish government *De Watergroep*. The study was mainly conducted at *Ghent University* under the guidance of Daan Van Hauwermeiren and under the supervision of Ingmar Nopens. Ben Cools was my chief supporter from the company. *De Watergroep* stated the research question and made available the necessary information and data for the subject of investigation. Furthermore, the company provided the entire work equipment. *De Watergroep*, especially Ben Cools, Liesbeth Verdickt, and Gisèle Peleman, supported me throughout the entire duration of research.

CHAPTER 1

Problem statement, research objectives and outline

"Water is not a commercial product like any other but, rather, a heritage which must be protected, defended and treated as such."

European Parliament, Directive 2000/60/EC (?)

1.1 Introduction

Drinking water is the most important source of life. Its availability and accessibility must be secured for everyone. However, in today's time of extensive environmental pollution and progressing climate change, it becomes more and more difficult to meet the increasing water demand of our growing population. For this reason, an economical and considerate exploitation of our water resources is crucial for our future security of water supply.
It is therefore a matter of importance that today's drinking water supply is adapted perfectly to the latest technologies, which enables us to attain a sustainable use of the earth's water resources ensuring the world's future water requirements.

The present work contributes to the improvement of the ecological efficiency of a drinking water production center. The water production plant does not meet any longer the current technical requirements and is therefore to be renewed. The computational model developed in the course of this thesis will serve as a planning basis for the renovation of the water storage reservoir of the production center.
From the scientific point of view, the application of computational fluid dynamics modelling to large-scale water structures with very low velocities will be investigated and may provide new evidence of applicability and validity of computational fluid dynamics models.

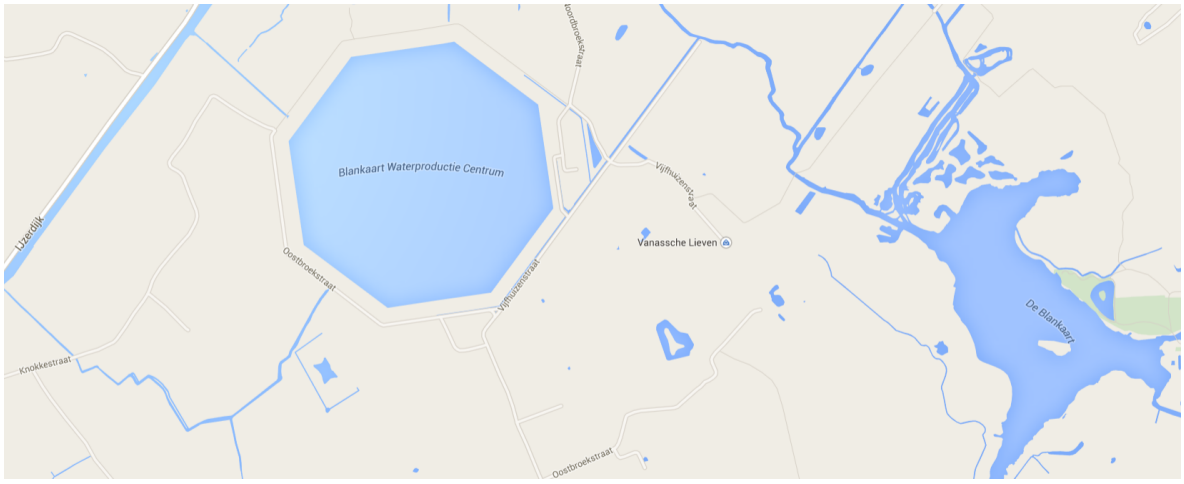
1.2 Problem statement

The purpose of this Master's thesis is the development of a computational fluid dynamic model of a drinking water reservoir (Fig. 1.1) located in West Flanders, Belgium.

Source: `http://www.dewatergroep.be`

Figure 1.1: Aerial photograph of the reservoir *De Blankaart*.

The water reservoir was built in the nineteen seventies as part of the water production center (WTC) *De Blankaart* (see octagonal area of water in Fig. 1.2). Its function is the storage of surface water charged between September and May to ensure the availability of raw water for the drinking water production during summer. The raw water sources are the river *Ijzer* (located to the north-east of the reservoir, see Fig. 1.2), which springs in the north of France and flows into the North Sea in Belgium, and the pond *De Blankaart* (located to the west of the reservoir, see Fig. 1.2), which collects water from the surrounding low lands. The water taken from the river *Ijzer* runs through a ring channel around the reservoir before reaching the intake structure.
The daily water production is about $40\,000\,m^3/day$ from October to May. During the summer months, the daily water production is lowered to about $20\,000\,m^3/day$ due to the limited availability of raw water in summer. The storage capacity of the reservoir is too small to continue production at a rate of $40\,000\,m^3/day$ throughout the year.
After more than forty years in operation, the water production center shows structural problems due to ageing of materials. The purification technology installed is no longer full operative and sustainable. Due to that reason the WTC will undergo renovation in the upcoming years.



Source: `https://www.google.be/maps`

Figure 1.2: Map of region *De Blankaart*: octagonal area of water is the reservoir *De Blankaart*, large river located to the north-east of the reservoir is the river *Ijzer*, water body located to the west of the reservoir is the pond *De Blankaart*.

In the course of the extended regeneration, an improvement of the flow pattern inside the reservoir is being considered. At present, non-uniform discharge, dead zones (see Fig. 1.3) and partly long residence time cause deterioration of the water quality, which leads to the development of algal bloom (see Fig. 1.3)and zones of anaerobic water. The outflow structure might be moved to a neighbouring wall if this were to improve the residence time and decrease the volume of dead zones.

Figure 1.3: Dead zones and algal blooms inside the reservoir: dead zone at the outlet (left figure) and dead zone at a corner (right figure).

1.3 Objectives of this research

The objective of this research is the development of a computational fluid dynamics (CFD) model of the water reservoir *De Blankaart*, in order to identify the present flow pattern and to detect potential dead zones. Further analysis is intended to demonstrate whether the change of the outlet location has a beneficial effect on the water flow inside the reservoir.

1.4 Outline: The road map through this thesis

First the literature on fluid dynamics as well as on drinking water and reservoirs are reviewed. Then, the materials and methods used in this study are discussed. Thereafter the case-study is described in detail, in which the materials and methods of the previous chapter are applied. Finally, the study results are presented and the conclusion and further prospects conclude this thesis.

CHAPTER 2

Literature review

2.1 Introduction

The literature review of this thesis is dealing with fluid dynamics on the one hand and with drinking water and reservoirs on the other hand. The section "fluid dynamics" gives a concise overview of fluid properties and flow classification before discussing fluid kinematics, equations of fluid dynamics, and turbulence modelling in detail.
In a next section, sources and quality requirements of drinking water are discussed. Finally, the structure and functions of reservoirs are reviewed.

2.2 Fluid dynamics

A fluid is a system of many particles interacting with each other. Fluids can be gases, liquids or plasma. It withstands normal compressive stress but no tensile stress, and is not or slightly resisting deformation and fluidity.
In fluid dynamics, the flowing fluid is generally described as a continuum. This means that only its macroscopic quantities are modelled ignoring its molecular behaviour. Instead of calculating the particle interactions, variables like pressure or density are determined for each volume element of the continuum. These volume elements are considerably larger than the intermolecular spaces. Their maximum size is restricted by numerical stability criteria and given tolerances. Their minimum size is limited by the available computational power. The results are only approximations but give a sufficiently accurate prediction of the fluid flow behaviour (Shivamoggi, 1998).

2.2.1 Fluid properties

Referring to fluid dynamic problems, the most important properties of fluids are density, compressibility and viscosity, as well as surface tension and capillarity for liquid fluids. In this study, density, compressibility and viscosity are of main interest, which are explained below.

Density and compressibility

The density ρ of a fluid is defined by mass m per volume V (2.1) and is a function of temperature and pressure.

$$\rho = \frac{m}{V} \tag{2.1}$$

The compressibility K of fluids is the volume elasticity of fluids. Analogous to Hook's law for solids a volume elasticity module for fluids E (2.2) can be defined.

$$\frac{1}{E} = -\frac{\Delta V}{V_0} \cdot \frac{1}{\Delta p} \tag{2.2}$$

where V_0 the initial volume, ΔV the change in volume and Δp the change in pressure. Compressibility K is defined as the reciprocal value of E (2.3).

$$K = \frac{1}{E} = -\frac{\Delta V}{V_0} \cdot \frac{1}{\Delta p} \tag{2.3}$$

The density of a compressible fluid is a function of pressure and temperature. The density variations of incompressible fluids are a function of temperature only. Their density is a fixed property at a certain temperature, independent of pressure. Liquid fluids are mostly treated as quasi-incompressible, because their compressibility is negligibly small at pressures lower than 500 bar (Sigloch, 2014).

Viscosity

The viscosity of a fluid is its resistance to deformation and translation. Due to friction between the particles, mechanical energy dissipates to heat energy. This process is called internal friction. According to Newton, the shear stress τ is proportional to the spatial gradient of velocity $\frac{\mathrm{d}u}{\mathrm{d}y}$ (2.4). This means that pressure has no influence on the fluid's shear stress.

$$\tau = \mu \cdot \frac{\mathrm{d}u}{\mathrm{d}y} \tag{2.4}$$

where μ is the dynamic viscosity.
In fluid dynamics, mostly, the kinematic viscosity ν is used. ν is the ratio between the dynamic viscosity and the density of the fluid at a certain temperature (2.5).

$$\nu = \frac{\mu}{\rho} \tag{2.5}$$

In terms of viscosity, fluids are divided in ideal and real fluids. **Ideal fluids** are free of internal friction. This means its viscosity is zero. **Real fluids'** viscosity is above zero. If its viscosity is constant at constant temperature the fluid is called Newtonian fluid.
Newtonian fluids are a special type of real fluids. Their viscosity is linearly proportional to the velocity gradient. **Non-Newtonian fluids** have a non-linear dependence of velocity gradient on the viscosity. Liquid fluids are mostly treated as quasi-incompressible, because their compressibility is negligibly small at pressures lower than 500 bar (Sigloch, 2014).

2.2.2 Flow classification

A fluid flow can be classified in several different ways. In the most general sense, fluid flow can be classified in its dimensions: one-dimensional, two-dimensional or three-dimensional flow.
In relation to hydraulics, fluid flows fall into two main classes: potential flow and eddy flow. **Potential flow** is a frictionless and rotation-free flow. It is mostly used for the simulation of ground water flow. **Eddy flow**, which in turn can be divided in viscid flow (real fluid) and inviscid flow (ideal fluid), includes friction and rotation. Flow in pipes and open channels are eddy flows. In this work, only eddy flows are considered.
Concerning the time-dependency, fluid flow is distinguished in steady-state flow and transient flow. **Transient flow** is dependent on location and time, **stead-state flow** is only location-dependent. In terms of velocity and viscosity, the flow can be laminar or turbulent. The division in laminar and turbulent flow only makes sense for eddy flow of real (viscid) fluids, as the definition of viscosity is a necessity (Sigloch, 2014).

Laminar flow

In a laminar flow the fluid particles move in well-ordered parallel layers without any mixing. These layers can have different velocities. Diffuse transport occurs solely due to thermal molecular motion, the so-called Brownian motion (Sigloch, 2014).

Turbulent flow

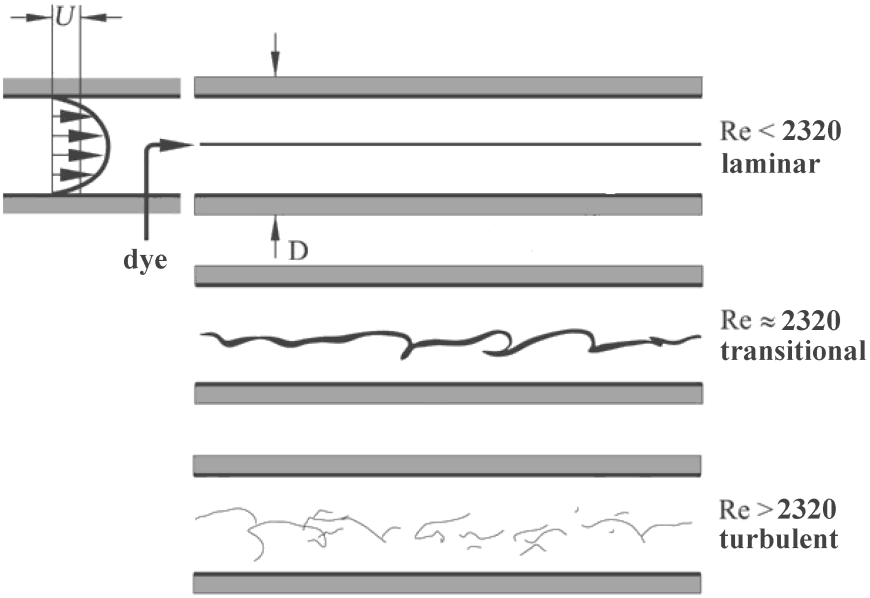
Turbulent flow is characterized by eddies, strong mixing and instability. The macroscopic turbulent motion supervene the fluid transport. The turbulence dissipates into consequently smaller eddy vortices, until it finally dissipates to heat. Without external energy input, the turbulence in a fluid decreases by time (Sigloch, 2014).

Reynolds number

To identify a flow laminar or turbulent, the Reynolds number Re has to be determined. Re is the ratio of inertial forces to viscous forces (2.6) and can be determined as follows.

$$Re = \frac{\bar{u} \cdot L}{\nu} \tag{2.6}$$

where $\bar{u}$ is the mean velocity, L a characteristic length, and ν the kinematic viscosity.
If the Reynolds number or rather the mean velocity exceeds a critical value, the laminar flow makes a transition into a turbulent flow (in the case of constant process set-up and choice of fluid). The critical Reynolds number Re_{crit} was detected by the Reynolds transition experiment, which is a dye experiment after Osborne Reynolds (Fig. 2.1). The experiment shows that the transition from laminar into turbulent flow is not a sharp shift but more a transitional zone (Fig. 2.1 in the middle). The critical Reynolds number can be set at around 2 320 for flow in pipes and open channels (Laurien and jr., 2011).

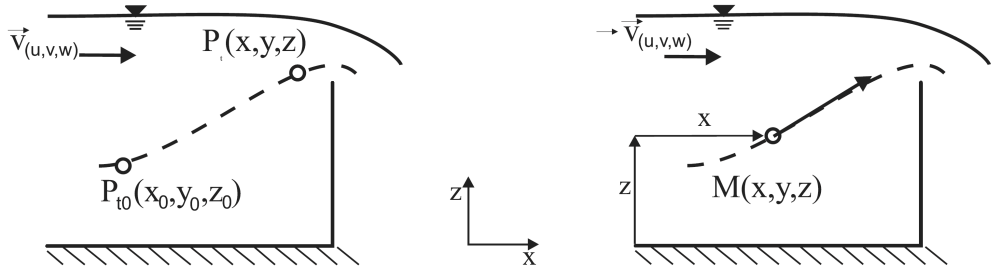


Source: Laurien and jr. (2011), page 33

Figure 2.1: Reynolds transition experiment: by use of a probe, dye is passed into a pipe flow. In case of low Reynolds number, the dye thread stays straight and parallel to the pipe axis (see on top) indicating laminar flow. In case of high Reynolds number, the dye mixes across the flow direction (see on bottom) indicating turbulent flow. At $Re \approx 2\,320$, the straight dye threat becomes unstable (see in the mid figure) indicating the transiotional zone.

2.2.3 Fluid kinematics

Fluid kinematics describe the movement mechanism of the fluid mathematically, without consideration of occurring forces. There are two methods to describe fluid flow. These are the Lagrangian (Fig. 2.2 left) and the Eulerian (Fig. 2.2 right) reference frame.



Source: Bollrich (2000)

Figure 2.2: Lagrangian reference frame (left figure): moving with a particle. Eulerian reference frame (right figure): stationary position.

Lagrangian reference frame

The Lagrangian reference frame describes analytically the pathway of each fluid particle. The quantities like velocity, pressure, density or temperature are linked to one particle and the particle's coordinates which refer to the start coordinates x (2.7), y (2.8), and z (2.9) of the particle (Bestehron, 2006).

$$x = f_1(t, x_0, y_0, z_0) \tag{2.7}$$

$$y = f_2(t, x_0, y_0, z_0) \tag{2.8}$$

$$z = f_3(t, x_0, y_0, z_0) \tag{2.9}$$

Eulerian reference frame

The Eulerian reference frame describes the quantities u (2.10), v (2.11), w (2.12), and p (2.13) of each fixed point inside a defined domain as a function of time and space.

$$u = g_1(t, x, y, z) \tag{2.10}$$

$$v = g_2(t, x, y, z) \tag{2.11}$$

$$w = g_3(t, x, y, z) \tag{2.12}$$

$$p = g_4(t, x, y, z) \tag{2.13}$$

In engineering applications, generally, the velocity and pressure are of interest. The movement of individual particles is of less importance in most engineering problems. Therefore, the Eulerian reference frame is most commonly used in fluid dynamics. In mechanics of point masses, the Lagrangian approach is preferably used (Bestehron, 2006).

2.2.4 Governing equations of fluid dynamics

The governing equations of fluid dynamics are derived from the conservation principles. Generally, these physical laws are valid for all continuums. The equations derived here, are valid for Newtonian fluids.

Material derivative

The material derivative gives the rate of change of a property in time within the volume element. In terms of density, the material derivative is equal to equation (2.14).

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} = \frac{\partial\rho}{\partial t} + \nabla\rho \cdot \vec{u} \tag{2.14}$$

Conservation of mass

The mass of a volume can change only due to incoming or outgoing mass flow. The system itself cannot produce or consume mass. Assuming there are no losses or sources within the volume element, the total mass is constant. Considering the material derivative in the conservation of mass, leads to the continuity equation both in differential (2.15) and integral (2.16) form.

$$\frac{\partial}{\partial t}\,\rho + \nabla\,(\rho \cdot \vec{u}) = 0 \tag{2.15}$$

$$\iiint_V \rho\,\mathrm{d}V + \iint_S \rho \cdot \vec{u}\,\mathrm{d}S = 0 \tag{2.16}$$

Conservation of momentum

Newton's second law of motion (2.17) says that force F is equal to mass m times acceleration a.

$$\vec{F} = m \cdot \vec{a} \tag{2.17}$$

This law can be applied on the momentum $\mathbf{p} = m \cdot \vec{u}$ as well revealing to equation (2.18).

$$\vec{F} = \frac{\mathrm{d}(m \cdot \vec{u})}{\mathrm{d}t} \tag{2.18}$$

Taking into account that the conservation of mass assumption is valid, the equation reads as equation (2.19).

$$\vec{F} = m \cdot \frac{\mathrm{d}\vec{u}}{\mathrm{d}t} \tag{2.19}$$

The forces acting on a volume element are **body forces** like force of gravity or electromagnetic forces as well as **surface forces** like compressive force and friction force, which in term is composed of normal compressive stress and shear stress (Lecheler, 2011).

Incorporating these forces in the mass conservation equations (2.15) and (2.16), leads to the so-called *Cauchy momentum equation* both in differential (2.20) and integral (2.21) form.

$$\overbrace{\rho \cdot \vec{g}}^{gravity} - \overbrace{\nabla\, p \cdot I}^{compression} + \overbrace{\nabla\, \tau}^{friction} = \overbrace{\frac{\partial}{\partial t}\left(\rho \cdot \vec{u}\right) + \nabla\, \left(\rho \cdot \vec{u} \otimes \vec{u}\right)}^{momentum} \tag{2.20}$$

$$\underbrace{\iiint_V \rho \cdot g \,\mathrm{d}x}_{gravity} - \underbrace{\iint_S \nabla\, p \cdot I \,\mathrm{d}S}_{compression} + \underbrace{\iint_S \nabla\, \tau \,\mathrm{d}S}_{friction} = \underbrace{\iiint_V \left[\frac{d}{\mathrm{d}t}\left(\rho \cdot \vec{u}\right) + \nabla\left(\rho \cdot \vec{u} \otimes \vec{u}\right)\right] \mathrm{d}x}_{momentum} \tag{2.21}$$

Conservation of energy

The conservation of energy is not relevant in this study. But for the sake of completeness it is concisely discussed.
The energy conservation (2.22) is known as the first law of thermodynamics.

$$\frac{\mathrm{d}E_{tot}}{\mathrm{d}t} = \dot{W} + \dot{Q} \tag{2.22}$$

This means that the total change in energy E_{tot} inside a volume element is equal to work $\dot{W}$ of the volume element plus heat transfer $\dot{Q}$ into to the element assuming a closed system without consideration of the convective terms.
Total energy is composed of

- Internal energy $E_{in} = m \cdot e$, where e is the specific energy.
- Kinetic energy $E_{kin} = \frac{1}{2}\, m \cdot \vec{u}^2$ with $\vec{u}^2 = (u^2 + v^2 + w^2)$
- Potential energy $E_{pot} = m \cdot g \cdot h$, but which is not considered further in this study.

Work is composed of

- Gravitation g acting on the volume element
- Compression p acting on the element's surface
- Normal compressive stress and shear stress τ acting on the element's surface

Heat transfer is composed of

- Heat radiation $\dot{q}_r$ onto the element
- Heat conduction $\dot{q}_k$ due to temperature gradients
- Heat convection $\dot{q}_c$ acting on the element's surface, which is not considered here since the element is moving with the flow.

