university of groningen

faculty of science and engineering

CFD modelling of a free surface flow in the barrier of the Ostend harbour with OpenFOAM

Witteveen+Bos

Internship MSc Applied Mathematics

November 2019

Student: S.I.O. Scholte

Company/Institute: Witteveen+Bos, Deventer

Internal UoG supervisor/first assessor: Prof.dr.ir. R.W.C.P. Verstappen

External supervisor: Dr.ir. A.C. de Niet

Internal UoG second assessor: Dr.ir. R. Luppes

Abstract

This report will contain the results of modelling a free surface flow to calculate the loss coefficient of the sluice in the barrier of Ostend. These simulations were performed with the open source CFD software OpenFOAM. During the process of computing a free surface flow in OpenFOAM we encountered several problems. These problems were mainly to keep the water level stable at the outflow boundary and to get a good looking flow profile, because we also needed to introduce backflow. These problems were mainly created by the choices of boundary conditions at the outflow boundary. Next to solving these problems, we also made a simulation with a rigid lid condition where we see that the loss coefficient is lower than the value that was found in all the free surface models. At last, a better design of the sluice is proposed that decreases the loss coefficient drastically.

Contents

1 Introduction

Witteveen+Bos is involved in a flooding protection project in Ostend. The coast needs to be strengthened and some of the water constructions in the harbour will be rebuilt. One of these constructions is the barrage that connects the harbour with a huge water basin. The two water bodies exchange water during tides such that the water in the water basin gets freshened. The exchange of water is done via a sluice. This sluice will be designed by Witteveen+Bos and needs to meet certain conditions regarding the water level, rate of refreshment and discharge rate.

Figure 1.1: The Ostend harbour

To get a feeling how efficient the design of the sluice is, we can use the empirical relation between the discharge through the sluice and the difference in water height of the two water bodies on both sides of the sluice:

$$Q = \mu A \sqrt{2g(h_S - h_A)}. \tag{1}$$

Here, Q is the discharge (m^3/s), μ is the contraction coefficient (-), A is the surface of the cross section of the pipe (m^2), g is the gravitational acceleration (m/s^2) and h_A, h_S (both m) are the water heights of the tide driven harbour and the water basin respectively. This contraction coefficient μ depends on the design of the sluice. The more the flow contracts at the inflow of the sluice, the more energy will be lost in the system. This loss is expressed in the loss coefficient ξ with the relation $\xi = (1 - \frac{1}{\mu})^2$.

Witteveen+Bos uses CFD to calculate the loss coefficient of the design. The calculations are done with the commercial CFD software COMSOL Multiphysics. In their model, they make use of a rigid lid assumption. This means that the effects of the atmosphere are ignored and only water is considered with a free slip condition at the top of the geometry. This model calculated a loss coefficient of $\xi = 0.071$. A more realistic model would be to consider the flow as a free surface flow, where the water level can vary and waves can occur due to the water flow. This free surface flow is a lot more challenging due to a more complex set of equations that has to be solved. Different boundary conditions have to be imposed and to track the interface between air and water we need to add an extra equation to our set of Navier-Stokes equations.

Although COMSOL Multiphysics is able to compute free surface flow, the company wants to do the calculations in OpenFOAM, an open source software program for CFD calculations. Customers of Witteveen+Bos sometimes demand calculations in OpenFOAM. Therefore it is important that they get more familiar with the capabilities of OpenFOAM. The internship consists therefore of two main elements. The first part of this internship is getting to understand OpenFOAM, perform free surface flow simulations with it and compare the results with a model that uses a rigid lid assumption. The second objective is to document the use of OpenFOAM and their most useful capabilities regarding the computations of free surface flows.

Our goal then is to answer the main question whether it is necessary to use a free surface model for the calculation of the loss coefficient, or do rigid lid models suffice. The outline of the chapters that will help answering this question is presented below.

We start this report with chapter 2, where we describe the preparation process in the first weeks of the internship. This consists at first of an overview of the project in Ostend. Next we will cover the software programs we will use and we will explain the most important properties of free surface flows. We end the section with an overview of the weekly schedule of the internship.

Chapter 3 will consist of describing the setup of the free surface model. For this, we used the model Witteveen+Bos made in COMSOL as starting point. In this section we start to describe the process of building the geometry with the software SALOME. Next we give the equations together with the boundary conditions belonging to modelling a free surface flow and how OpenFOAM solves these equations. From these equations, we sketch how we made the mesh accordingly for this model. Throughout this section, we mention the challenges we are about to expect. These include getting a stable solution near the outflow boundary and the choice of boundary conditions with OpenFOAM in general.