Inserting these components yields to equation (2.23).

$$\frac{\partial}{\partial t}\left(\rho \cdot e + \frac{1}{2}\,\rho \cdot \vec{u}^2\right) + \nabla\left[\rho \cdot \vec{u}\left(h + \frac{1}{2} \cdot \vec{u}^2\right) - \tau \cdot \vec{u} - \dot{q}_k\right] = \rho \cdot \vec{g} \cdot \vec{u} + \rho \cdot \dot{q}_r \tag{2.23}$$

2.2.5 Navier-Stokes equations

The equations of conservation of mass, momentum and energy are called Navier-Stokes equations. It is a system of non-linear differential equations, which can only be solved analytically for special cases. A more general approach, is solving the Navier-Stokes system numerically. The Navier-Stokes equations are written in divergence form in equation (2.24) (Lecheler, 2011).

$$\frac{\partial}{\partial t}\begin{bmatrix} \rho \\ \rho \cdot \vec{u} \\ \rho \cdot (e + \frac{1}{2} \cdot \vec{u}^2) \end{bmatrix} + \nabla \begin{bmatrix} \rho \cdot \vec{u} \\ \rho \cdot \vec{u} \otimes \vec{u} + p \cdot I - \tau \\ \rho \cdot \vec{u}(h + \frac{1}{2} \cdot \vec{u}^2) - \tau \cdot \vec{u} - \dot{q}_k \end{bmatrix} = \begin{bmatrix} 0 \\ \rho \cdot \vec{g} \\ \rho \cdot \vec{g} \cdot \vec{u} + \rho \cdot \dot{q}_r \end{bmatrix} \tag{2.24}$$

The conservation laws are not sufficient to arrive at a solution since there are more unknowns than equations. Additional equations, so-called constitutive relations are needed depending on the fluid under study.

Constitutive relations

The following constitutive relations are valid for Newtonian fluids since these fluids are of interest in this study. Some of these relations are already included in the derived Navier-Stokes equation stated above. Additionally, conditions for incompressible and isothermal fluids are given.

The thermodynamic equation of state (2.25) relates the pressure with density and temperature.

$$p = f\,(\rho,\, T) \tag{2.25}$$

The caloric equation of state (2.26)relates the internal energy or, more specifically, the enthalpy $h = e + \frac{p}{\rho}$ with pressure and temperature.

$$h = f\,(p,\, T) \tag{2.26}$$

As previously mentioned, it can be assumed that the total energy consists solely of internal energy and kinetic energy.

Fourier's law of heat conduction (2.27) states that the rate of heat conduction $\dot{q}_k$ is proportional to the negative temperature gradient ∇T.

$$\dot{q_k} = -\lambda \nabla T \tag{2.27}$$

where λ is the heat conductivity of the material.

Based on Newton's law of viscosity, **Stokes's stress relations** link the stress with the velocity. The elements τ_{ii} (2.28) represent the normal stresses which cause contraction or extension. The elements τ_{ij} (2.29) represent the shear stresses which cause shear strain. μ is the dynamic viscosity (Lecheler, 2011).

$$\tau_{ii} = -\frac{2}{3} \cdot \mu \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} \right) + 2 \cdot \mu \left(\frac{\partial u_i}{\partial i} \right) \tag{2.28}$$

$$\tau_{ij} = \mu \left(\frac{\partial u_j}{\partial i} + \frac{\partial u_i}{\partial j} \right) \tag{2.29}$$

The transport coefficients, which are the dynamic viscosity μ (2.30) and the heat conductivity λ (2.31), are also functions of pressure and temperature.

$$\mu = f\,(p,\, T) \tag{2.30}$$

$$\lambda = f\,(p,\, T) \tag{2.31}$$

Incompressible fluid's density ρ remains constant, thus, its derivative in time (2.32) is equal to zero.

$$\frac{\partial \rho}{\partial t} = 0 \tag{2.32}$$

Isothermal flow remains at a constant temperature, thus, the heat conductivity λ (2.33) is infinite.

$$\lambda = \infty \tag{2.33}$$

Concerning the constitutive relations and incompressible, isothermal conditions of Newtonian fluids, the mass balance equation (2.15) and the *Cauchy momentum equation* (2.20) can be simplified to equation (2.34) and 2.35) respectively (Jasak, 1996).

$$\nabla\, \vec{u} = 0 \tag{2.34}$$

$$\frac{\partial \vec{u}}{\partial t} + \nabla\, (\vec{u} \otimes \vec{u}) + \nabla\, p = \vec{g} + \nabla\, (\nu \cdot \nabla \vec{u}) \tag{2.35}$$

2.2.6 Turbulence modelling and simulation

By means of the Navier-Stokes equations, laminar and turbulent fluid flow can be fully described. But the calculation of turbulent flow efforts extreme high computational power due to the non-linearity of the equations. A vast amount of grid points would be necessary to solve the smallest turbulence (Lecheler, 2011). Therefore, turbulence models and simulation methods were developed to model turbulence flow at an acceptable computational load.

Reynolds-averaged Navier-Stokes model

The Reynolds-averaged Navier-Stokes (RANS) model is an averaged form of the Navier-Stokes equations. It predicts the mean flow of fluids but with significant error, and it gives no information on turbulent fluctuations. Even though, it is widely used in engineering analysis due to its relatively low computational costs.

The Reynolds-averaged model applies the Navier-Stokes equations but solely concerns the time-averaged values. Each state variable, e.g. the velocity (2.36), can be conceived as the sum of a time-averaged value $\bar{u}$ and a fluctuation value u'.

$$u = \bar{u} + u' \tag{2.36}$$

The time-averaged values represent the smooth mean flow and cover the low frequency, time-dependent variation of the flow. The fluctuation values represent the high frequency, turbulent fluctuations. This means that the Reynolds-averaged Navier-Stokes equations describe the large-scale component of the flow. The turbulent fluctuations influencing the mean flow significantly are described by turbulence models (Laurien and jr., 2011).

By means of turbulence models, an approximate solution of the turbulent fluctuations can be estimated. In RANS simulations, eddy viscosity models are applied. The eddy viscosity hypotheses states that the turbulent transport is dependent on the mean flow velocity gradients. That implies that the turbulent fluctuations can be described by the eddy viscosity μ_t (2.37) (Zikanov, 2010).

$$\mu_t = C_\mu \cdot \rho \cdot q \cdot l \tag{2.37}$$

$$with \qquad q = \left(\frac{2}{3} \cdot k\right)^{1/2} \tag{2.38}$$

where C_μ is a dimensionless proportionality constant, k is the kinetic energy of fluctuation, and l is a characteristic length within the flow. μ_t is determined by algebraic or two-equations models.

Algebraic models determine k and l by algebraic functions with the mean flow quantities as input. The Baldwin-Lomax model is a well-known algebraic model. It is simple and computational efficiently, but only applicable to parallel shear flows (Zikanov, 2010).

Two-equations models are theoretically applicable to all types of flow. It determines k and l by two additional differential equations (Zikanov, 2010). The most common two-equations models are the $k-\epsilon$ and the $k-\omega$ model. The $k-\epsilon$ model applied in this study is discussed in detail in chapter 3: *Materials and Methods*.

Large eddy simulation

The large eddy simulation (LES) calculates the mean flow as well as the unsteady large-scale, and intermediate-scale components of fluid flow directly by the Navier-Stokes equations. The small-scale fluctuations are described by turbulence models leading to an approximated solution. This simulation method is based on the concept of filtering. A low-pass filter cuts out the turbulent fluctuations smaller than the filter width. The mean flow and fluctuations larger than the filter width are calculated numerically. The fluctuations smaller than the filter width are approximated by closure models. It is sufficient that the grid size is equal to the filter width.
The error of this method is significantly smaller than of the RANS model, but its computational cost is much higher. Due to increasing computational power, the application of large eddy simulation is expanding (Zikanov, 2010).

Direct numerical simulation

The direct numerical simulation (DNS) is preferably used in fundamental studies on turbulence. In case of **homogeneous turbulence**, the spectral method is the most qualified numerical method to apply in DNS. Homogeneous turbulent flow can be assumed considering a small zone far from walls. Flow with realistic boundaries generates inhomogeneous turbulence. **Inhomogeneous turbulence** can be solved numerically by finite difference method or finite volume method. Nonetheless, DNS is restricted to flow with simple domains and low Reynolds numbers if its computational costs need to be kept at an acceptable load (Zikanov, 2010).

2.3 Drinking water and reservoirs

> *"Water is critical for sustainable development, including environmental integrity and the eradication of poverty and hunger, and is indispensable for human health and well-being."*
>
> *(United Nations, A/RES/58/217 (UN, 2004))*

2.3.1 Water on earth and Europe's water policy

Water is the most important resource on earth, as it is needed in all aspects of life. Without sufficiently clean water, viability is impossible for all living organisms on this earth.
When determining the total quantity of water on earth, $1\,600\,million\,km^3$, it seems that the availability of water is no central issue in international politics today. But on closer inspection, the amount of fresh water is only 0.5% of all water on earth, and most of it is situated over 800 m deep in the ground (Table 2.1). This is aggravated by the fact that the fresh water is unevenly distributed over the earth (de Moel et al., 2006).
In view of these issues, the European Union developed the EU Water Framework Directive (WFD) being operative since 2000. The WFD sets out the environmental objectives for Europe's water resources focused on inland surface waters, transitional waters, coastal waters and groundwater. The key aims of the Water Framework Directive are (Parlament, 2000):

- expanding the scope of water protection to all waters, surface waters and groundwater
- achieving "good status" for all waters by a set deadline
- water management based on river basins
- "combined approach" of emission limit values and quality standards
- getting the prices right
- getting the citizen involved more closely
- streamlining legislation

The protection of all waters achieving good water quality is a substantial contribution to ensure the drinking water supply for the population. Particularly the pollution of water bodies used for the abstraction of drinking water is to be avoided in order to reduce the costs of drinking water treatment (Parlament, 2000).

Table 2.1: Water on earth: quantities and occurence of different types of water

Type of water	Quantity $million\,km^3$	Occurence in the ground $depth\,in\,m$
Total quantity of water	1 600	3 100
chemically bound	230	450
salt water	1 300	2 610
damp	0.015	0.03
Total quantity of fresh water	8.2	16.1
in the underground	8	15.7
in lakes and rivers	0.2	0.4

Source: de Moel et al. (2006), page 213

2.3.2 Drinking water

Drinking water is a valuable food and must satisfy certain quality criteria. It is qualified for human consumption if it does not jeopardise human health, and its smell, taste and colour is unobjectionable. In the European Union's *Council Directive 98/83/EC of 3 November 1998 on the quality of water intended for human consumption* (Council, 1998), as well as in the *European standards for drinking water* (WHO, 1970), the microbiological, chemical and indicator parameters and its values to be complied with are specified .

Water provided for human consumption can originate from groundwater, spring water, surface water or precipitable water. Groundwater and spring water are usually better in quality and therefore should be preferred for drinking water production. However, in regions of intensive agriculture, the rainwater draining into the soil transports the harmful fertiliser and pesticides into the aquifer. High nitration rates and pesticide levels in the groundwater are the consequences (de Moel et al., 2006).

In the event of lack of groundwater, surface water is necessarily used for the drinking water production. River water is affected by many environmental impacts. From its origin to its outfall, a river passes inhabited regions, industrial zones, farmland and is maybe used for shipping. Its contamination ranges from domestic sewage and industrial waste water, to fertiliser and pesticides residues. A major uprising and not completely understood problem is hormone inputs from human and agriculture changing the rivers' biologic. The high content of pollutants necessitates an elaborate and cost-intensive water treatment. During growing season accompanied by heavy use of fertiliser and pesticides, it is sometimes the case that

the waste load reaches a certain level that drinking water treatment is no longer efficiently feasible. If this event occurs yearly over a longer period, a reservoir should be installed enabling a continuous drinking water production all year round. In times of excess water, the reservoir is filled up. In times of water shortage, the water stored in the reservoir can be used to satisfy the drinking water demand (de Moel et al., 2006).
The water production center dealt with in this study has to cope with such difficulties. The reservoir installed has a capacity of $3\,million\,m^3$ allowing a water production lowered to 50 % during the period of water shortage extending from June to August.

Surface water captured from lakes is usually of better quality than from rivers, because it is mostly enclosed by nature reserve. But its use for drinking water production is strictly limited to preserve the lake's ecosystem.

2.3.3 Reservoirs

A reservoir is a man-made form of surface water and is often defined as an artificial lake. It is built either by a surrounding dike or by a dam. Reservoirs enclosed by a dam are also called dam lakes and are widely constructed in mountain regions.
The major function of a reservoir is to decrease the fluctuation in the river's flow, enabling a steady water abstraction for either drinking water production, irrigation or energy production. In terms of drinking water production, another main purpose is to increase the natural process of self-purification. With respect to the river's flow characteristic, different types of reservoirs with their specific functions are installed. These functions are: storage, self-purification, decay of micro-organisms, and levelling (de Moel et al., 2006).

Storage

Water storage enables a continuous drinking water production all year round. During times of low river discharge or bad water quality, the raw water requirements are (partly) covered by storage water. Storage reservoirs usually have level fluctuations. But by means of storage reservoirs, the water level of reservoirs connected in series can be stabilised (de Moel et al., 2006).

Self-purification

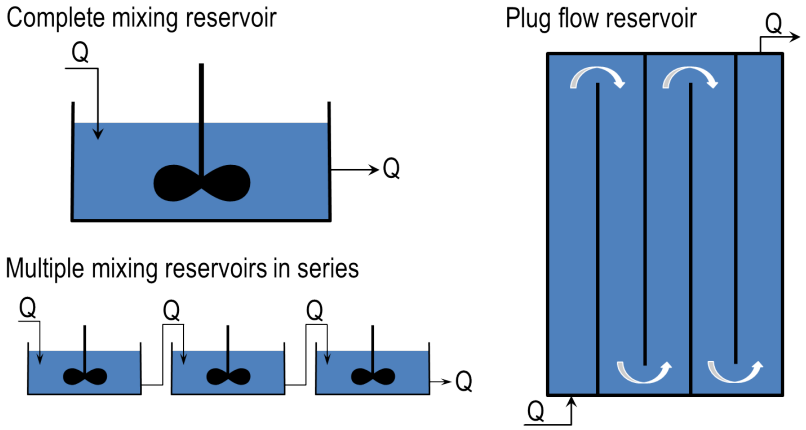
Self-purification is a natural process improving the water quality by particle settling, conversion of organic matter, decreasing water temperature, removal of micro-organisms and concentration damping. All these processes are turbulence- and time-dependent, and therefore, the residence time is decisive for the rate of self-purification (Levenspiel, 1999).

Plug flow and decay of micro-organisms

In a plug flow, the water particles move concentrated through an elongated system without mixing with other water particles (Fig. 2.3 right). In turbulent flow conditions, all water particles have the same retention time, but their concentration varies. This means that plug flow is very efficient for the decay of bacteria. Decay and mortality of micro-organisms in plug flow is always higher than in a mixed system. Disadvantages are its high construction costs (Levenspiel, 1999).

Mixing and levelling

A lake is usually a mixed system with large differences in its residence times. In an ideally mixed system, the water inflow completely mixes with the lake water as soon as it enters the lake (Fig. 2.3 top left). That implies that the retention times of the particles vary, but the concentration of the particles is uniform throughout the system. This means that mixed systems are less effective for removal of micro-organisms than plug flows, but much more effective for concentration levelling. Water abstracted from rivers is subjected to large variations in concentration. In case of concentration peaks exceeding the limits, abstraction is banned bringing the drinking water production to a standstill. Due to mixing of high-concentrated water inflow with low-concentrated lake water, concentration peaks are shaved off in the reservoir. With respect to the maximum concentration occurring, a certain size of the reservoir is required allowing the water abstraction all year round.
In order to limit the construction costs, multiple mixing reservoirs are constructed in series (Fig. 2.3 bottom left) creating a system in between complete mixing and plug flow. With the increasing number of reservoirs, the variation in residence time decreases while the mortality increases (Levenspiel, 1999).



Source: own figure (compare de Moel et al. (2006), page 330 ff.)

Figure 2.3: Schematic illustrations of a complete mixing reservoir (top left), multiple mixing reservoirs in series (bottom left), and a plug flow reservoir (right).

Shallow reservoirs

Since the reservoir under investigation is a shallow basin, particular problems and requirements of shallow reservoirs are finally reviewed.

In shallow reservoirs, the water composition is almost uniform over the depth. Only near the water surface and near the bottom, the rate of suspended solids is slightly higher. If the access for humans or animals is provided or not preventable, the water pollution will be greater near the shore than towards the center.
For these reasons, the water outlet is preferably located towards the middle of the reservoir and with some distance to the bottom avoiding abstraction of sludge. Mostly, an intake tower is constructed, which is connected to the pumping station by a conduit or a hinged pipe. If an intake tower represents a barrier for boats, the intake tower can be omitted installing a cover right atop the out-take (de Moel et al., 2006).

The main problem of shallow reservoirs is the significant development of algae during spring and summer. Algae needs water, carbon dioxide, nutrients and energy from sunlight to produce organic matter. In spring, the rising temperatures increase the water temperature, particularly on the top, initiating the growth of algae. The addition of nutrients, including nitrogen, sulphur and phosphorus through waste water, fertilizers, or detergents, precipitate the growth of algae.
In case of deep reservoirs, algal blooms occurs only in the upper layer where light can penetrate. At a depth greater than 7.0 m to 10.0 m, the water temperature remains lower and the sunlight cannot enter averting algae development within that region. By continuous mixing of the layers, stratification can be prevented and thus, algae growth is restrained. This means that in the case of deep reservoirs, the algae population can be handled by continuous circulation of the water.
In case of shallow reservoirs less than 7.0 m to 10.0 m deep, the entire water body is sundrenched and warms up. A cool bottom layer is not available to mix with the warm top layer decreasing its temperature. This means that the algae growth cannot be limited by circulation. As a consequence, the water becomes turbid and turns green, brown or red. At worst, algae generates objectionable odour and taste. In this particular case, the only way to restrict the algae population is removing one of the nutrients from the water.
If the growth of algae is not going to be regulated, micro strainers need to be installed as a first treatment process. In addition to hydrochloride for disinfection, and a flocculant for coagulation of suspended solids, powdered activated carbon needs to be added removing the micro pollutants caused by algal bloom. Due to the additional treatment steps required, the costs of drinking water production increase significantly (de Moel et al., 2006).

CHAPTER 3

Materials and Methods

3.1 Introduction

In this chapter, the methods applied in this study are presented. Next, the case study and the conducted scenarios are described in detail. Subsequently, model simplifications and assumptions are discussed, and the boundary and initial conditions are determined. Finally, the software used throughout this work is briefly reviewed.

3.2 Computational fluid dynamics (CFD)

> *"CFD is a set of numerical methods applied to obtain approximate solutions of problems of fluid dynamics and heat transfer."*
>
> *(Oleg Zikanov (2010))*

Since the problems of fluid dynamics are very complex, an analytical solution of the governing equations of fluid dynamics only exists for very simple applications (Lecheler, 2011). The equations for real fluid flow within real boundary conditions are non-linear partial differential equations, which have to be solved numerically.

3.2.1 Numerical discretisation

Numerical discretisation is the substitution of partial differential equations in a continuum domain with algebraic equations in a discrete domain. Several methods of numerical discretisation are available in literature. In fluid dynamics, commonly used techniques are the finite difference method and the finite volume method. Other methods are the spectral method mostly applied in fundamental studies on turbulence, and the finite element method, primarily used in structural analysis (Zikanov, 2010).

In the following sections, the numerical discretisation is mathematically explained by means of the finite difference method. It is also applied for temporal discretisation in the finite volume method. The spectral and the finite element method are shortly introduced to show their main differences to the finite difference and the finite volume method. Finally, the finite volume method applied in this study will be discussed more in detail.

Finite difference method

The finite difference method transforms the partial derivatives $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ by finite differences $\left(\frac{\Delta}{\Delta x}, \frac{\Delta}{\Delta y}, \frac{\Delta}{\Delta z}\right)$ using a truncated Taylor series. To achieve this, the domain is subdivided by a computational grid, which consists of spatial points $(x, y, z)_i$ and time layers t^n, where the finite differences are separately applied to.
These are three commonly used schemes to convert the partial derivatives, e.g. $\left(\frac{\partial A}{\partial x}\right)_{i,j}$, where A is an arbitrary state variable, to finite differences: forward difference of first order (3.1), backward difference of first order (3.2), and central difference of second order (3.3) of approximation (Lecheler, 2011).

$$\left(\frac{\partial A}{\partial x}\right)_{i,j} = \frac{A_{i+1,j} - A_{i,j}}{\Delta x} + O(\Delta x) \tag{3.1}$$

$$\left(\frac{\partial A}{\partial x}\right)_{i,j} = \frac{A_{i,j} - A_{i-1,j}}{\Delta x} + O(\Delta x) \tag{3.2}$$

$$\left(\frac{\partial A}{\partial x}\right)_{i,j} = \frac{A_{i+1,j} - A_{i-1,j}}{2 \cdot \Delta x} + O(\Delta x)^2 \tag{3.3}$$

where $O(\Delta x)$ is the truncation error. The temporal discretisation can be conducted in the same way, see section *Finite volume method*.