The following chapters are about showing the results of different OpenFOAM models. Some of them describe a different situation with respect to the geometry or water tide, others will describe a change in boundary conditions to improve the flow. We start with chapter 4 where we perform a simulation of the initial setup. We show the results of the flow and notice that the current outflow boundary conditions lead to a water drop. This issue forces us to change our conditions and geometry in our future models. We describe the change of having separated outflow boundaries for air and water in chapter 5, such that we can fix the water level at the outflow boundary. Although there are still disturbances at the end of our domain, the change to two boundaries has some promising results and make us able to analyse the flow. The next step is to simulate a rigid lid model in order to discuss the results of both models. The rigid lid model is covered in chapter 6. We will see that the loss in the rigid lid situation is a bit lower.

As we had still some strange behaviour at the end of our domain, we we're trying to find ways to stabilize the flow near the outflow boundary. These stabilizing models are described in chapter 7. The first approach was to make the outflow boundary a lot smaller. Moreover, we placed some pillars in front of that boundary such that the flow is almost completely damped out. The flow is a lot more stable, but the water level at the harbour side starts to rise because the water can't flow out very easy. Our second model was to prescribe a small pressure gradient at the water outflow boundary to force water to flow out and prevent strange backflow behaviour. This model also leads to a stable flow pattern, but it takes more time for the loss coefficient to get more stable.

The last simulations we have performed were a model with a water height corresponding with the Lowest Astronomical Tide and a model with optimized inflow conditions for the sluice with round edges instead of sharp edges. We cover these models in chapters 8 and 9 respectively. We will see that changing the water height at the harbour doesn't influence the simulation very much. We will also see that the improved geometry for the inflow of the sluice leads to a much lower loss coefficient.

We end the report with a discussion of the results of the different simulations. Then we are able to answer our main research question. We will give a recommendation of work that can be done in the future to optimize this study. The documentation of the software OpenFOAM can be found in the first appendix.

1.1 Witteveen+Bos

Witteveen+Bos is an engineering and consultancy company with 21 locations spread over 11 countries. They have over 1000 employees and seven of their offices are in the Netherlands. Witteveen+Bos has four main business lines: 'Delta, Coasts and Rivers', 'Built Environment', 'Energy, Water and Environment' and 'Infrastructure and Mobility'. Each business line is subdivided into smaller Product-Market Combinations (PMC's) who have around 30-40 employees each. Those PMC's have to take care of their own project acquisition and contract management.

During my internship, I was placed in the 'Data Analysis and Supply Information' group of the PMC 'Coasts, Rivers and Land Reclamation'. The group consists of 7 people, who work on projects regarding data validation and analysis. My workplace was flexible, as they work with the so called 'Pluswerken'. This means that there is a wide variety of workplaces that don't belong to a certain employee. Every day you can choose the workplace that fits best to your tasks for that day.

2 Preparation

The main goal of the internship is to create a model in OpenFOAM for the water flow through the barrier in Ostend. Before we start with this specific assignment, we need to get a better understanding of the whole project and the software we will use. In this section we will start with an overview of the project in Ostend. After that we give a brief introduction of the software programs OpenFOAM, SALOME and ParaView. At last we give a schematic overview of the timeline of the internship.

2.1 The project

The internship is a part of a big project about the flood protection in the harbour of Ostend (Belgium). As a consequence of all these new protection measures, a new water outflow facility from the big water basin to the harbour has to be built. Witteveen+Bos is currently busy with the sketch phase of the construction of the dam. For the sluice inside the dam, they have to meet certain conditions. At first, the water level in the basin can't fluctuate too much. Due to activities and sports organised in the basin, the water level margins are set to a minimum water level of 2.97 TAW (Belgian reference height) and a maximum level of 3.37 TAW.

During each tide the sluice has to be opened. At high tide, water will flow from the harbour to the water basin and during low tide water will flow the other way around. Another request for the design is that the time of total refreshment of the water basin is minimized. As a result, we need to realize the maximum outflow during a tidal cycle, without going below the water level of 2.97 TAW in the water basin. With this information, Witteveen+Bos already derived that a surface of the sluice of 10m^2 would suffice in freshening the water basin to its fullest during a tidal cycle.

Witteveen+Bos chose to divide one sluice with an area of 10m^2 into two smaller sluices of 5m^2. In this way, the hindrance for the ships in the harbour will be minimized as the maximal instantaneous flow rate gets lowered. The construction costs of smaller sluices are also lower [1].

2.2 Software

For the CFD computations we will use the Open-source Field Operation and Manipulation, OpenFOAM in short, software. The version we used is OpenFOAM version 7. The OpenFOAM toolbox is written in C++ and is primarily used to solve CFD problems. Besides, it comes with handy pre- and post-processing utilities.

The geometry of the models is built in the open source CAD software SALOME, version 9.3.0. SALOME is able to construct complex designs and it has some utilities which are beneficial to the CFD simulations in OpenFOAM.