Spectral method

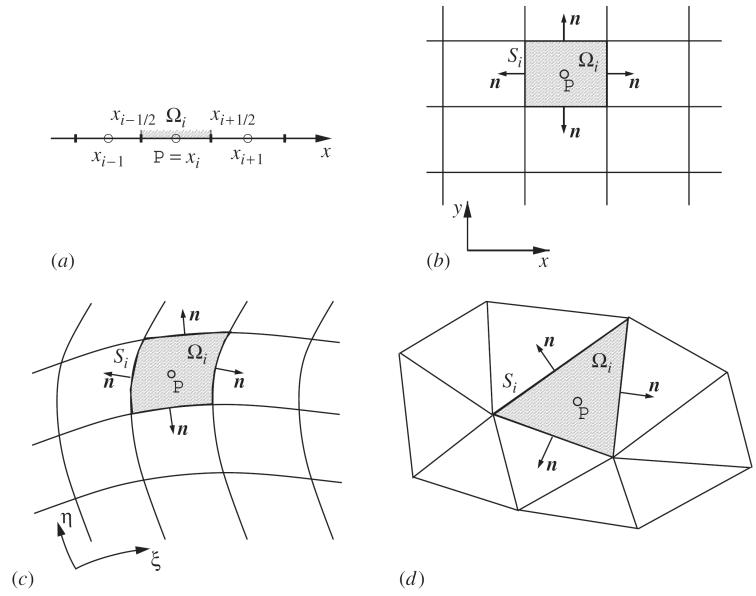
The spectral method approximates the non-linear partial differential equations by a finite series of linearly independent trial functions. The error of approximation decreases with increasing number of trial functions. The trial functions are ordinary differential equations, which can be solved numerically. With a moderate number of trial functions, the spectral method leads to results of good accuracy. However, its application is restricted to flow with simple domains. Therefore, it is preferably used in fundamental studies on turbulence (Zikanov, 2010).

Finite element method

The finite element method makes use of trial functions similar to the spectral method. The main difference is, that the domain is divided into a finite number of elements to which the set of trail functions is separately applied. Thus, this method does apply to complex geometries. In comparison to the finite volume method, the finite element method is not exactly conservative, but is a weak form of volumetric integrals. The finite volume method is an exact conservation statement expressed by surface integrals (Zikanov, 2010).

Finite volume method

The finite volume method is comprised of a spatial discretisation, an equation discretisation, and a temporal discretisation.
In terms of **the spatial discretisation**, the solution domain is subdivided into small, finite control volumes, called cells. The cells are non-overlapped, share their faces with their neighbouring cells, and completely fill the solution domain. The result is a finite volume grid, i.e. the computational mesh, where the integral equations are solved subsequently for each control volume. Inside the mesh, internal and boundary faces are defined. Internal faces connect two cells and belong to an owner cell and a neighbour cell. Faces coinciding with the boundary of the domain are defined as boundary faces and only belong to one cell. Furthermore, a distinction is made between the cell-centred method where the properties are stored at the cell centroids, and the cell-vertex method where the properties are stored at the vertexes of the grid. In the latter, the control volumes are built around the vertexes (Jasak, 1996).
There are different types of computational grids available. Figure 3.1 shows four examples of grids: an one-dimensional grid (a), a Cartesian structured grid (b), a curvilinear structured grid (c), and an unstructured grid (d). A structured grid consists of quadrilateral (2D) or hexahedral (3D) cells aligned to a Cartesian or curvilinear coordinate system. Unstructured grids are composed of various shapes such as triangles or quadrilaterals (2D), and tetrahedrons, hexahedrons, or pyramids (3D) (Zikanov, 2010).
The accuracy of the result depends amongst others on the grid quality. The most accurate results are achieved by orthogonal structured grids. Distorted structured grids with angles smaller than 45° and larger than 135° will not provide adequate results and are not recommended. It is also noted that the ratio of adjacent cell sizes should not be larger than two. The transition from a fine grid to a course grid should be always gradual. In some cases it is useful to subdivide the geometry into zones of simple shapes, so-called blocks, where each of them has a structured grid. Particular attention shall be given to the matching conditions at the interfaces of the blocks. In case of complex geometries, unstructured grids can be used, or "snapping" a complex surface with hexahedrals is also possible. Unstructured grids generally requires more computational power (Lecheler, 2011).



Source: Zikanov (2010), page 88

Figure 3.1: Examples of finite volume grids: (a) one-dimensional grid, (b) two-dimensional Cartesian structured grid, (c) two-dimensional curvilinear structured grid, (d) two-dimensional unstructured grid.

The equation discretisation is applied to the integral form of the governing equations. The transport equation for a scalar property ϕ is shown in integral form (3.4), where CV is the control volume, Γ the diffusion constant, and Q_ϕ sources of ϕ. The convective and diffusive flux at the faces of each cell are explicitly determined retaining the local and global balance (Schwarze, 2013).

$$\underbrace{\frac{\partial}{\partial t}\int_{CV}\rho\,\phi\,\mathrm{d}V}_{temporal derivative} = \underbrace{\int_{CV}\nabla\left(\rho\,\Gamma\,\nabla\phi\right)\mathrm{d}V}_{diffusion} - \underbrace{\int_{CV}\nabla\left(\rho\,\vec{u}\,\phi\right)\mathrm{d}V}_{convection} + \underbrace{\int_{CV}Q_\phi\,\mathrm{d}V}_{source} \tag{3.4}$$

Considering the Gauss' theorem, the diffusion and the convection term can be expressed by surface integrals (3.5), where S is the closed surface of the control volume CV, and $\hat{n}$ is the outward facing normal to the infinitesimal surface $\mathrm{d}A$.

$$\underbrace{\frac{\partial}{\partial t}\int_{CV}\rho\,\phi\,\mathrm{d}V}_{temporal derivative} = \underbrace{\oint_S\hat{n}\left(\rho\,\Gamma\,\nabla\phi\right)\mathrm{d}A}_{diffusion} - \underbrace{\oint_S\hat{n}\left(\rho\,\vec{u}\,\phi\right)\mathrm{d}V}_{convection} + \underbrace{\int_{CV}Q_\phi\,\mathrm{d}V}_{source} \tag{3.5}$$

Regarding incompressible flow without internal sources, equation (3.5) reduces to (3.6).

$$0 = \oint_S\hat{n}\left(\rho\,\Gamma\,\nabla\phi\right)\mathrm{d}A - \oint_S\hat{n}\left(\rho\,\vec{u}\,\phi\right)\mathrm{d}V \tag{3.6}$$

The integral equation is approximated by means of discretisation schemes. There are many interpolation and differentiation schemes available. Three commonly used schemes of first and second order are: the upwind differencing (UD), the central differencing (CD), and the bounded (blended) differencing (BD) scheme which is a combination of UD and CD. Well-known high-order schemes are the quadratic interpolation for convective kinematics (QUICK), or the total variation diminishing (TVD) scheme (Schwarze, 2013). Generally, a spatial approximation of second order is sufficiently accurate in engineering applications (Lecheler, 2011). To stay within the limits of this study, there is no further discussion on differentiation schemes.

The temporal discretisation is done by subdivision of the time interval into small, finite time-steps. This can be performed in the same way as the finite difference method. Equation (3.7) is a temporal central difference of second order of $\left(\frac{\partial A}{\partial t}\right)_i^n$, where A is an arbitrary state variable. Temporal discretisation is necessary in case of transient flow conditions where an accuracy of second order is required. Steady state calculations are not temporal dependent (Lecheler, 2011).

$$\left(\frac{\partial A}{\partial t}\right)_i^n = \frac{A_i^{n+1} - A_i^{n-1}}{2 \cdot \Delta t} + O(\Delta t)^2 \tag{3.7}$$

In terms of temporal discretisation, a distinction is made between explicit (3.8) and implicit schemes (3.9). Equation (3.8) and (3.9) are one-dimensional difference equations without source terms, where A is any conserved quantity and B any flow term (Lecheler, 2011)).

$$A_i^{n+1} = A_i^n - \frac{\Delta t}{2 \cdot \Delta x} \cdot \left(B_{i+1}^n - B_{i-1}^n\right) \tag{3.8}$$

$$A_i^{n+1} + \frac{\Delta t}{2 \cdot \Delta x} \cdot \left(B_{i+1}^{n+1} - B_{i-1}^{n+1}\right) = A_i^n \tag{3.9}$$

In the explicit scheme, the flow terms B are determined at the time of $t = n$. Thus, A_i^{n+1} at $t = n + 1$ can be solved explicitly. In the implicit scheme, B is determined at the unknown time of $t = n + 1$. Thus, B_{i+1}^{n+1} and B_{i-1}^{n+1} are unknown and A_i^{n+1} at $t = n + 1$ cannot be solved directly. Therefore, a equation system including all grid points is formulated. Using the central spatial discretisation, the equation system is a tridiagonal matrix. In order to solve this equation system, the equations are linearised, and then solved iteratively by means of relaxation techniques (Lecheler, 2011).

Pressure-velocity coupling

The discretised form of the Navier-Stokes equations requires a special treatment due to the linear dependence of the velocity on pressure and conversely (Jasak, 1996). In order to deal with this pressure-velocity coupling, different methods were developed for transient and steady-state conditions. Most commonly used methods are the pressure-implicit split-operator (PISO), and the semi-implicit method for pressure-linked equations (SIMPLE) algorithms, which are briefly introduced next.

The pressure-implicit split-operator (PISO) is an iterative method to solve equations for velocity and pressure of transient problems. Using the PISO mode, the time step is strictly limited fulfilling the *Courant* criterion (CFL ¡ 1). Thus, solving a real time problem with transient conditions can be computationally expensive, especially in very complex geometries (Jasak, 1996).

The semi-implicit method for pressure-linked equations (SIMPLE) algorithm is developed for steady-state problems and couples the Navier-Stokes equations with an iterative procedure. It is developed to reach steady-state conditions very fast by means of relaxation factors. Since the time derivation is not concerned, the SIMPLE algorithm does not provide time information (Jasak, 1996).

The PIMPLE algorithm is a merged PISO-SIMPLE algorithm using the advantage of both the SIMPLE method (relaxation factors) and the PISO method (multiple momentum correctors). This means that the solution is converged at each time step to a certain limit. In this way, the PIMPLE algorithm is very effective in large cases to increase the CFL number, and thus, larger time-steps can be used speeding up the simulation to reach a stable solution (Jasak, 1996).

Numerical stability

A numerical system becomes unstable if the truncation error (which is neglected in the numerical solution) becomes too large. Therefore, certain stability criteria need to be satisfied. In terms of **discretisation errors**, the Courant-Friedrichs-Levy (CFL) condition is a necessary stability condition, but not a sufficient one (Courant et al., 1928). CFL is determined by equation (3.10). In case of explicit schemes, CFL is restricted to 1. This means that the number of time steps per iteration needs to be small to guarantee stability. Thus, many time steps are necessary in order to obtain a converged solution. In implicit schemes, CFL can be significantly larger than 1 (10-1000 depending on the case). Thus, larger time steps can be used reducing the total number of time steps needed until the solution has converged.

In this case, the accuracy of numerical approximation depends on Δx and order of spatial approximation, but not on Δt. This means that Δt can become even smaller if the grid needs to be further refined (smaller Δx).

$$CFL = u \cdot \frac{\Delta t}{\Delta x} \tag{3.10}$$

$$CFL = \begin{cases} \leq 1 & \text{for explicit scheme} \\ \gg 1 & \text{for implicit scheme} \end{cases}$$

In terms of **round-off errors** ϵ, a weak stability (3.11), and a strong stability (3.12) is analysed. The stability can be analysed by the Neumann method based on Fourier expansions, but it is not used in practice as it is too time consuming (Zikanov, 2010).

$$weak\ stability \qquad \left| \frac{\epsilon_i^{n+1}}{\epsilon_i^n} \right| \leq 1 \tag{3.11}$$

$$strong\ stability \qquad \frac{\|\epsilon_i^{n+1}\|}{\|\epsilon_i^n\|} \leq 1 \tag{3.12}$$

3.2.2 Turbulence modelling

The governing equations of fluid dynamics and the Reynolds-averaged Navier-Stokes model were already reviewed in section 2.2. To recapitulate, turbulence models prescribe a relationship between the Reynolds stress and the mean velocity gradient by means of the eddy viscosity, see equation (2.37). In a two-equation turbulence model, the eddy viscosity μ_t is expressed as a function of the turbulent kinetic energy k and its dissipation rate ε.

In this study, the standard $k-\varepsilon$ turbulence model by Launder and Spalding (1974) is applied, which is discussed in the next section.

k-epsilon turbulence model

The equations of the $k-\varepsilon$ turbulence model are the transport equations for the turbulent kinetic energy k and its dissipation rate ε. k is defined in equation (3.13), where u', v', and w' are the turbulent fluctuations in x-, y-, and z-direction respectively.

$$k = \frac{1}{2} \cdot \left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2} \right) \tag{3.13}$$

When assuming isotropic conditions, equation (3.13) reduces to (3.14).

$$k = \frac{3}{2} \cdot \left(\overline{u'^{\,2}}\right) \tag{3.14}$$

where the term $\left(\overline{u'^{\,2}}\right)$ is the Reynolds stress tensor, which is approximated by means of the averaged velocity $\bar{u}$ and the turbulence intensity I (3.15). I is also referred to as turbulence level.

$$u' = \bar{u} \cdot I \tag{3.15}$$

The dissipation ratio ε is determined by means of the turbulent kinetic energy k, and a turbulence length scale l_T which represents the length scale of the largest fluctuations (3.16).

$$\varepsilon \approx \frac{k^{3/2}}{l_T} \tag{3.16}$$

Finally, the eddy viscosity can be determined (3.17).

$$\mu_t = C_\mu \cdot \rho \cdot \frac{k^2}{\varepsilon} \tag{3.17}$$

where ρ is the fluid density, and C_μ an empirical model constant.

3.2.3 Transport modelling

Based on a transport model, a virtual tracer test can be conducted analysing the advective transport and the residence time throughout the reservoir. A passive scalar not influencing the fluid flow is assumed, which is valid for small molecules as only large chunks or polymers will affect fluid flow. The governing equation for scalar transport modelling is the advection-diffusion equation. Assuming a non-degradable tracer and no sources, the molecular diffusion term and the source term are negligible. With respect to the Reynolds stress hypotheses, the advection-diffusion equation can be written in following form (3.18) (Angeloudis, 2014).

$$\frac{\partial c_T}{\partial t} + \vec{u}_j \frac{\partial c_T}{\partial x_j} = \frac{\partial}{\partial x_j} \left(-\overline{u'_l \, c_t}\right) \tag{3.18}$$

where c_T is the scalar quantity of the tracer concentration, and the term $\left(-\overline{u'_l \, c_t}\right)$ represents the turbulent diffusion relative to the turbulent fluctuation u', which can be approximated by (3.19).

$$\left(-\overline{u'_l \, c_t}\right) = D_t \frac{\partial c_T}{\partial x_j} \tag{3.19}$$

where D_t is the turbulent mass diffusivity which is dependent on the turbulence intensity.

3.2.4 Boundary conditions

Since the CFD model is a finite domain, appropriate conditions at cells adjacent to the domain boundaries need to be defined for the validity of the numerical simulation. In the present study, numerical boundary conditions are identified for the consideration of inlet, outlet, and solid wall effects. The surface areas "front and back" are empty in 2D, thus, no specification is required. With respect to the accuracy of the $k-\varepsilon$ turbulence model, empirical wall-functions are necessarily applied near solid surfaces. All boundary conditions used in this study are summarised in Table 3.1.

Dirichlet boundary

The Dirichlet boundary condition is the simplest boundary condition type, where a **fixed value** is specified on the boundary of the domain. It is applied on the inflow boundary condition with the mean velocity over the entire inlet area, and impermeable wall boundaries with no-slip conditions. No-slip condition means that the velocity of the fluid on the wall is equal to that of the wall itself, which is zero in this case. At the outlet boundaries, the pressure field is defined as a Dirichlet boundary with zero pressure field (Jasak, 1996).

Neumann's boundary

The Neumann's boundary condition signifies a **fixed gradient** boundary condition, where the derivative of a solution is set on the domain boundary. This boundary condition is employed on the outflow surfaces with zero gradient for velocity, k and ε, and on the inflow surfaces with zero pressure gradient. As there is no flux through the solid wall, the pressure gradient is zero at wall boundary faces (Jasak, 1996).

Wall-function boundary

In case of viscid fluid flow, a higher resolution grid for the viscous sub-layer would be needed increasing the computational time. Instead, empirical **wall functions** are implemented for cells adjacent to solid walls in order to avoid the increase of computational expense. Thereby, it is assumed that the flow near solid walls behaves like a fully developed turbulent boundary layer, which consequently lowers the accuracy (Jasak, 1996).

Table 3.1: Overview of the different boundary conditions typically used for CFD

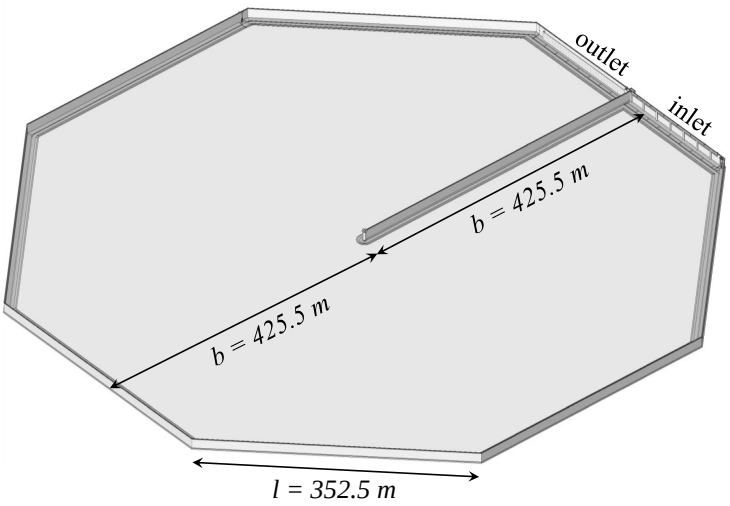
patch type	velocity	pressure	turbulent energy	dissipation ratio
inlet	fixed value	fixed gradient	fixed value	fixed value
outlet	fixed gradient	fixed value	fixed gradient	fixed gradient
wall	fixed value	fixed gradient	kqR wall function	epsilon wall function

3.3 Case study

The general functions of the reservoir *De Blankaart* were already introduced in section 1.2. In this section, the construction, the inlet and outlet structure, and other technical specifications of the reservoir are presented. Furthermore, simplifications and assumptions of the model are discussed. Finally, the scenarios conducted in this study are summarised.

3.3.1 Reservoir "De Blankaart"

The water reservoir has a capacity of $3\,million\,m^3$. Its shape is an octahedron with a side length of $352.50\,m$, a surface area of $600\,000\,m^2$ and a water depth of $2.0\,to\,5.0\,m$ (Fig. 3.2). To prevent short circuiting, there is a partition wall from the inlet/outlet interface to the center of the octahedron.



Source: De Watergroep

Figure 3.2: 3D plot of the reservoir with a side length of $352.50\,m$, inelt and outlet structure seperated by a partition wall with a length of $425.50\,m$.

There are five inlet pumps with a capacity of about $2\,000\,m^3/hour$ each, and one with a capacity of about $600\,m^3/hour$. Each pump is mounted on a vertical shaft. Maximum three pumps operate at the same time limiting the intake of solid matters from the ring channel. The raw water is pumped into a vertical shaft, which is connected to a chamber of a horizontal channel, where the water enters the reservoir by an overflow weir (Fig. 3.3 and 3.4). It should be noted that the plants visible on the photograph (Fig. 3.4 left) are not supposed to grow inside the horizontal channel. After flowing around the partition wall, the water flows through circular openings into the outlet channel. Finally, the water passes a rake and is pumped to the treatment plant using a pump with a capacity of $1\,800\,m^3/hour$ (Fig. 3.3 and 3.5).

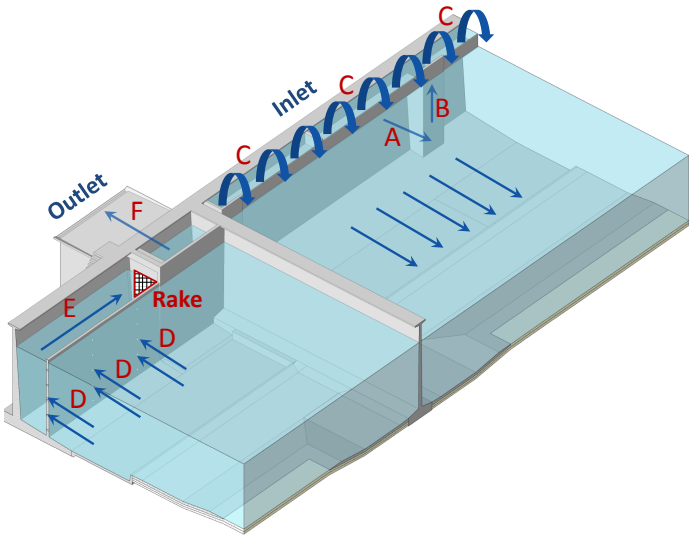


Figure 3.3: Schematic 3D illustration of inlet and outlet structure: the raw water (A) is pumped into a vertical shaft (B), which is connected to a chamber of a horizontal channel, where the water enters the reservoir by an overflow weir (C). After flowing around a partition wall, the water flows through circular openings (D) into the outlet channel (E). Finally, the water passes a rake (E to F) and is pumped to the treatment plant (F).

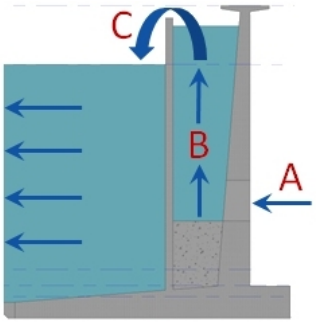


Figure 3.4: Photograph (left) and schematic illustration (right) of inlet structure: the raw water (A) is pumped into a vertical shaft (B), which is connected to a chamber of a horizontal channel, where the water enters the reservoir by an overflow weir (C). The plants visible on the photograph are not supposed to grow inside the horizontal channel.

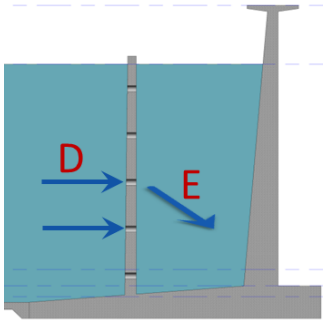


Figure 3.5: Photograph (left) and schematic illustration (right) of outlet structure: the water flows through circular openings (D) into the outlet channel (E).

3.3.2 Simplifications and assumptions

As mentioned before, the accuracy and computational cost of a numerical simulation depends on the mesh quality as well. The aim is the generation of an orthogonal structured mesh with low aspect ratios. Therefore, simplifications are made with respect to the geometry. The shape is simplified to an isosceles octahedron (Fig. 3.6). The bottom of the reservoir is assumed to be a plane surface having the same roughness as the walls made of concrete. The partition wall inside the reservoir is replaced by a baffle. This means that the cell faces along the wall axe are defined as an internal wall, and the boundary conditions of the internal wall are set to no-slip and zero flux through the faces, which are the same as for walls. Thus, the partition wall has a thickness of zero in the model. Furthermore, the inlet and outlet flow are uniform over the side wall and the inlet rate is assumed to be constant in the model.
Due to the fact that the ratio between the diameter and the water depth is more than 200, the main flow direction is horizontal. Therefore, this study is limited to horizontal fluid flow, and the CFD model is reduced to a two-dimensional problem (Fig. 3.6). Thus, fluctuations of the water level cannot be considered. During filling and emptying of the reservoir, the water height changes slow, thus, quasi steady state water height is a valid assumption. Furthermore, it is supposed that the outflow rate is equal to the inflow rate, and the precipitation and evaporation balance each other out.
In terms of fluid properties, pure water without any solids or solutes is assumed. The fluid is defined as an incompressible Newtonian fluid with constant temperature of 20.0°C and a kinematic viscosity ν of $1 \cdot 10^{-6} m^2/s$. Any heat transfer is neglected in this study.

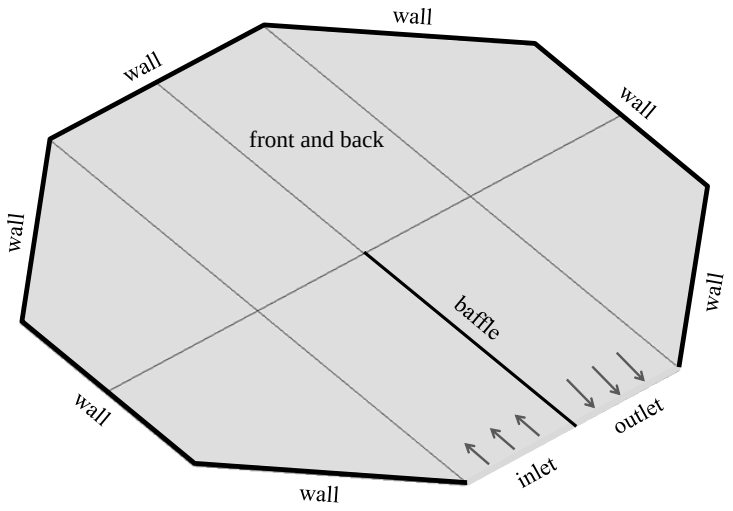


Figure 3.6: Schematic illustration of simplified 2D model: isosceles octahedron with wall boundaries (wall), plane surface (front and back), partition wall of zero thickness (baffle), and uniform inlet and outlet.

3.3.3 Case scenarios

Different scenarios are performed analysing the main flow pattern and the effect of structural changes. The simplifications and assumptions discussed in the previous section are valid for all scenarios. For comparability reasons, the scenarios are simulated using the exact same initial conditions defined in section 3.3.3.

Scenario 1: model verification

First, a CFD model of the original reservoir is developed determining the present flow pattern. A standard operation case was chosen. Assuming that the reservoir is filled completely ($h_w = 5.0\,m$), and three pumps with a capacity of $2\,000\,m^3/hour$ are operating simultaneously, the total inflow rate is $6\,000\,m^3/hour$ or $1.67\,m^3/sec$. Distributing uniformly the discharge Q over the total area of the simplified intake, with $l = 176.25\,m$, $h_w = 5.0\,m$, $and\,A = 528.75\,m^2$, the inlet velocity u_{in} is around $0.002\,m/s$, which is set as the velocity initial condition in the y-direction at the inlet boundary. For all internal fields within the domain, zero initial conditions are set for the velocity and pressure variables (Jasak 1996 (Jasak, 1996)). The model of the reservoir is shown schematically in Fig. 3.7 and its dimensions and parameters are listed in Table 3.2.

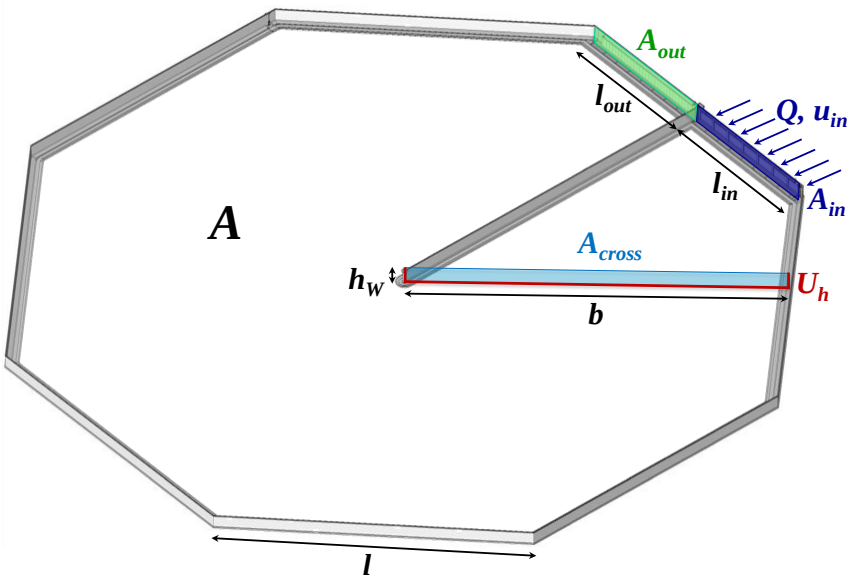


Figure 3.7: Schematic illustration of model *Scenario 1* with side length l, breadth b, water height h_W, surface area A, length l_{in} and surface area A_{in} of inlet, length l_{out} and surface area A_{out} of outlet, hydraulic cross sectional area A_{cross}, hydraulic perimeter U_h, discharge Q and inlet velocity u_{in}.

Table 3.2: Model dimensions and parameters

side length	$l =$	352.5	m
breadth	$b =$	425.5	m
surface area	$A =$	600 000	m^2
volume	$V = A \cdot h_w =$	3 000 000	m^3
cross-sectional area	$A_{cross} = b \cdot h_w =$	2 127.5	m^2
hydraulic perimeter	$U_h = b + 2 \cdot h_w =$	435.5	m
discharge	$Q =$	1.67	m^3/s
inlet velocity	$u_{in} = \frac{Q}{A} =$	0.002	m/s
kinematic viscosity	$\nu =$	$1 \cdot 10^{-6}$	m^2/s

Turbulence model parameters

Since the flow of study is turbulent, it is necessary to specify turbulence variables at the inlet. In terms of a $k - \varepsilon$ model, an initial estimation of the turbulent kinetic energy k and its dissipation ratio ε is required. Therefore, the Reynolds number Re (3.20), the turbulence intensity I (3.24) and the turbulence length scale l_T (3.26) are calculated. Finally, k (3.23) and ε (3.25) can be estimated and employed as initial conditions for all internal and boundary fields. The turbulence model parameters and initial conditions determined are summarised in Table 3.3.

Reynolds number Re :

$$Re = \frac{\bar{u} \cdot d_h}{\nu} \qquad Re = \frac{0.0008 \cdot 20}{1 \cdot 10^{-6}} = 16000 \tag{3.20}$$

$$with \qquad \bar{u} = \frac{Q}{A_{cross}} \qquad u = \frac{1.67}{2127.50} = 0.0008\, m/s \tag{3.21}$$

$$d_h = 4 \cdot \frac{A}{U_h} \qquad d_h = 4 \cdot \frac{2127.50}{435.50} \approx 20\, m \tag{3.22}$$

where $\bar{u}$ is the mean velocity, d_h the hydraulic diameter, Q the discharge, A_{cross} the cross-sectional area, and U_h the wetted perimeter.

Turbulent kinetic energy k :

$$k = \frac{3}{2} \left(u_{in} \cdot I\right)^2 \qquad k = \frac{3}{2} \left(0.002 \cdot 0.10\right)^2 = 6 \cdot 10^{-8}\, m^2/s^2 \tag{3.23}$$

where I is the turbulent intensity, which is evaluated next.

Turbulent intensity I **:** *CFD-Wiki* provides guidance setting the initial value of I.

> *"High-turbulence case: High-speed flow inside complex geometries like heat-exchangers and flow inside rotating machinery (turbines and compressors). Typically the turbulence intensity is between 5% and 20%.*
>
> *Medium-turbulence case: Flow in not-so-complex devices like large pipes, ventilation flows etc. or low speed flows (low Reynolds number). Typically the turbulence intensity is between 1% and 5%.*
>
> *Low-turbulence case: Flow originating from a fluid that stands still, like external flow across cars, submarines and aircrafts. Very high-quality wind-tunnels can also reach really low turbulence levels. Typically the turbulence intensity is very low, well below 1%."*
>
> (`http://www.cfd-online.com/Wiki/Turbulence_intensity`)

On the basis of these recommendations, a turbulent intensity of 10 % is chosen in this case study.

$$I \approx 0.1 \tag{3.24}$$

Dissipation ratio ε **:** The determination of ε requires an approximated turbulence length scale l_T. *CFD-Wiki* (Online, 1994) suggests to set l_T to 0.22 of the inlet boundary layer thickness t_{inlet} for wall-bounded inlet flows (`http://www.cfd-online.com/Wiki/Turbulent_length_scale`).

$$\varepsilon = c_\mu \cdot \frac{k^{3/2}}{l_T} \qquad \epsilon = 0.9 \cdot \frac{\left(6 \cdot 10^{-8}\right)^{3/2}}{1.1} = 1.2 \cdot 10^{-11}\, m^2/s^3 \tag{3.25}$$

$$with \qquad l_T \approx 0.22 \cdot t_{inlet} \qquad l_T = 0.22 \cdot 5.0 = 1.10\, m \tag{3.26}$$

where C_μ is a model constant predefined to 0.90 by default.

Table 3.3: Turbulence model parameters and initial conditions

patch type	velocity m/s	pressure m^2/s^2	turbulent energy m^2/s^2	dissipation ratio m^2/s^3
inlet	(0, 0.002, 0)	zero gradient	uniform $6 \cdot 10^{-8}$	uniform $1.2 \cdot 10^{-11}$
outlet	zero gradient	(0, 0, 0)	zero gradient	zero gradient
wall	(0, 0, 0)	zero gradient	uniform $6 \cdot 10^{-8}$	uniform $1.2 \cdot 10^{-11}$
front and back	empty	empty	empty	empty

Scenario 2: new outlet location

In order to investigate whether the relocation of the outlet has a beneficial effect on the water flow, the developed model is adapted. The location of the outlet is moved to the neighbouring wall, and the original outlet is altered into a impermeable wall (Fig. 3.8). It should be noted that the length of the new outlet is twice as long as the original one. As already mentioned, the initial conditions described in section *Scenario 1* are used in order to compare the scenarios.

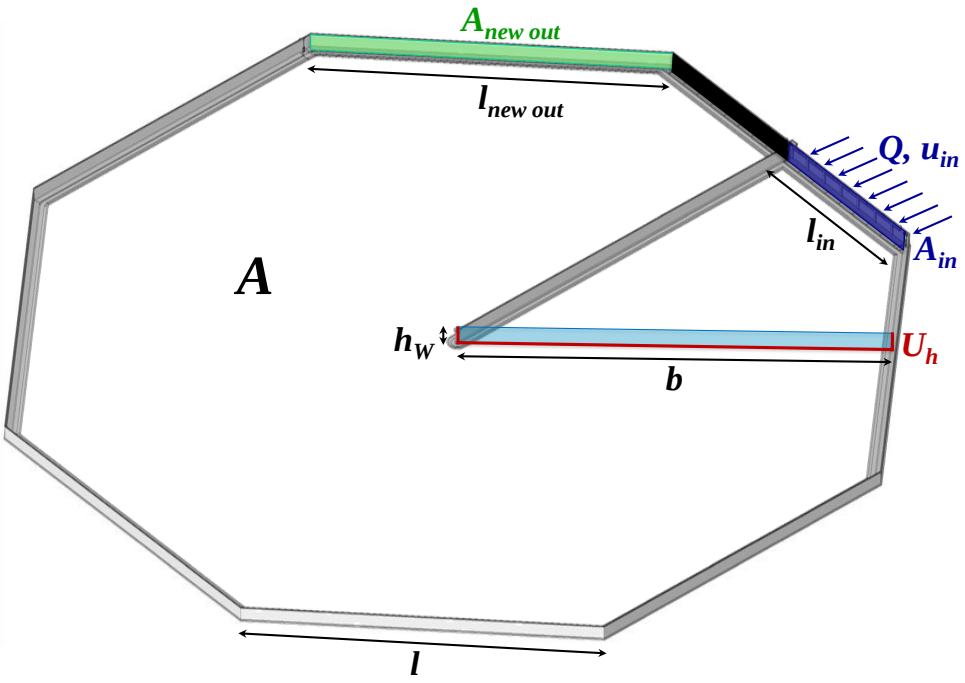


Figure 3.8: Schematic illustration of model *Scenario 2* with side length l, width b, water height h_W, surface area A, length l_{in} and surface area A_{in} of inlet, length $l_{new\,out}$ and surface area $A_{new\,out}$ of new outlet, hydraulic cross sectional area A_cross, hydraulic perimeter U_h, discharge Q, and inlet velocity u_{in}.

Scenario 3: additional partition wall

Based on the findings obtained from the previous case scenario analyses, a forth scenario is investigated additionally. The results of *Scenario 2* show that the relocation of the outlet does not improve the flow pattern. Thus, improvement measures should focus on achieving a more uniform flow inside the reservoir. This might be achieved by a second partition wall orthogonal to the present one. A scheme of the adapted model is shown in Fig. 3.9. In the model, the partition wall is built by a baffle, and thus, it has a thickness of zero. The outlet is situated at the original location. The boundary and initial conditions are the same as defined in *Scenario 1*.

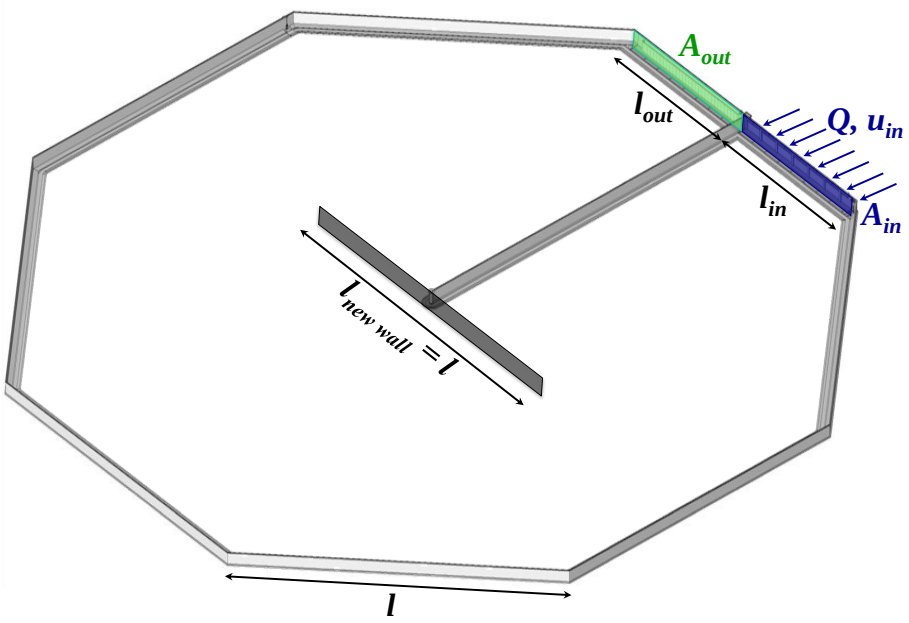


Figure 3.9: Schematic illustration of model *Scenario 4* with length $l_{new\,wall}$ of new partition wall which is equal to the side length l of the reservoir, length l_{in} and surface area A_{in} of inlet, length l_{out} and surface area A_{out} of outlet, discharge Q and inlet velocity u_{in}.

Scenario 4: three inlet sections

At the reservoir, there are six inlet pumps installed whereas only three pumps can be used simultaneously. By means of the CFD model, it can be investigated whether the flow pattern differs, if only pumps on the left or right side are operating. Therefore, a model with three inlet sections is developed and two different states are examined. In *State 1*, the left section is defined as inlet. The sections in the middle and on the right side are defined as external wall, Fig. 3.10. In *State 2*, the right section is defined as inlet. The sections in the middle and on the left side are defined as external wall, Fig. 3.11. The initial conditions are equal to the conditions in *Scenario 1*.

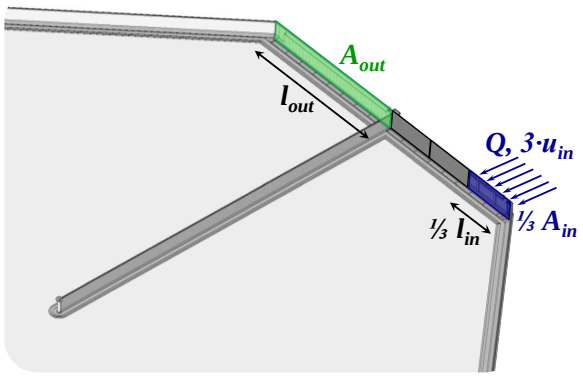


Figure 3.10: Schematic illustration of model *Scenario 3) State 1)*: inlet at left section with length $\frac{1}{3}l_{in}$ and surface area $\frac{1}{3}A_{in}$ of inlet section, length l_{out} and surface area A_{out} of outlet, discharge Q and inlet velocity $(3 \cdot u_{in})$.

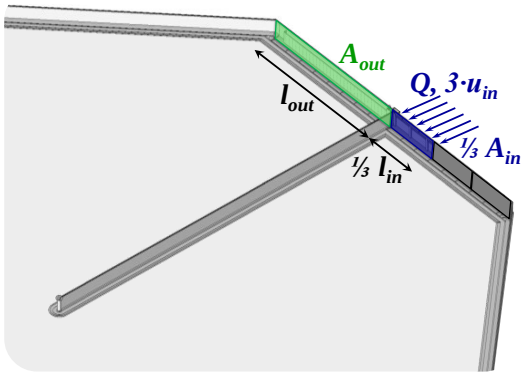


Figure 3.11: Schematic illustration of model *Scenario 3) State 2)*: inlet at right section with length $\frac{1}{3}l_{in}$ and surface area $\frac{1}{3}A_{in}$ of inlet section, length l_{out} and surface area A_{out} of outlet, discharge Q and inlet velocity $(3 \cdot u_{in})$.

3.4 Software

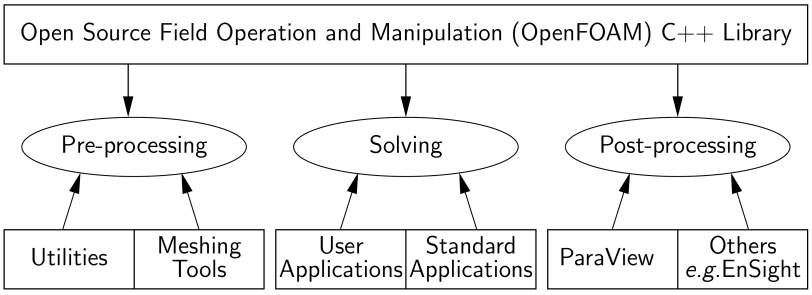
In this study, solely open-source software is used. The advantage is their cost-free and unconditionally availability from practically everywhere and at any time. It is very convenient for academic use, as it can be tuned and modified in any possible way to meet the needs of the research. The applied software for pre-processing, numerical solving and post-processing are concisely reviewed in this section.

3.4.1 SALOME

SALOME is a free, open source software to pre- and post-process CAD models for numerical simulations. The software runs via a graphical user interface, or a *Python* console (command language or script)(Python-Software-Foundation, 2014). The main feature of *SALOME* is the generation of a computational mesh using meshing algorithms. Based on predefined hypotheses, the mesh is computed. Several functions are available for quality controls of meshes, e.g. free faces, aspect ratio, skewness, double surfaces (Open Cascade, 2014).

3.4.2 OpenFOAM

OpenFOAM is a free, open source computational fluid dynamics software package to solve fluid flows, solid dynamics, and electro magnetics. In terms of fluid flow, problems of turbulences, chemical reactions, and heat transfer can be solved numerically. *OpenFOAM* also includes meshing tools, as well as pre- and post-processing features. Fig. 3.12 shows the basic structure of *OpenFOAM* (Open and Cfd, 2013).



Source: OpenFOAM (2014), page 15

Figure 3.12: Overview of OpenFOAM structure

3.4.3 ParaView

ParaView is a free, open source software for data analysing and visualization primarily, aimed at huge datasets (Sandia Corporation and Kitware Inc, 2014).

CHAPTER 4

Results

4.1 Introduction

In this chapter, the results of this work are presented. First, the created mesh, its quality and its degree of refinement are discussed. Then, the set-up for the calculations is defined. Finally, the results of *Case scenario 1*, *Case scenario 2*, and *Case scenario 3* are presented. Due to long calculation time (10 to 14 days) and limited availability of computational power, the simulation of *Case scenario 4* was not converged at the time of submission.

4.2 Mesh

In this study, the geometric model and its computational grid were created by *SALOME*. By means of vertexes, edges, wires, faces, and shells, solids were built, which were joined together to a compound representing the geometric model of the reservoir. Each geometric element was assigned to a predefined group according to its boundary patch type later on.
A Cartesian structured mesh (Fig. 4.1) made up of hexahedral cells was generated for the simplified model (see Fig. 3.6) of the reservoir. The hexahedral mesh was created using a regular 1D algorithm, a quadrangle 2D algorithm and a hexahedral 3D algorithm. With respect to the hypotheses, the number of segments in x-, y-, and z-direction were specified. In order to decrease the number of cells and thereby the computational expense, a cell ratio was added to the hypotheses generating a gradual, non equidistant mesh.
The low grid resolution of the mesh in figure 4.1 is shown for illustration purposes only. The mesh size of the mesh actually used is forty-times smaller than illustrated in figure 4.1. The grid dimensions of the applied mesh range from 0.50 m to 1.50 m using a cell ratio of three. The large cells are located in the vicinity of the walls as there are lower flow velocities expected, and thus, larger grid steps are considered to be sufficient. The total number of nodes and elements of the computational grid are listed in table 4.1.

Table 4.1: Basic information of the hexahedral mesh

	Total number	Geometrical form
Nodes	3 072 462	
Elements	4 634 840	
- edges	22 396	linear line
- faces	3 078 692	linear quadrangle
- volumes	1 533 752	linear hexahedron

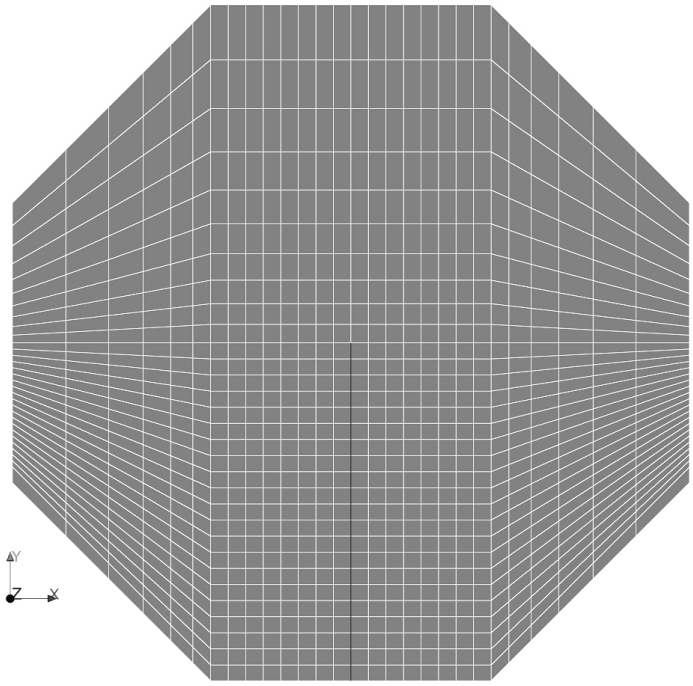
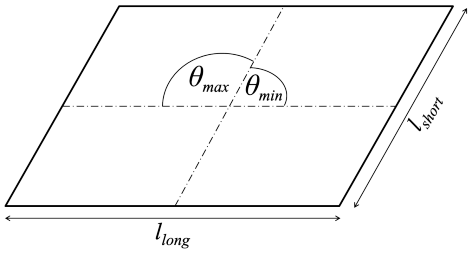


Figure 4.1: Schematic mesh illustration: Cartesian structured mesh with a cell ratio of three made up of hexahedral cells.

For the geometry and the mesh generation, a *Python* script (Python-Software-Foundation, 2014) was written making the modification of the mesh for future research easier. The script can be understood as a recipe for the creation of any mesh, leading to more reproducible science and results. The *Python* code is attached as an appendix, and supplied with the data disc as well.

4.2.1 Mesh quality

The mesh quality was checked by quality control tools of *SALOME* and by the *OpenFOAM checkMesh* function as well. It was verified that the computed mesh does not contain free nodes, double nodes, double edges, double faces, or double volumes. Furthermore, the skewness (in *OpenFOAM* called mesh non-orthogonality) and the face aspect ratio were analysed, which are explained by means of figure 4.2 and equation (4.1). In practice, the skewness/non-orthogonality S should not exceed 70° or 80° (Rhoads, 2014). In this case, the skewness ranges from 0° to maximum 45° in the vicinity of the corners, which is acceptable. In order to correct for this grid effect, an non-orthogonal correctors is available in *OpenFOAM* specified by the *nNonOrthogonalCorrectors* keyword, which were increased to 3 (default is 0) in this case study (de Oliveira Samel Moraes et al., 2013). The face aspect ratio A (4.2) is the ratio between the longest l_{long} and the shortest l_{short} length of a quadrangle (Fig. 4.2). The aspect ratio histogram in figure 4.3 clearly demonstrates that the aspect ratio is close to 1 (1 is the best) for the majority of the faces which is an excellent result.
The overall result of the quality controls carried out verify that the mesh is of good quality.



$$S = max\left[\theta_{max} - 90^{\circ};\, 90^{\circ} - \theta_{min}\right] \qquad (4.1)$$

$$A = \frac{l_{long}}{l_{short}} \qquad (4.2)$$

Figure 4.2: Schematic illustration of a quadrangular cell face: the skewness is defined by the angle θ_{max} or θ_{max} between the two medians and qualified by equation (4.1). The face aspect ratio A is the ratio between the longest l_{long} and the shortest l_{short} length and determined by equation (4.2).

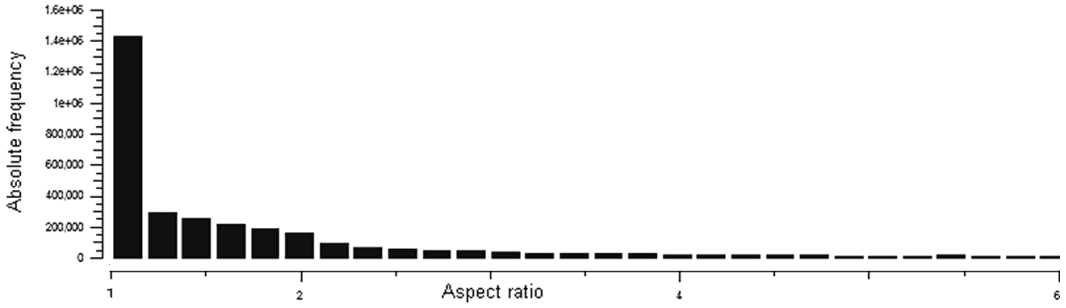


Figure 4.3: Face aspect ratio histogram with the aspect ratio on the ordinate, and the absolute frequency on the abscissae: for the majority of the faces the aspect ratio is close to 1 (1 is the best) demonstrating the good quality of the mesh.

4.2.2 Mesh independence

In order to attain reasonably accurate results, an sufficient small mesh size need to be applied. To test the mesh independence of the model, simulations using different mesh sizes were conducted. Mesh independence is reached when further refinement does no longer affect the accuracy of the results (Roache, 1997).

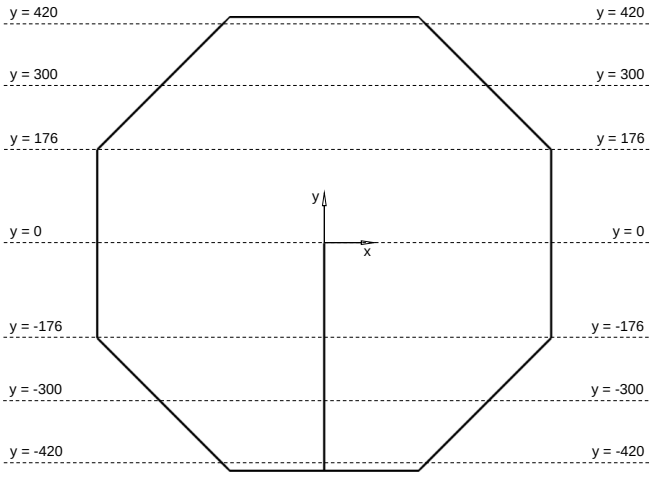


Figure 4.4: Schematic illustration of the reservoir showing the locations of the seven cross sections used for velocity data sampling.

For the mesh dependence test, three meshes were created with a mesh size of 2.00 m (Mesh A), 1.00 m (Mesh B), and 0.50 m (Mesh C). For the purpose of comparison, velocity data were sampled at seven cross sections as illustrated in Figure 4.4, and then plotted for visual analysis. In figure 4.5 and 4.5, the velocity profiles of the velocities in x- and y-direction sampled at y = 176 m are presented as an example. It can bee seen that the velocity profile of Mesh B and Mesh C are almost identical The profile of Mesh A is different. This means that Mesh A is too course, Mesh B is sufficient fine, and Mesh C is adequate but not efficient (Gruber et al., 2012). Furthermore, the comparison shows that at low velocity magnitudes, the velocities in Mesh A are in line with Mesh B and C. Thus, larger cells can be used in regions of low magnitudes. It is therefore advisable to use a gradient mesh, which will have less cells, and hence, will take less computational time.

Additional simulations were conducted using a grid with a mesh size ranging from 1.00 m to 2.00 m to decrease the number of cells even more. The first results looked promising. However, the simulations showed issues with resolving the turbulent parameters blowing up the number of iterations required, which increases enormously the computational time. Due to that reason, this mesh was not pursued any further.

Finally, a computational grid with a mesh size ranging from 0.50 m in the center to 1.50 m at the edge was applied (see section 4.2).

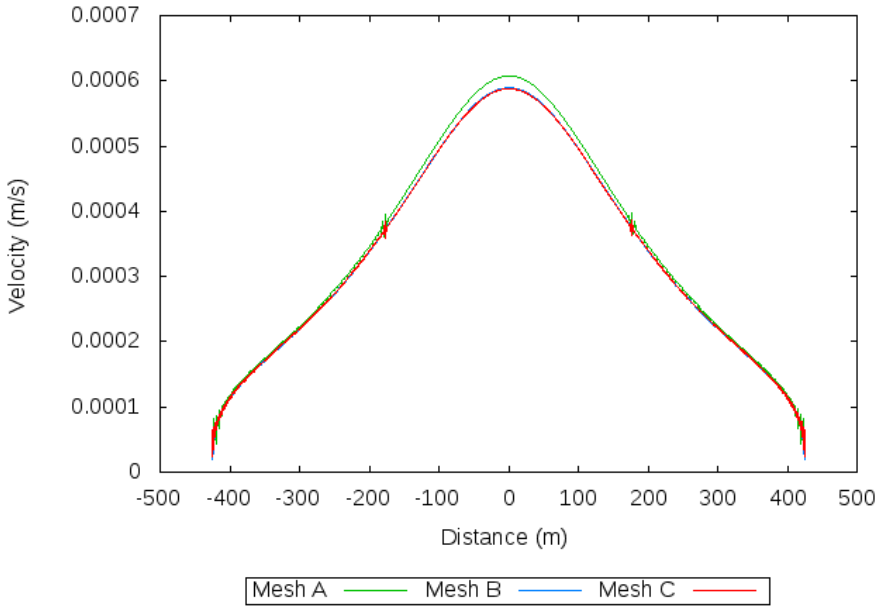


Figure 4.5: Velocity profile of the velocities in x- direction sampled at y = 176.25 m: the velocity profile of Mesh B and Mesh C are almost identical. The profile of Mesh A is different. This means that Mesh A is too course, Mesh B is sufficient fine, and Mesh C is adequate but not efficient.

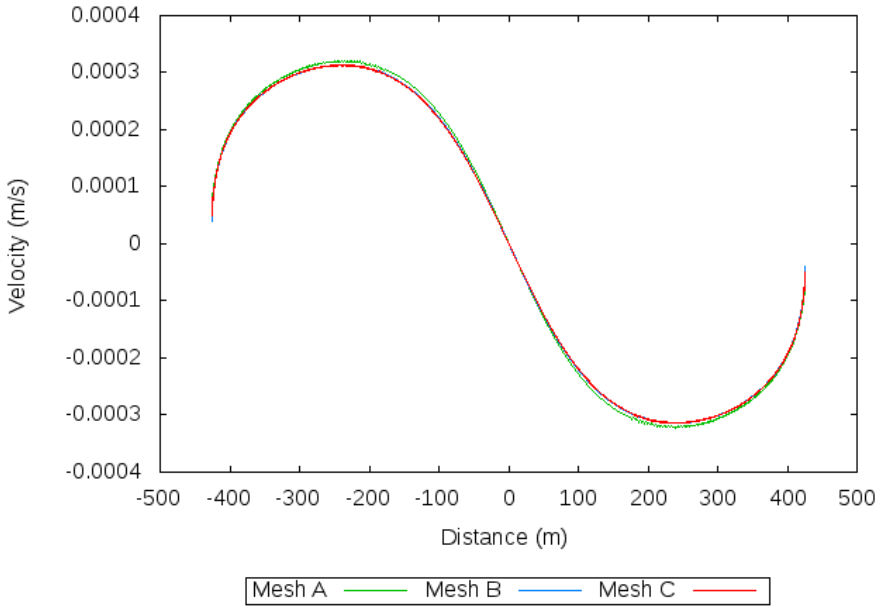


Figure 4.6: Velocity profile of the velocities in y- direction sampled at y = 176 m: the velocity profile of Mesh B and Mesh C are almost identical. The profile of Mesh A is different. This result is consistent with the results of the velocities in x- direction.

Regardless of the mesh dependence, the sampled data show up some issues for the velocity in x-direction at x = -176 m and x = +176 m (see in Figure 4.5 at x = -176 m and x = +176 m). The discontinuity indicates an interpolation error in x-direction due to the non-orthogonality of the cells in that region. Considering all velocity profiles, the interpolation error is greatest next to the wall and 0 at x = 0. Hence, the error only occurs in a small area, and its deviation is not that large, it is not essential to alleviate this error in the course of this study.

4.3 Case scenario 1

In *Case scenario 1*, the present flow pattern of the original reservoir were modelled. The inlet velocity of $0.002\,m/s$ were set as the velocity initial condition in the y-direction at the inlet boundary. For all internal fields within the domain, zero initial conditions were set for the velocity and pressure variables.
With respect to the simplifications and assumptions made, the model was reduced to a 2D, steady-state problem. Thus, no temporal discretisation is necessary and a steady-state algorithm can be used. In this case, first the SIMPLE algorithm (see section 3.2.1) was applied to reach steady state very fast. However, oscillating behaviour of the pressure field led to convergence problems.
In order to cope with the occurring oscillations, a pseudo-transient simulation using the PIMPLE algorithm (see section 3.2.1) was conducted. Pseudo-transient means to solve a physically steady-state problem using a transient algorithm. For cases that are very unstable, the PIMPLE algorithm is very useful to get a smooth convergence, since it applies relaxation factors and fully resolves the linear pressure-velocity coupling. Furthermore, an adjustable time step can be set which complies with the predefined *CFL* criterion speeding up the simulation to converge.

Table 4.2: Control settings for simulation in PIMPLE mode

controlDict	**variable**	**setting**
Algorithm	application	pimpleFoam
Start time	startTime	0
End time	endTime	1000000
Time step	deltaT	2
Adjustable time step	adjustTimeStep	yes
Maximal CFL number	maxCO	1

Table 4.3: Algorithm settings for simulation in PIMPLE mode

fvSolution		**pressure**	**velocity**	**k/epsilon**
Solvers:	Tolerance	1e-07	1e-07	1e-07
	Relative tolerance	0	0	0
Residual control:	Tolerance	1e-03	1e-04	1e-04
	Relative tolerance	0	0	0
Relaxation factors:	fields	3e-01		
	equations		7e-01	7e-01
nNonOrthogonalCorrectors	2			
nCorrectors	2			
nOuterCcorrectors	20			

For the simulation conducted in PIMPLE mode, the calculation set-up were configured for the algorithm specifications in the file *fvSolution*, and for the run control in the file *controlDict*. The control and algorithm settings are listed in Table 4.3 and 4.2. The schemes applied were chosen according to best practice settings provided by *OpenFOAM* and are not reviewed further. The initial conditions were already defined in section 3.3.3.

The model result is presented in Figure 4.9 and 4.10. Figure 4.9 shows the stream lines at $t = 940\,000\,seconds$, and figure 4.10 shows the velocity vectors at $t = 940\,000\,seconds$. In consequence of decreasing residuals, the simulation was considered to be converged at $t = 940\,000\,seconds$. The numerical model shows clearly a main water stream flowing along the partition wall. Arrived at the edge of the partition wall, the major part flows to the right of which a large part flows to the outlet exiting the reservoir, and a small part forms a large vortex (see on the right-hand side of the partition wall). The water stream flowing to the left at the end of the partition wall, forms a another vortex (see on the left-hand side of the partition wall). To the left of the inlet, there is a large dead zone present. Another dead zone is located at the upper left corner. Further smaller dead zones are located at the corners.

With respect to the validity of the model, the result looks reasonable. But from the technical point of view, the result of the present flow pattern is not satisfactory. The irregular flow pattern indicates a non-uniform residence time, but a uniform residence time is to be accomplished for the planned mixing and levelling inside the reservoir.
In consideration of the problems revealed, two improvement measures have been designed and investigated, of which results are presented in the following sections.

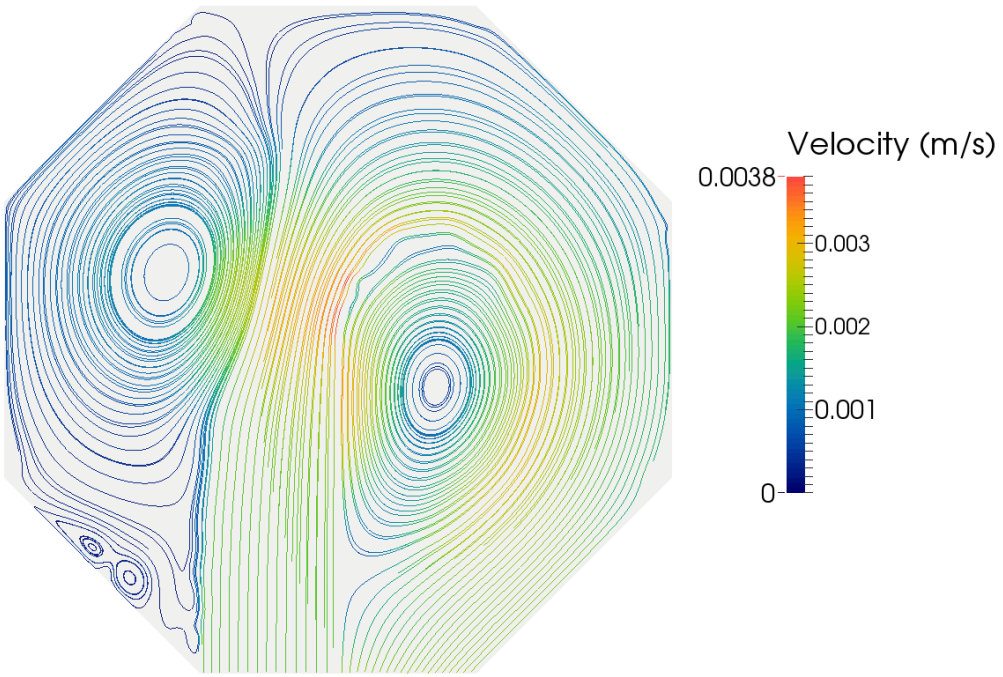


Figure 4.7: *Case scenario 1* - stream lines at t = 940 000 seconds

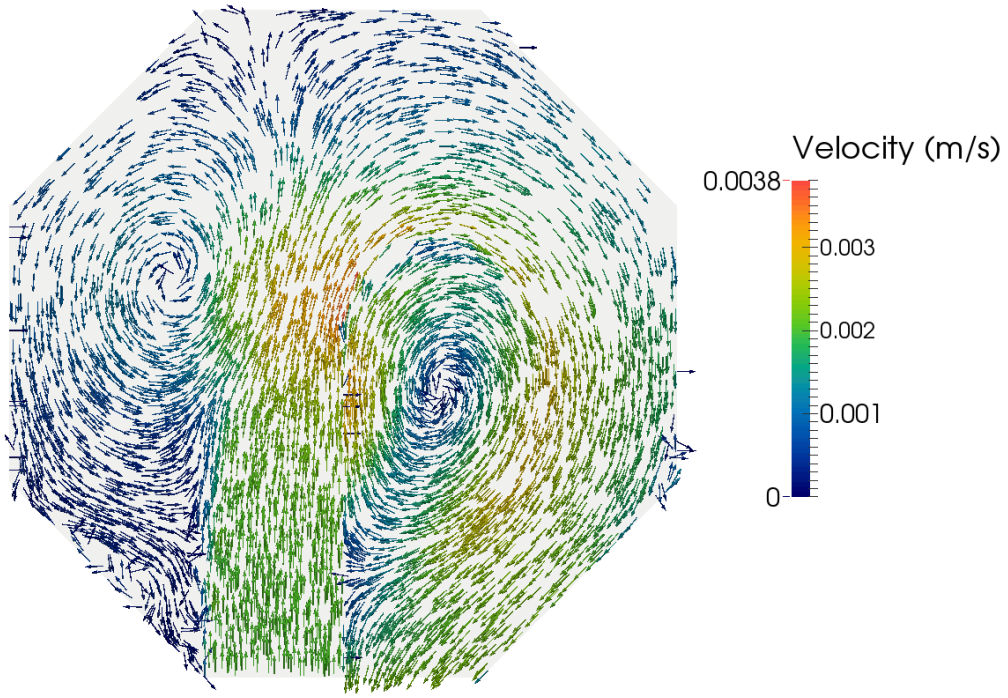


Figure 4.8: *Case scenario 1* - velocity vectors at t = 940 000 seconds

4.4 Case scenario 2

In *Case scenario 2* it was investigated whether the relocation of the outlet has a beneficial effect on the water flow. By modification of the boundary conditions of the developed model, the outlet was virtually moved to the neighbouring wall. The patches located at the new outlet location were defined as outlet, and the patches at the original outlet location were altered into wall patches. It should be noted that the length of the new outlet is twice as long as the original one.
The other boundary conditions remained the same, and the algorithm specifications and settings for the run control are identical to the set-up in *Case scenario 1* listed in Table 4.3 and 4.2. The initial conditions were already defined in section 3.3.3.

The results of the modelling are again presented in the form of stream lines (Fig. 4.9), and velocity vectors (Fig. 4.10). The model shows a flow pattern similar to *Case scenario 1.* A main water stream flows along the partition wall. At the end of the partition wall, one part flows to the right, of which a large part flows to the outlet exiting the reservoir, and a small part forms a large vortex (see on the right-hand side of the partition wall). The other part flows to the left and forms a huge vortex (see on the left-hand side of the partition wall) covering almost the entire left sector of the reservoir. To the left of the inlet, the large dead zone is still present. The relocation of the outlet causes even a formation of an additional dead zone in the region of the original outlet location (see on the bottom right of the partition wall).

The result of *Case scenario 2* demonstrates clearly that the relocation of the outlet to the neighbouring wall has no beneficial effect on the water flow. The contrary is the case. The relocation causes the occurrence of additional dead zones, and thus, the flow pattern deteriorates compared with the original conditions.

Since, the relocation of the outlet does not improve the flow pattern, another improvement measure was investigated, whose results are presented in *Case scenario 3.*

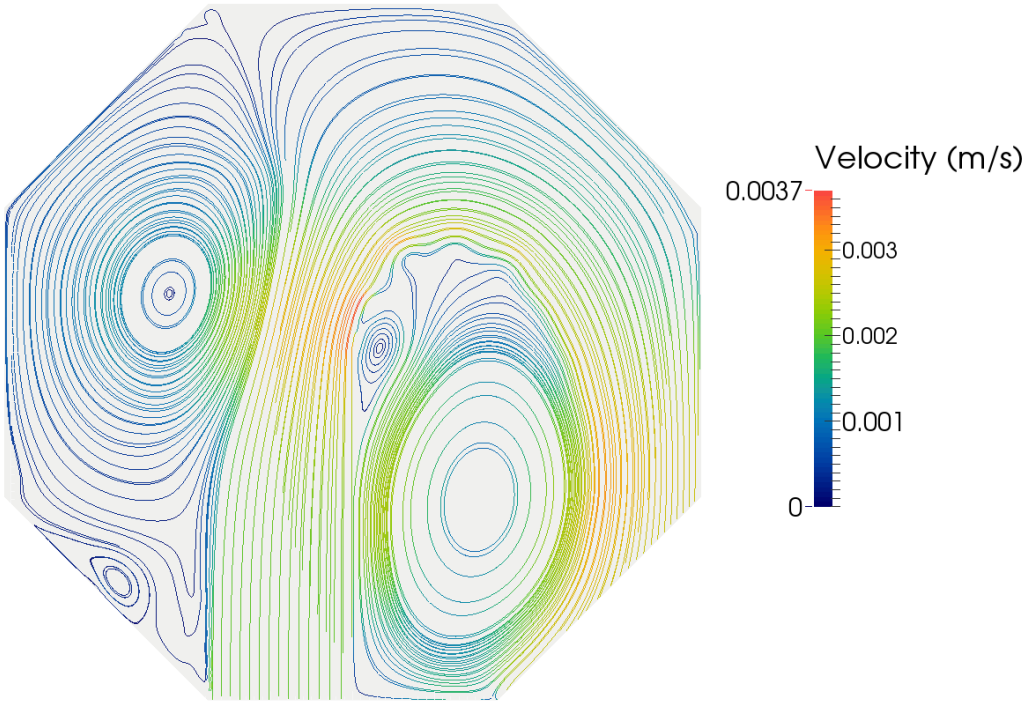


Figure 4.9: *Case scenario 2* - stream lines at t = 940 000 seconds

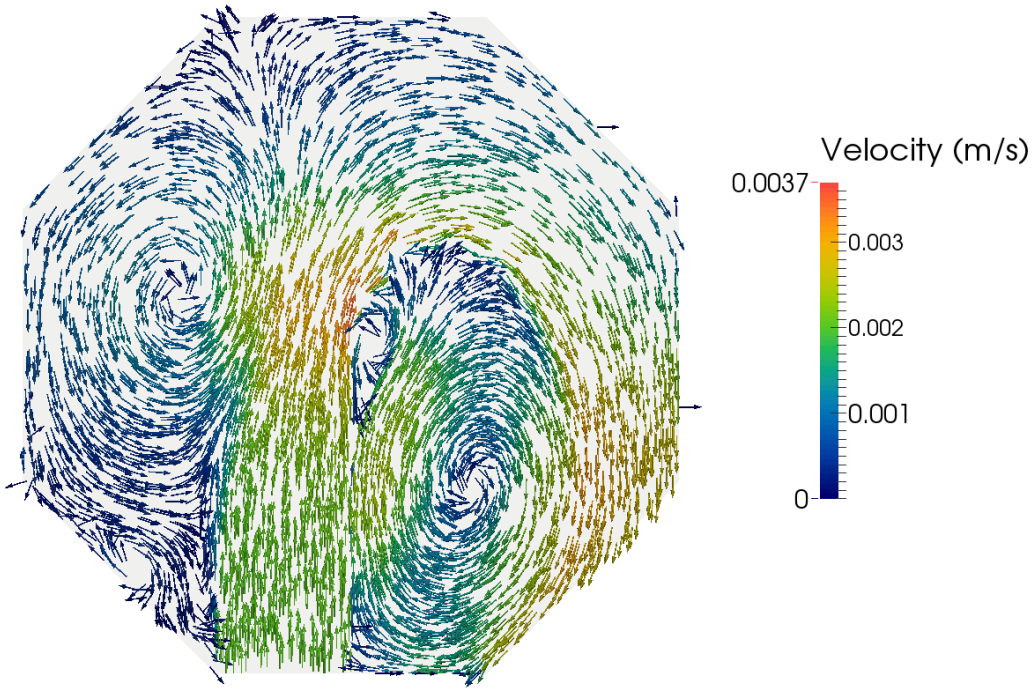


Figure 4.10: *Case scenario 2* - velocity vectors at t = 940 000 seconds

4.5 Case scenario 3

Based on the findings obtained from the previous case scenario analyses, a forth scenario was investigated. As in the previous section discussed, the relocation of the outlet does not improve the flow pattern. In *Case scenario 3*, an internal wall was implemented to the model, and its effects on the flow pattern were analysed. Therefore, an additional wall boundary was built by a baffle with a thickness of zero. The algorithm specifications and the settings for the run control are defined as in *Case scenario 1* and are listed in Table 4.3 and 4.2. The initial conditions of *Scenario 3* were already defined in section 3.3.3.

The results of the model are once more presented in the form of stream lines (Fig. 4.11), and velocity vectors (Fig. 4.12). The result of *Case scenario 3* shows an significant improvement of the flow pattern due to the construction of the internal wall. The internal wall prevents the shortcut around the partition wall and causes a mixing process at the upper section of the reservoir. Furthermore, dead zone at the left corner has been reduced by the measure implemented.
The main flow at the right section of the reservoirs is less subjected by the restructure, since the pattern looks similar to the original one. Nevertheless, the dead zone on the left of the partition wall covers a considerable smaller area than in the original case. An improvement at the right section might be achieved deleting the right part of the internal wall, so that the internal wall only expense to the left side of the partition wall. In order to prove this presumption, further analyses are necessary. However, the construction of an internal wall was proved to have a positive impact on the flow pattern.

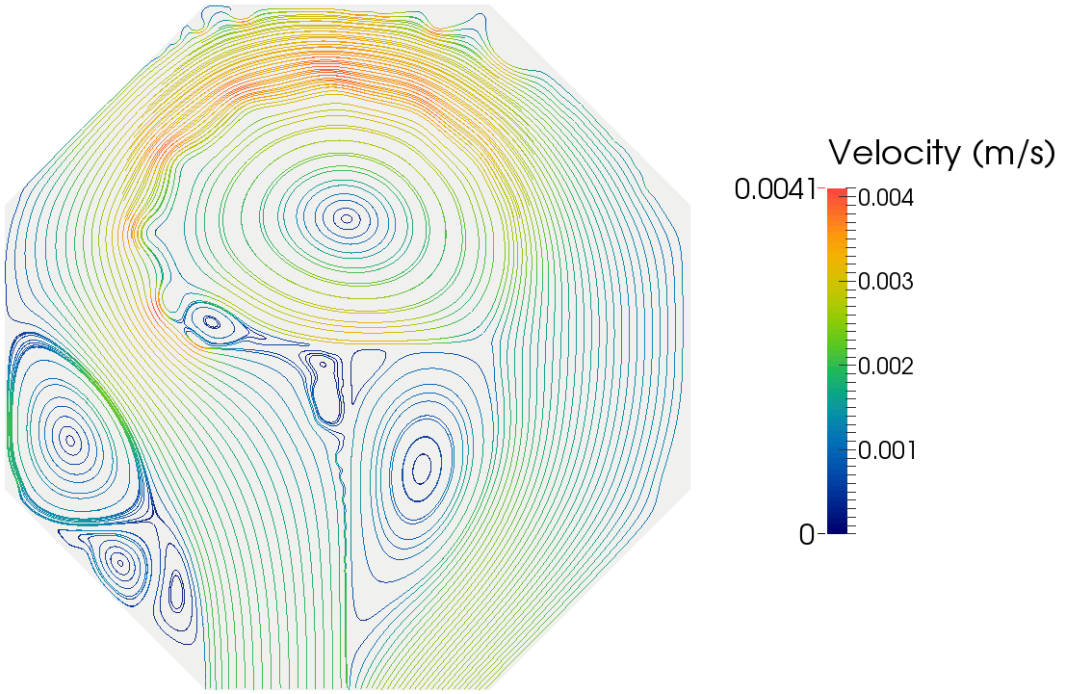


Figure 4.11: *Case scenario 3* - stream lines at t = 940 000 seconds

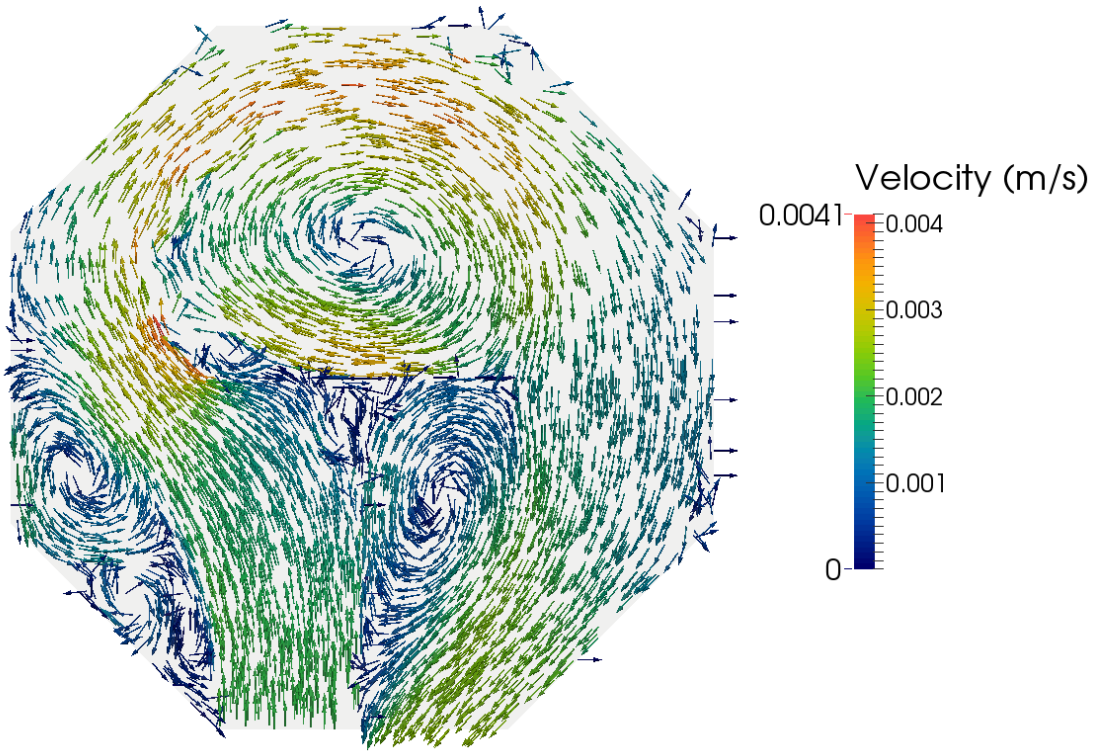


Figure 4.12: *Case scenario 3* - velocity vectors at t = 940 000 seconds

CHAPTER 5

Conclusions and future prospects

5.1 Summary

The objective of this research was the successful application of computational fluid dynamics modelling on a large, shallow water reservoir. The water production center *De Blankaart* served as a case study.
The CFD model was developed to estimate the flow inside the reservoir, and to provide guidance about possible improvements by restructuring. As a first step, an appropriate mesh was designed for a simplified geometry of the reservoir. Then a computational fluid dynamics model was developed applying the finite volume method to identify the present flow pattern. Furthermore, different scenarios were modelled concerning improvement measures achieving a more uniform flow inside the reservoir. In a first case scenario, the outlet were virtually relocated with the aim of improving the flow pattern. In a second scenario, an internal wall attached to the partition wall was implemented into the CFD model, and its effect on the flow pattern was investigated.

5.2 Conclusions

The result of the first case scenario determining the present flow pattern is in accordance with the presumptions made. The flow pattern is characterised by a main stream around the partition wall and large areas of dead zones. The irregular flow pattern indicates a non-uniform residence time, which restrains the mixing and levelling process.
The relocation of the outlet with the aim of improving the flow pattern, has proved incapable. The restructuring causes additional dead zones, which should be avoided.
With respect to the results obtained, another improvement measure was investigated. An internal wall was implemented into the CFD model in order to prevent the shortcut around the partition wall. The results showed an improved flow pattern. The internal wall acting as

a barrier causes mixing and circulation of the water. Furthermore, the areas of dead zones decreased significantly. Thus, the implementation of an internal wall was proved to have a positive impact on the flow pattern.
To conclude, the CFD model of the reservoir provides adequate information to understand the present flow pattern. Furthermore, an efficient improvement measure was found by the case study analysis.

The computational fluid dynamics model shows reasonable results, thus, the application of CFD modelling on large, shallow water reservoir has been successful. But it has to be noted that, due to the large dimension of the reservoir, the simulation was very computationally expensive exceeding calculation times of 14 days. In further research, this should be taken into account to be equipped accordingly.

5.3 Future prospects

To complete the research on the reservoir *De Blankaart*, more analysis need to be done. Following specific investigations are suggested.
The *Case scenario 4* presented in section 3.3.3, is to be finalised, which can be done straight away as the entire simulation set-up is prepared. As discussed in section 4.5, *Case scenario 3* is to be modified reducing the length of the internal wall, and then recalculated. It is to analyse whether the remodelling improves the flow pattern with comparison to the original model of *Case scenario 3*.

Furthermore, a virtual tracer test is to be conducted determining the residence time. All necessary solvers, and settings are already provided. It is suggested to apply the virtual tracer test to all case scenarios as it provides additional information on the flow pattern.

Finally, the validation of the developed model is of main importance. Therefore, velocity measurements or a tracer test are necessary. Both techniques are going to be a difficult task to undertake. In this specific case, it is very difficult to measure the velocity inside the reservoir since its magnitude is extremely low. Measuring instruments applicable for such low magnitudes are rarely available. The tracer tests would be very time-consuming due to the long residence time of about 4 months on average. Furthermore, the degradation of the tracer is of major concern. In the light of current knowledge, it is suggested to conduct point measurements using a electromagnetic current meter as it is well applicable in low flow environments.

Bibliography

Angeloudis, A. (2014). *Numerical and experimental modelling of flow and kinetic processes in serpentine disinfection tanks*. PhD thesis, Cardiff University.

Bestehron, M. (2006). *Hydrodynamik und Strukturbildung*. Springer-Verlag.

Bollrich, G. (2000). *Technische Hydromechanik: Grundlagen*. Beuth.

Council, T. E. U. (1998). Council directive 98/83/ec of 3 november 1998 on the quality of water intended for human consumption. *Official Journal of the European Communities*.

Courant, R., Friedrichs, K., and Lewy, H. (1928). Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74.

de Moel, P. J., Verberk, J. Q. J. C., and van Dijk, J. C. (2006). *Drinking Water*. World Scientific Publishing Co. Pte. Ltd.

de Oliveira Samel Moraes, A., da Cunha Lage, P. L., Cunha, G. C., and da Silva, L. F. L. R. (2013). Analysis of the non-orthogonality correction of finite volume discretization on unstructured meshes. *22nd International Congress of Mechanical Engineering (COBEM 2013)*, pages 3519–3530.

Gruber, M. F., Johnson, C. J., Tang, C., Jensen, M. H., Yde, L., and Hélix-Nielsen, C. (2012). Validation and analysis of forward osmosis cfd model in complex 3d geometries. *Membranes*, pages 764–782.

Jasak, H. (1996). *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, Department of Mechanical Engineering Imperial College of Science, Technology and Medicine.

Launder, B. and Spalding, D. (1974). The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269 – 289.

Laurien, E. and jr., H. O. (2011). *Numerische Strmungsmechanik*. Vieweg und Teubner Verlag, Springer Fachmedien Wiesbaden GmbH 2011.

Lecheler, S. (2011). *Numerische Strmungsberechnung*. Vieweg und Teubner Verlag, Springer Fachmedien Wiesbaden GmbH 2011.

Levenspiel, O. (1999). *Chemical Reaction Engineering*. John Wiley & Sons, Inc.

Online, C. (1994). CFD Online webpage.

Open, T. and Cfd, S. (2013). Open FOAM. (June).

Open Cascade (2014). Salome Platform web page.

OpenFOAM (2014). The open source cfd toolbox - user guide. *OpenFOAM Foundation*.

Parlament, T. E. (2000). Directive 2000/60/ec of the european parliament and of the council of 23 october 2000 establishing a framework for community action in the field of water policy. *Official Journal of the European Communities*.

Python-Software-Foundation (2014). Python web page.

Rhoads, J. (2014). Eects of grid quality on solution accuracy. *OpenFOAM Workshop 2014*.

Roache, P. J. (1997). Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics*, pages 124–161.

Sandia Corporation and Kitware Inc (2014). Paraview web page.

Schwarze, R. (2013). *CFD-Modellierung*. Springer Vieweg.

Shivamoggi, B. K. (1998). *Theoretical Fluid Dynamics*. John Wiley & Sons, Inc.

Sigloch, H. (2014). *Technische Fluidmechanik*. Springer-Verlag.

UN (2004). Resolution adopted by the general assembly. *International Decade for Action, Water for Life, 2005-2015*.

WHO (1970). European standards for drinking-water. *World Health Organization*.

Zikanov, O. (2010). *Essential computational fluid dynamics*. John Wiley & Sons, Inc.

APPENDIX A

Mesh generation code

```
####------------------------####
#### Mesh generation SALOME ####
####------------------------####

#importing necessary libraries and defining functions

import salome
from salome.geom import geomBuilder
geompy = geomBuilder.New(salome.myStudy)
gg = salome.ImportComponentGUI("GEOM")

import SMESH, SALOMEDS
from salome.smesh import smeshBuilder
smesh =  smeshBuilder.New(salome.myStudy)

import math

####-----------####
#### Constants ####
####-----------####

### Specifying the geometric parameters of the isosceles octahedron! ###

# NOTE: isosceles = all 8 sides of the octahedron have the same length!
# NOTE: angle between the sides = 135 degree!

# Enter length in meter of one side of octahedron here
l = 352.5
# Enter water hight in meter of octahedron here
h = 5.

# NOTE: Width of inlet is defined to be equal to one cell!
```

```
33 # NOTE: Width of middle wall is neglected!
34 # NOTE: Length of middle wall is equal to half diameter!
35
36 #Enter number of cells per 1 meter in x/y/z
37 n_x_inlet_west = 2./1.
38 n_x_outlet_west = 2./1.
39 n_y_south = 2./1.
40 n_z = 0.2/1.
41 cellratio = 3.
42
43 ### Calculation of other geometric parameters of octahedron ###
44
45 # length of short-sides of triangle
46 s = l/(math.sqrt(2.))
47 # "radius" = l/2+s = half diameter = length of middle wall
48 r = l/2.+l/(math.sqrt(2.))
49 # Width of inlet: defined to be equal to one cell
50 w = 1./n_y_south
51
52 ####----------####
53 #### Geometry ####
54 ####----------####
55
56 ### Making first quarter of octahedron ###
57
58 print "making points at z=0"
59 point0_1 = geompy.MakeVertex(0., 0., 0.)
60 point0_2 = geompy.MakeVertex(l/2, 0., 0.)
61 point0_3 = geompy.MakeVertex(r, 0., 0.)
62 point0_4 = geompy.MakeVertex(r, l/2, 0.)
63 point0_5 = geompy.MakeVertex(l/2, r, 0.)
64 point0_6 = geompy.MakeVertex(0., r, 0.)
65
66 print "making points at z=h"
67 pointh_1 = geompy.MakeVertex(0., 0., h)
68 pointh_2 = geompy.MakeVertex(l/2, 0., h)
69 pointh_3 = geompy.MakeVertex(r, 0., h)
70 pointh_4 = geompy.MakeVertex(r, l/2, h)
71 pointh_5 = geompy.MakeVertex(l/2, r, h)
72 pointh_6 = geompy.MakeVertex(0., r, h)
73
74 print "making horizontal edges at z=0"
75 edge0_12 = geompy.MakeEdge(point0_1, point0_2)
76 edge0_16 = geompy.MakeEdge(point0_1, point0_6)
77 edge0_23 = geompy.MakeEdge(point0_2, point0_3)
78 edge0_25 = geompy.MakeEdge(point0_2, point0_5)
79 edge0_34 = geompy.MakeEdge(point0_3, point0_4)
80 edge0_45 = geompy.MakeEdge(point0_4, point0_5)
```

```
81 edge0_56 = geompy.MakeEdge(point0_5, point0_6)
82
83 print "making horizontal edges at z=h"
84 edgeh_12 = geompy.MakeEdge(pointh_1, pointh_2)
85 edgeh_16 = geompy.MakeEdge(pointh_1, pointh_6)
86 edgeh_23 = geompy.MakeEdge(pointh_2, pointh_3)
87 edgeh_25 = geompy.MakeEdge(pointh_2, pointh_5)
88 edgeh_34 = geompy.MakeEdge(pointh_3, pointh_4)
89 edgeh_45 = geompy.MakeEdge(pointh_4, pointh_5)
90 edgeh_56 = geompy.MakeEdge(pointh_5, pointh_6)
91
92 print "making vertical edges"
93 edgev_11 = geompy.MakeEdge(point0_1, pointh_1)
94 edgev_22 = geompy.MakeEdge(point0_2, pointh_2)
95 edgev_33 = geompy.MakeEdge(point0_3, pointh_3)
96 edgev_44 = geompy.MakeEdge(point0_4, pointh_4)
97 edgev_55 = geompy.MakeEdge(point0_5, pointh_5)
98 edgev_66 = geompy.MakeEdge(point0_6, pointh_6)
99
100 print "making horizontal wires"
101 wire_down_west = geompy.MakeWire([edge0_12, edge0_25, edge0_56, edge0_16
      ],0)
102 wire_up_west = geompy.MakeWire([edgeh_12, edgeh_25, edgeh_56, edgeh_16],0)
103 wire_down_east = geompy.MakeWire([edge0_23, edge0_34, edge0_45, edge0_25
      ],0)
104 wire_up_east = geompy.MakeWire([edgeh_23, edgeh_34, edgeh_45, edgeh_25],0)
105
106 print "making vertical wires"
107 wire_side_southwest = geompy.MakeWire([edge0_12, edgev_22, edgeh_12,
      edgev_11],0)
108 wire_side_southeast = geompy.MakeWire([edge0_23, edgev_33, edgeh_23,
      edgev_22],0)
109 wire_side_east = geompy.MakeWire([edge0_34, edgev_44, edgeh_34, edgev_33
      ],0)
110 wire_side_diagonal = geompy.MakeWire([edge0_45, edgev_55, edgeh_45,
      edgev_44],0)
111 wire_side_north= geompy.MakeWire([edge0_56, edgev_66, edgeh_56, edgev_55
      ],0)
112 wire_side_west= geompy.MakeWire([edge0_16, edgev_66, edgeh_16, edgev_11
      ],0)
113 wire_side_middle= geompy.MakeWire([edge0_25, edgev_55, edgeh_25, edgev_22
      ],0)
114
115 print "making horizontal faces"
116 F_down_west = geompy.MakeFace(wire_down_west,1) #1 = planar face
117 F_up_west = geompy.MakeFace(wire_up_west,1)
118 F_down_east = geompy.MakeFace(wire_down_east,1)
119 F_up_east = geompy.MakeFace(wire_up_east,1)
```

```
120
121 print "making vertical faces"
122 F_side_southwest = geompy.MakeFace(wire_side_southwest,1)
123 F_side_southeast = geompy.MakeFace(wire_side_southeast,1)
124 F_side_east = geompy.MakeFace(wire_side_east,1)
125 F_side_diagonal = geompy.MakeFace(wire_side_diagonal,1)
126 F_side_north = geompy.MakeFace(wire_side_north,1)
127 F_side_west = geompy.MakeFace(wire_side_west,1)
128 F_side_middle = geompy.MakeFace(wire_side_middle,1)
129
130 print "making 2 shells"
131 shell_west = geompy.MakeShell([F_down_west, F_up_west, F_side_southwest,
       F_side_middle, F_side_north, F_side_west])
132 shell_east = geompy.MakeShell([F_down_east, F_up_east, F_side_southeast,
       F_side_east, F_side_diagonal, F_side_middle])
133
134 print "making 2 solids"
135 solid_west = geompy.MakeSolid([shell_west])
136 solid_east = geompy.MakeSolid([shell_east])
137
138 print "making compound_solid"
139 compound_solid = geompy.MakeCompound([solid_west, solid_east])
140 print "glueing compound_solid"
141 compound_solid = geompy.MakeGlueFaces(compound_solid,10**-7,1)
142 print "adding compound_solid to study"
143 geompy.addToStudy(compound_solid,"compound_solid")
144
145 ### Making second quarter of octahedron using symmetry plane ###
146
147 print "making mirrored compound of compound_solid at y-axis"
148 #vector normal to symmetry plane
149 vector_x00 = geompy.MakeVectorDXDYDZ(-1., 0., 0.)
150 #symmetry plane at y-axis
151 plane_symm_y = geompy.MakePlane(point0_1,vector_x00,1000)
152 #making mirror of compound_solid
153 compound_mir_solid = geompy.MakeMirrorByPlane(compound_solid,plane_symm_y)
154 print "adding compound_mir_solid to study"
155 geompy.addToStudy(compound_mir_solid,"compound_mir_solid")
156
157 ### Making outlet quarter of octahedron ###
158
159 print "making points at z=0"
160 point0_1 = geompy.MakeVertex(0., 0., 0.)
161 point0_2 = geompy.MakeVertex(l/2, 0., 0.)
162 point0_3 = geompy.MakeVertex(r, 0., 0.)
163 point0_4 = geompy.MakeVertex(r, -l/2, 0.)
164 point0_5 = geompy.MakeVertex(l/2, -r, 0.)
165 point0_6 = geompy.MakeVertex(0., -r, 0.)
```

```
166 point0_7 = geompy.MakeVertex(0., -(r-w), 0.)
167 point0_8 = geompy.MakeVertex(l/2., -(r-w), 0.)
168 point0_9 = geompy.MakeVertex(r, -(l/2.-w), 0.)
169
170 print "making points at z=h"
171 pointh_1 = geompy.MakeVertex(0., 0., h)
172 pointh_2 = geompy.MakeVertex(l/2, 0., h)
173 pointh_3 = geompy.MakeVertex(r, 0., h)
174 pointh_4 = geompy.MakeVertex(r, -l/2, h)
175 pointh_5 = geompy.MakeVertex(l/2, -r, h)
176 pointh_6 = geompy.MakeVertex(0., -r, h)
177 pointh_7 = geompy.MakeVertex(0., -(r-w), h)
178 pointh_8 = geompy.MakeVertex(l/2., -(r-w), h)
179 pointh_9 = geompy.MakeVertex(r, -(l/2.-w), h)
180
181 print "making horizontal edges at z=0"
182 edge0_12 = geompy.MakeEdge(point0_1, point0_2)
183 edge0_17 = geompy.MakeEdge(point0_1, point0_7)
184 edge0_23 = geompy.MakeEdge(point0_2, point0_3)
185 edge0_28 = geompy.MakeEdge(point0_2, point0_8)
186 edge0_39 = geompy.MakeEdge(point0_3, point0_9)
187 edge0_45 = geompy.MakeEdge(point0_4, point0_5)
188 edge0_49 = geompy.MakeEdge(point0_4, point0_9)
189 edge0_56 = geompy.MakeEdge(point0_5, point0_6)
190 edge0_58 = geompy.MakeEdge(point0_5, point0_8)
191 edge0_67 = geompy.MakeEdge(point0_6, point0_7)
192 edge0_78 = geompy.MakeEdge(point0_7, point0_8)
193 edge0_89 = geompy.MakeEdge(point0_8, point0_9)
194
195 print "making horizontal edges at z=h"
196 edgeh_12 = geompy.MakeEdge(pointh_1, pointh_2)
197 edgeh_17 = geompy.MakeEdge(pointh_1, pointh_7)
198 edgeh_23 = geompy.MakeEdge(pointh_2, pointh_3)
199 edgeh_28 = geompy.MakeEdge(pointh_2, pointh_8)
200 edgeh_39 = geompy.MakeEdge(pointh_3, pointh_9)
201 edgeh_45 = geompy.MakeEdge(pointh_4, pointh_5)
202 edgeh_49 = geompy.MakeEdge(pointh_4, pointh_9)
203 edgeh_56 = geompy.MakeEdge(pointh_5, pointh_6)
204 edgeh_58 = geompy.MakeEdge(pointh_5, pointh_8)
205 edgeh_67 = geompy.MakeEdge(pointh_6, pointh_7)
206 edgeh_78 = geompy.MakeEdge(pointh_7, pointh_8)
207 edgeh_89 = geompy.MakeEdge(pointh_8, pointh_9)
208
209 print "making vertical edges"
210 edgev_11 = geompy.MakeEdge(point0_1, pointh_1)
211 edgev_22 = geompy.MakeEdge(point0_2, pointh_2)
212 edgev_33 = geompy.MakeEdge(point0_3, pointh_3)
213 edgev_44 = geompy.MakeEdge(point0_4, pointh_4)
```

```
214 edgev_55 = geompy.MakeEdge(point0_5, pointh_5)
215 edgev_66 = geompy.MakeEdge(point0_6, pointh_6)
216 edgev_77 = geompy.MakeEdge(point0_7, pointh_7)
217 edgev_88 = geompy.MakeEdge(point0_8, pointh_8)
218 edgev_99 = geompy.MakeEdge(point0_9, pointh_9)
219
220 print "making horizontal wires"
221 wire_down_west = geompy.MakeWire([edge0_12, edge0_28, edge0_78, edge0_17
        ],0)
222 wire_up_west = geompy.MakeWire([edgeh_12, edgeh_28, edgeh_78, edgeh_17],0)
223 wire_down_east = geompy.MakeWire([edge0_23, edge0_39, edge0_89, edge0_28
        ],0)
224 wire_up_east = geompy.MakeWire([edgeh_23, edgeh_39, edgeh_89, edgeh_28],0)
225 wire_down_out_west = geompy.MakeWire([edge0_67, edge0_78, edge0_58,
        edge0_56],0)
226 wire_up_out_west = geompy.MakeWire([edgeh_67, edgeh_78, edgeh_58, edgeh_56
        ],0)
227 wire_down_out_east = geompy.MakeWire([edge0_45, edge0_58, edge0_89,
        edge0_49],0)
228 wire_up_out_east = geompy.MakeWire([edgeh_45, edgeh_58, edgeh_89, edgeh_49
        ],0)
229
230 print "making vertical wires"
231 wire_side_southwest = geompy.MakeWire([edge0_12, edgev_22, edgeh_12,
        edgev_11],0)
232 wire_side_southeast = geompy.MakeWire([edge0_23, edgev_33, edgeh_23,
        edgev_22],0)
233 wire_side_east = geompy.MakeWire([edge0_39, edgev_99, edgeh_39, edgev_33
        ],0)
234 wire_side_out_east = geompy.MakeWire([edge0_49, edgev_99, edgeh_49,
        edgev_44],0)
235 wire_side_diagonal = geompy.MakeWire([edge0_45, edgev_55, edgeh_45,
        edgev_44],0)
236 wire_side_out_diagonal = geompy.MakeWire([edge0_89, edgev_99, edgeh_89,
        edgev_88],0)
237 wire_side_north= geompy.MakeWire([edge0_56, edgev_66, edgeh_56, edgev_55
        ],0)
238 wire_side_out_north= geompy.MakeWire([edge0_78, edgev_88, edgeh_78,
        edgev_77],0)
239 wire_side_west= geompy.MakeWire([edge0_17, edgev_77, edgeh_17, edgev_11
        ],0)
240 wire_side_out_west= geompy.MakeWire([edge0_67, edgev_77, edgeh_67,
        edgev_66],0)
241 wire_side_middle= geompy.MakeWire([edge0_28, edgev_88, edgeh_28, edgev_22
        ],0)
242 wire_side_out_middle = geompy.MakeWire([edge0_58, edgev_88, edgeh_58,
        edgev_55],0)
243
```

```
244 print "making horizontal faces"
245 F_down_west = geompy.MakeFace(wire_down_west,1) #1 = planar face
246 F_up_west = geompy.MakeFace(wire_up_west,1)
247 F_down_east = geompy.MakeFace(wire_down_east,1)
248 F_up_east = geompy.MakeFace(wire_up_east,1)
249 F_up_out_west = geompy.MakeFace(wire_up_out_west,1)
250 F_down_out_west = geompy.MakeFace(wire_down_out_west,1)
251 F_up_out_east = geompy.MakeFace(wire_up_out_east,1)
252 F_down_out_east = geompy.MakeFace(wire_down_out_east,1)
253
254 print "making vertical faces"
255 F_side_southwest = geompy.MakeFace(wire_side_southwest,1)
256 F_side_southeast = geompy.MakeFace(wire_side_southeast,1)
257 F_side_east = geompy.MakeFace(wire_side_east,1)
258 F_side_out_east = geompy.MakeFace(wire_side_out_east,1)
259 F_side_diagonal = geompy.MakeFace(wire_side_diagonal,1)
260 F_side_out_diagonal = geompy.MakeFace(wire_side_out_diagonal,1)
261 F_side_north = geompy.MakeFace(wire_side_north,1)
262 F_side_out_north = geompy.MakeFace(wire_side_out_north,1)
263 F_side_west = geompy.MakeFace(wire_side_west,1)
264 F_side_out_west = geompy.MakeFace(wire_side_out_west,1)
265 F_side_middle = geompy.MakeFace(wire_side_middle,1)
266 F_side_out_middle = geompy.MakeFace(wire_side_out_middle,1)
267
268 print "making 3 shells"
269 shell_out_west = geompy.MakeShell([F_down_out_west, F_up_out_west,
       F_side_out_north, F_side_out_middle, F_side_north, F_side_out_west])
270 shell_out_east = geompy.MakeShell([F_down_out_east, F_up_out_east,
       F_side_out_diagonal, F_side_out_middle, F_side_diagonal,
       F_side_out_east])
271 shell_west = geompy.MakeShell([F_down_west, F_up_west, F_side_southwest,
       F_side_middle, F_side_out_north, F_side_west])
272 shell_east = geompy.MakeShell([F_down_east, F_up_east, F_side_southeast,
       F_side_east, F_side_out_diagonal, F_side_middle])
273
274 print "making 3 solids"
275 outlet_west = geompy.MakeSolid([shell_out_west])
276 outlet_east = geompy.MakeSolid([shell_out_east])
277 solid_west = geompy.MakeSolid([shell_west])
278 solid_east = geompy.MakeSolid([shell_east])
279
280 print "making compound_west of solids outlet_west and solid_west"
281 compound_west = geompy.MakeCompound([outlet_west, solid_west])
282 print "glueing compound_west"
283 compound_west = geompy.MakeGlueFaces(compound_west,10**-7,1)
284
285 print "making compound_east of solids outlet_east and solid_east"
286 compound_east = geompy.MakeCompound([outlet_east, solid_east])
```

```
287 print "glueing compound_east"
288 compound_east = geompy.MakeGlueFaces(compound_east,10**-7,1)
289
290 print "making compound_outlet of compound_west and solid compound_east"
291 compound_outlet = geompy.MakeCompound([compound_west, compound_east])
292 print "glueing compound_outlet"
293 compound_outlet = geompy.MakeGlueFaces(compound_outlet,10**-7,1)
294
295 print "adding compound_outlet to study"
296 geompy.addToStudy(compound_outlet,"compound_outlet")
297
298 ### Making inlet quarter of octahedron ###
299
300 print "making points at z=0"
301 point0_1 = geompy.MakeVertex(0., 0., 0.)
302 point0_2 = geompy.MakeVertex(-l/2., 0., 0.)
303 point0_3 = geompy.MakeVertex(-r, 0., 0.)
304 point0_4 = geompy.MakeVertex(-r, -l/2., 0.)
305 point0_5 = geompy.MakeVertex(-l/2., -r, 0.)
306 point0_6 = geompy.MakeVertex(0., -r, 0.)
307 point0_7 = geompy.MakeVertex(0., -(r-w), 0.)
308 point0_8 = geompy.MakeVertex(-l/2., -(r-w), 0.)
309 point0_9 = geompy.MakeVertex(-r, -(l/2.-w), 0.)
310
311 print "making points at z=h"
312 pointh_1 = geompy.MakeVertex(0., 0., h)
313 pointh_2 = geompy.MakeVertex(-l/2., 0., h)
314 pointh_3 = geompy.MakeVertex(-r, 0., h)
315 pointh_4 = geompy.MakeVertex(-r, -l/2., h)
316 pointh_5 = geompy.MakeVertex(-l/2., -r, h)
317 pointh_6 = geompy.MakeVertex(0., -r, h)
318 pointh_7 = geompy.MakeVertex(0., -(r-w), h)
319 pointh_8 = geompy.MakeVertex(-l/2., -(r-w), h)
320 pointh_9 = geompy.MakeVertex(-r, -(l/2.-w), h)
321
322 print "making horizontal edges at z=0"
323 edge0_12 = geompy.MakeEdge(point0_1, point0_2)
324 edge0_17 = geompy.MakeEdge(point0_1, point0_7)
325 edge0_23 = geompy.MakeEdge(point0_2, point0_3)
326 edge0_28 = geompy.MakeEdge(point0_2, point0_8)
327 edge0_39 = geompy.MakeEdge(point0_3, point0_9)
328 edge0_45 = geompy.MakeEdge(point0_4, point0_5)
329 edge0_49 = geompy.MakeEdge(point0_4, point0_9)
330 edge0_56 = geompy.MakeEdge(point0_5, point0_6)
331 edge0_58 = geompy.MakeEdge(point0_5, point0_8)
332 edge0_67 = geompy.MakeEdge(point0_6, point0_7)
333 edge0_78 = geompy.MakeEdge(point0_7, point0_8)
334 edge0_89 = geompy.MakeEdge(point0_8, point0_9)
```

```
335
336 print "making horizontal edges at z=h"
337 edgeh_12 = geompy.MakeEdge(pointh_1, pointh_2)
338 edgeh_17 = geompy.MakeEdge(pointh_1, pointh_7)
339 edgeh_23 = geompy.MakeEdge(pointh_2, pointh_3)
340 edgeh_28 = geompy.MakeEdge(pointh_2, pointh_8)
341 edgeh_39 = geompy.MakeEdge(pointh_3, pointh_9)
342 edgeh_45 = geompy.MakeEdge(pointh_4, pointh_5)
343 edgeh_49 = geompy.MakeEdge(pointh_4, pointh_9)
344 edgeh_56 = geompy.MakeEdge(pointh_5, pointh_6)
345 edgeh_58 = geompy.MakeEdge(pointh_5, pointh_8)
346 edgeh_67 = geompy.MakeEdge(pointh_6, pointh_7)
347 edgeh_78 = geompy.MakeEdge(pointh_7, pointh_8)
348 edgeh_89 = geompy.MakeEdge(pointh_8, pointh_9)
349
350 print "making vertical edges"
351 edgev_11 = geompy.MakeEdge(point0_1, pointh_1)
352 edgev_22 = geompy.MakeEdge(point0_2, pointh_2)
353 edgev_33 = geompy.MakeEdge(point0_3, pointh_3)
354 edgev_44 = geompy.MakeEdge(point0_4, pointh_4)
355 edgev_55 = geompy.MakeEdge(point0_5, pointh_5)
356 edgev_66 = geompy.MakeEdge(point0_6, pointh_6)
357 edgev_77 = geompy.MakeEdge(point0_7, pointh_7)
358 edgev_88 = geompy.MakeEdge(point0_8, pointh_8)
359 edgev_99 = geompy.MakeEdge(point0_9, pointh_9)
360
361 print "making horizontal wires"
362 wire_down_west = geompy.MakeWire([edge0_12, edge0_28, edge0_78, edge0_17
      ],0)
363 wire_up_west = geompy.MakeWire([edgeh_12, edgeh_28, edgeh_78, edgeh_17],0)
364 wire_down_east = geompy.MakeWire([edge0_23, edge0_39, edge0_89, edge0_28
      ],0)
365 wire_up_east = geompy.MakeWire([edgeh_23, edgeh_39, edgeh_89, edgeh_28],0)
366 wire_down_in_west = geompy.MakeWire([edge0_67, edge0_78, edge0_58,
      edge0_56],0)
367 wire_up_in_west = geompy.MakeWire([edgeh_67, edgeh_78, edgeh_58, edgeh_56
      ],0)
368 wire_down_in_east = geompy.MakeWire([edge0_49, edge0_89, edge0_58,
      edge0_45],0)
369 wire_up_in_east = geompy.MakeWire([edgeh_49, edgeh_89, edgeh_58, edgeh_45
      ],0)
370
371 print "making vertical wires"
372 wire_side_southwest = geompy.MakeWire([edge0_12, edgev_22, edgeh_12,
      edgev_11],0)
373 wire_side_southeast = geompy.MakeWire([edge0_23, edgev_33, edgeh_23,
      edgev_22],0)
```

```
374 wire_side_east = geompy.MakeWire([edge0_39, edgev_99, edgeh_39, edgev_33
      ],0)
375 wire_side_in_east = geompy.MakeWire([edge0_49, edgev_99, edgeh_49,
      edgev_44],0)
376 wire_side_diagonal = geompy.MakeWire([edge0_45, edgev_55, edgeh_45,
      edgev_44],0)
377 wire_side_in_diagonal = geompy.MakeWire([edge0_89, edgev_99, edgeh_89,
      edgev_88],0)
378 wire_side_north= geompy.MakeWire([edge0_56, edgev_66, edgeh_56, edgev_55
      ],0)
379 wire_side_in_north= geompy.MakeWire([edge0_78, edgev_88, edgeh_78,
      edgev_77],0)
380 wire_side_west= geompy.MakeWire([edge0_17, edgev_77, edgeh_17, edgev_11
      ],0)
381 wire_side_in_west= geompy.MakeWire([edge0_67, edgev_77, edgeh_67, edgev_66
      ],0)
382 wire_side_middle= geompy.MakeWire([edge0_28, edgev_88, edgeh_28, edgev_22
      ],0)
383 wire_side_in_middle = geompy.MakeWire([edge0_58, edgev_88, edgeh_58,
      edgev_55],0)
384
385 print "making horizontal faces"
386 F_down_west = geompy.MakeFace(wire_down_west,1) #1 = planar face
387 F_up_west = geompy.MakeFace(wire_up_west,1)
388 F_down_east = geompy.MakeFace(wire_down_east,1)
389 F_up_east = geompy.MakeFace(wire_up_east,1)
390 F_up_in_west = geompy.MakeFace(wire_up_in_west,1)
391 F_down_in_west = geompy.MakeFace(wire_down_in_west,1)
392 F_up_in_east = geompy.MakeFace(wire_up_in_east,1)
393 F_down_in_east = geompy.MakeFace(wire_down_in_east,1)
394
395 print "making vertical faces"
396 F_side_southwest = geompy.MakeFace(wire_side_southwest,1)
397 F_side_southeast = geompy.MakeFace(wire_side_southeast,1)
398 F_side_east = geompy.MakeFace(wire_side_east,1)
399 F_side_in_east = geompy.MakeFace(wire_side_in_east,1)
400 F_side_diagonal = geompy.MakeFace(wire_side_diagonal,1)
401 F_side_in_diagonal = geompy.MakeFace(wire_side_in_diagonal,1)
402 F_side_middle = geompy.MakeFace(wire_side_middle,1)
403 F_side_in_middle = geompy.MakeFace(wire_side_in_middle,1)
404 F_side_north = geompy.MakeFace(wire_side_north,1)
405 F_side_in_north = geompy.MakeFace(wire_side_in_north,1)
406 F_side_west = geompy.MakeFace(wire_side_west,1)
407 F_side_in_west = geompy.MakeFace(wire_side_in_west,1)
408
409 print "making 3 shells"
410 shell_in_west = geompy.MakeShell([F_down_in_west, F_up_in_west,
      F_side_in_north, F_side_in_middle, F_side_north, F_side_in_west])
```

```
411 shell_in_east = geompy.MakeShell([F_down_in_east, F_up_in_east,
        F_side_in_diagonal, F_side_diagonal, F_side_in_middle, F_side_in_east])
412 shell_west = geompy.MakeShell([F_down_west, F_up_west, F_side_southwest,
        F_side_middle, F_side_in_north, F_side_west])
413 shell_east = geompy.MakeShell([F_down_east, F_up_east, F_side_southeast,
        F_side_east, F_side_in_diagonal, F_side_middle])
414
415 print "making 3 solids"
416 inlet_west = geompy.MakeSolid([shell_in_west])
417 inlet_east = geompy.MakeSolid([shell_in_east])
418 solid_west = geompy.MakeSolid([shell_west])
419 solid_east = geompy.MakeSolid([shell_east])
420
421 print "making compound_west of solids inlet_west and solid_west"
422 compound_west = geompy.MakeCompound([inlet_west, solid_west])
423 print "glueing compound_west"
424 compound_west = geompy.MakeGlueFaces(compound_west,10**-7,1)
425
426 print "making compound_east of solids inlet_east and solid_east"
427 compound_east = geompy.MakeCompound([inlet_east, solid_east])
428 print "glueing compound_east"
429 compound_east = geompy.MakeGlueFaces(compound_east,10**-7,1)
430
431 print "making compound_inlet of compound_west and compound_east"
432 compound_inlet = geompy.MakeCompound([compound_west, compound_east])
433 print "glueing compound_inlet"
434 compound_inlet = geompy.MakeGlueFaces(compound_inlet,10**-7,1)
435
436 print "adding compound_inlet to study"
437 geompy.addToStudy(compound_inlet,"compound_inlet")
438
439 ### Finally compounding all 4 compounds together ###
440
441 print "making compound"
442 compound = geompy.MakeCompound([compound_solid, compound_mir_solid,
        compound_outlet, compound_inlet])
443 print "glueing compound"
444 compound = geompy.MakeGlueFaces(compound,10**-7,1)
445 print "adding compound to study"
446 geompy.addToStudy(compound,"compound")
447
448 ###Defining faces for different boundary conditions###
449
450 print "initializing faces of solid_west of compound_solid"
451 F_side_north = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l/4.,r,
        h/2.))
452 F_down_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l/4.,r
        /2,0.))
```

```
453 F_up_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l/4.,r/2,h
      ))
454
455 print "initializing faces of solid_east of compound_solid"
456 F_side_diagonal = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l
      /2.+s/2.),(l/2.+s/2.),h/2.))
457 F_side_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(r,l/4.,h
      /2.))
458 F_down_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l/2.+s
      /2.),l/2.,0.))
459 F_up_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l/2.+s
      /2.),l/2.,h))
460
461 print "initializing faces of mir_y_solid_west of compound_mir_solid"
462 mir_y_F_side_north = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-
      l/4.,r,h/2.))
463 mir_y_F_down_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
      /4.,r/2,0.))
464 mir_y_F_up_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
      /4.,r/2,h))
465
466 print "initializing faces of mir_y_solid_east of compound_mir_solid"
467 mir_y_F_side_diagonal = geompy.GetFaceNearPoint(compound, geompy.
      MakeVertex(-(l/2.+s/2.),(l/2.+s/2.),h/2.))
468 mir_y_F_side_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-r
      ,l/4.,h/2.))
469 mir_y_F_down_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(
      l/2.+s/2.),l/2.,0.))
470 mir_y_F_up_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(l
      /2.+s/2.),l/2.,h))
471
472 print "initializing faces of compound_west of compound_outlet"
473 out_F_side_north = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l
      /4.,-r,h/2.))
474 out_F_down_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l
      /4.,-(r-w)/2,0.))
475 out_F_up_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l
      /4.,-(r-w)/2.,h))
476 out_F_down_out_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(
      l/4.,-(r-w/2.),0.))
477 out_F_up_out_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(l
      /4.,-(r-w/2.),h))
478
479 print "initializing faces of compound_east of compound_outlet"
480 out_F_side_diagonal = geompy.GetFaceNearPoint(compound, geompy.MakeVertex
      ((l/2.+s/2.),-(l/2.+s/2.),h/2.))
481 out_F_side_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(r,-(
      l/2.-w)/2.,h/2.))
```

```
482 out_F_side_out_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(
        r,-(l/2.-w/2.),h/2.))
483 out_F_down_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l
        /2.+s/2.),-l/2.,0.))
484 out_F_up_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l/2.+
        s/2.),-l/2.,h))
485 out_F_down_out_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex
        ((l/2.+s/2.),-(l/2.+s/2.-w/2.),0.))
486 out_F_up_out_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex((l
        /2.+s/2.),-(l/2.+s/2.-w/2.),h))
487
488 print "initializing faces of compound_west of compound_inlet"
489 in_F_side_north = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
        /4.,-r,h/2.))
490 in_F_side_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(0.,-(
        r-w)/2.,h/2.)) # middle wall
491 in_F_down_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
        /4.,-(r-w)/2.,0.))
492 in_F_up_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
        /4.,-(r-w)/2.,h))
493 in_F_up_in_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
        /4.,-(r-w/2.),h))
494 in_F_down_in_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-l
        /4.,-(r-w/2.),0.))
495 in_F_side_in_west = geompy.GetFaceNearPoint(compound, geompy.MakeVertex
        (0.,-(r-w/2.),h/2.))  # middle wall
496
497 print "initializing faces of compound_east of compound_inlet"
498 in_F_side_diagonal = geompy.GetFaceNearPoint(compound, geompy.MakeVertex
        (-(l/2.+s/2.),-(l/2.+s/2.),h/2.))
499 in_F_side_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-r,-(
        l/2.-w)/2.,h/2.))
500 in_F_side_in_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-r
        ,-(l/2.-w/2.),h/2.))
501 in_F_down_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(l
        /2.+s/2.),-l/2.,0.))
502 in_F_up_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(l/2.+
        s/2.),-l/2.,h))
503 in_F_down_in_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(
        l/2.+s/2.),-(l/2.+s/2.-w/2.),0.))
504 in_F_up_in_east = geompy.GetFaceNearPoint(compound, geompy.MakeVertex(-(l
        /2.+s/2.),-(l/2.+s/2.-w/2.),h))
505
506 ###Defining face groups###
507
508 print "initializing face groups"
509 group_inlet = geompy.CreateGroup(compound, geompy.ShapeType["FACE"]) #
        inlet
```

```
510 group_externalwall = geompy.CreateGroup(compound, geompy.ShapeType["FACE"
        ]) # walls of octahedron
511 group_frontAndback = geompy.CreateGroup(compound, geompy.ShapeType["FACE"
        ]) # bottom and surface
512 group_outlet = geompy.CreateGroup(compound, geompy.ShapeType["FACE"]) #
        outlet
513 group_baffle = geompy.CreateGroup(compound, geompy.ShapeType["FACE"]) #
        middle wall
514
515 print "adding your faces of choice to the group"
516 geompy.UnionList(group_inlet, [in_F_side_north])
517 geompy.UnionList(group_externalwall,[F_side_east, F_side_diagonal,
        F_side_north, mir_y_F_side_north, mir_y_F_side_diagonal,
        mir_y_F_side_east,in_F_side_east, in_F_side_in_east, in_F_side_diagonal
        , out_F_side_diagonal, out_F_side_out_east, out_F_side_east])
518 geompy.UnionList(group_frontAndback, [F_up_west, F_up_east, F_down_west,
        F_down_east, mir_y_F_down_east, mir_y_F_down_west, mir_y_F_up_east,
        mir_y_F_up_west, out_F_down_west, out_F_up_west, out_F_up_out_west,
        out_F_down_out_west, out_F_down_east, out_F_up_east,
        out_F_down_out_east, out_F_up_out_east, in_F_down_west, in_F_up_west,
        in_F_down_east, in_F_up_east, in_F_down_in_west, in_F_up_in_west,
        in_F_down_in_east, in_F_up_in_east])
519 geompy.UnionList(group_outlet, [out_F_side_north])
520 geompy.UnionList(group_baffle, [in_F_side_west, in_F_side_in_west])
521
522 print "adding the groups to the study (IN OBJECT compound !!!)"
523 # IMPORTANT: choose the names equal to the boundaries in OpenFoam
524 geompy.addToStudyInFather(compound, group_inlet, "inlet")
525 geompy.addToStudyInFather(compound, group_externalwall, "externalwall")
526 geompy.addToStudyInFather(compound, group_frontAndback, "frontAndback")
527 geompy.addToStudyInFather(compound, group_outlet, "outlet")
528 geompy.addToStudyInFather(compound, group_baffle, "baffle")
529
530 ###Getting edges for local mesh size###
531
532 E_x_inlet_west = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(-l
        /4., 0., 0.))
533 E_x_outlet_west = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(l
        /4., 0., 0.))
534 E_x_inlet_east = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(-(l
        /2.+s/2.), 0., 0.))
535 E_x_outlet_east = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex((l
        /2.+s/2.), 0., 0.))
536 E_y_north = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(0, r/2.,
        0.))
537 E_y_south = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(0., (-(r-w
        )/2.), 0.))
```

```
538 E_y_inlet = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(0., -(r-w
      /2.), 0.))
539 E_z = geompy.GetEdgeNearPoint(compound, geompy.MakeVertex(0., 0., h/2.))
540
541 #Create a hexahedrical mesh
542 hexa_compound = smesh.Mesh(compound, "compound: hexahedrical mesh")
543
544 # create a Regular 1D algorithm for edges
545 algo1D_compound  = hexa_compound.Segment()
546
547 # create a quadrangle 2D algorithm for faces
548 algo2D_compound = hexa_compound.Quadrangle()
549
550 # create a hexahedron 3D algorithm for solids
551 algo3D_compound = hexa_compound.Hexahedron()
552
553 # define hypotheses
554 # global mesh hypotheses
555 algo1D_compound.NumberOfSegments(int(n_y_south*r))
556
557 # local mesh hypotheses
558 #local mesh outlet x-direction
559 algo_local_x = hexa_compound.Segment(E_x_outlet_west)
560 # set number of segments
561 algo_local_x.NumberOfSegments(int(n_x_outlet_west*l/2.))
562 # propagate through whole reservoir
563 algo_local_x.Propagation()
564 print "dividing E_x_outlet_west in %g segments" %(int(n_x_outlet_west*l
      /2.))
565 print "with a length in x-direction of l(x)= %g m" %(float(l/2./int(
      n_x_outlet_west*l/2.)))
566
567 algo_local_x = hexa_compound.Segment(E_x_outlet_east)
568 # set number of segments
569 algo_local_x.NumberOfSegments(int(2.*s/((1./n_x_outlet_west)*(1.+cellratio
      ))),cellratio)
570 # propagate through whole reservoir
571 algo_local_x.Propagation()
572 print "dividing E_x_outlet_east in %g segments" %(int(2.*s/((1./
      n_x_outlet_west)*(1.+cellratio))))
573 print "with a length in x-direction of l(x)= %g m" %(float(s/int(2.*s
      /((1./n_x_outlet_west)*(1.+cellratio)))))
574
575 #local mesh inlet x-direction
576 algo_local_x = hexa_compound.Segment(E_x_inlet_west)
577 # set number of segments
578 algo_local_x.NumberOfSegments(int(n_x_inlet_west*l/2.))
579 # propagate through whole reservoir
```

```
algo_local_x.Propagation()
print "dividing E_x_inlet_west in %g segments" %(int(n_x_inlet_west*l/2.))
print "with a length in x-direction of l(x)= %g m" %(float(l/2./int((
    n_x_inlet_west*l/2.))))

algo_local_x = hexa_compound.Segment(E_x_inlet_east)
# set number of segments
algo_local_x.NumberOfSegments(int(2.*s/((1./n_x_inlet_west)*(1.+cellratio)
    )),cellratio)
# propagate through whole reservoir
algo_local_x.Propagation()
print "dividing E_x_inlet_east in %g segments" %(int(2.*s/((1./
    n_x_inlet_west)*(1.+cellratio))))
print "with a length in x-direction of l(x)= %g m" %(float(s/int(2.*s
    /((1./n_x_inlet_west)*(1.+cellratio)))))

#local mesh north y-direction
algo_local_y = hexa_compound.Segment(E_y_north)
# set number of segments
algo_local_y.NumberOfSegments(int(2.*r/((1./n_y_south)*(1.+cellratio))),
    cellratio)
# propagate through whole reservoir
algo_local_y.Propagation()
print "dividing E_y_north in %g segments" %(int(2.*r/((1./n_y_south)*(1.+
    cellratio))))
print "with a length in y-direction of l(y)= %g m" %(float(r/(int(2.*r
    /((1./n_y_south)*(1.+cellratio))))))

#local mesh south y-direction
algo_local_y = hexa_compound.Segment(E_y_south)
# set number of segments
algo_local_y.NumberOfSegments(int(n_y_south*(r-w)))
# propagate through whole reservoir
algo_local_y.Propagation()
print "dividing E_y_south in %g segments" %(int(n_y_south*(r-w)))
print "with a length in y-direction of l(y)= %g m" %(float((r-w)/int((
    n_y_south*(r-w)))))

#local mesh inlet y-direction
algo_local_y = hexa_compound.Segment(E_y_inlet)
# set number of segments
algo_local_y.NumberOfSegments(int(n_y_south*w))
# propagate through whole reservoir
algo_local_y.Propagation()
print "dividing E_y_inlet in %g segments" %(int(n_y_south*w))
print "with a length in y-direction of l(y)= %g m" %(float(w/int((
    n_y_south*w))))

```

```
619 #local mesh z-direction
620 algo_local_z = hexa_compound.Segment(E_z)
621 # set number of segments
622 algo_local_z.NumberOfSegments(int(n_z*h))
623 # propagate through whole reservoir
624 algo_local_z.Propagation()
625 print "dividing E_z in %g segments" %(int(n_z*h))
626 print "with a length in z-direction of l(z)= %g m" %(float(h/int((n_z*h)))
      )
627
628 print "Computing the mesh ..."
629 # compute the mesh
630 hexa_compound.Compute()
631
632 hexa_compound.GroupOnGeom(group_inlet, "inlet",SMESH.FACE)
633 hexa_compound.GroupOnGeom(group_externalwall,"externalwall",SMESH.FACE)
634 hexa_compound.GroupOnGeom(group_frontAndback, "frontAndback",SMESH.FACE)
635 hexa_compound.GroupOnGeom(group_outlet, "outlet",SMESH.FACE)
636 hexa_compound.GroupOnGeom(group_baffle, "baffle",SMESH.FACE)
```