For the visualization of the flow and the mesh, we use the open source software ParaView. version 5.6.0. ParaView is a visualization software which has a strong connectivity with OpenFOAM. It has many options to get a good look at your solution and it is also able to perform different kind of analysis on it.

A detailed overview of how to use these software programs can be found in appendix A.

2.3 Free surface flows

The main type of flow we are dealing with in this project is a free surface flow. The free surface will be the boundary between the air in the atmosphere and the water at the water basin and the harbour. This boundary is physically subjected to a homogeneous perpendicular stress and a homogeneous parallel shear stress. The first condition means that the normal forces of both fluids onto that boundary layer are equal in magnitude and opposite in direction. The second condition states that both the magnitude as the direction of the parallel forces at the boundary layer of both fluids are equal.

The boundary layer between both fluids is normally a thin, sharp layer. The fact that the location of the boundary layer constantly changes over time, makes it computationally very challenging to capture. Several methods have been created to capture the location of the interface. For a two-phase flow, these methods are not made to compute a sharp interface layer as it would be very demanding in terms of mesh refinement. Instead, the goal is to compute the shape of the interface with all the cells near the interface.

2.4 Timeline

During the internship, a lot of different steps have been taken to setup and improve the model of a free surface flow. Below is a brief overview of the weekly schedule.

Week 1
The goal of the first week was primarily about getting to know Witteveen+Bos and the project I was helping with. Therefore I spent most of the time reading material about the development of the project. Also I started with reading about OpenFOAM and getting used to the software. At the last day of the week I had a meeting about the goals they want me to achieve during the internship period, so from that week forward I could really getting busy with my assignment.

Week 2
The main focus of this week was the use of OpenFOAM. An OpenFOAM project consists of a lot of files and to be able to perform a CFD computation it is important to know what all files and their contents are meant for. Furthermore I tried to find a program suitable for creating more complex geometries and found one in the form of SALOME. In this program I succeeded to rebuild the geometry Witteveen+Bos already made in COMSOL and I was able to perform my first simulations on it with OpenFOAM with a course grid.

Week 3
After the first CFD computation, there arose some points of improvements to work on this week. The first point of attention was the mesh. To create a finer mesh with in particular a finer boundary layer, I had several options. Two examples are using SALOME for the meshing as well and the snappyHexMesh tool of OpenFOAM itself. After experimenting with both options, snappyHexMesh became the method of use due to the in-built quality checks such that we can make sure it is approved by OpenFOAM. For the snappyHexMesh utility, I needed to enter all the parameters for mesh refinements myself in an additional C++ dictionary file for snappyHexMesh.

Week 4
The fourth week was about learning on how to perform simulations on the internal cluster network of Witteveen+Bos. On this cluster we have access to multiple processor cores such that our calculations can be performed much faster. After the good version of OpenFOAM was installed on the cluster and I was helped with how to start a simulation, I was able to perform my first calculations on it.

Week 5
After my first simulations, there were a few issues in the flow caused by the outflow boundary. The water level dropped after the velocity jet hit the outflow boundary. Due to our chosen boundary conditions, air came in to the model on the top of the outflow boundary which 'pushed' the water downwards. I discussed ways to overcome this with my supervisors and I made a start with building a new geometry with two separate outlets (one for air, one for water).

Week 6
After building the new geometry with the separate outlet, I spent a good amount of time meshing and simulating the new model. I made a few runs to get the boundary conditions right and got a working, two minute simulation of the flow. The rest of the time was spent on working on the report and the OpenFOAM documentation.

Week 7
I was able to perform a longer simulation and did calculations on the flow regarding the loss coefficient. Although the simulation looks good, there are some disturbances at the end of our geometry. The outflow boundary is still responsible for this kind of behavior. Apart from the last meters of the geometry, the simulation looks pretty good and I decided to analyse the data of the flow. I also followed an OpenFOAM course at the TU Delft with my supervisor and learned a few new things about the OpenFOAM code.

Week 8
To analyse the flow, I made a Python script that reads data extracted from ParaView and OpenFOAM to calculate the loss coefficient and plot it against the time. Furthermore, I did a new simulation regarding a rigid lid condition to compare the outcome with the free surface model. I made use of a different OpenFOAM solver and had to adjust a few settings, such as the boundary conditions for the upper faces.

Week 9
I started this week with trying to get a more stable solution of the the flow at the end of the boundary. The first option I worked on, was to narrow the outflow boundary and to place some pillars in the water in front of that boundary. Because of that, the velocity jet would be damped almost fully, which leads to less backflow disturbances. The second option was to impose a small pressure gradient condition on the outflow boundary that force the water to flow out. Bot options worked really well.

Week 10
This week I performed the last simulations. At first I did a simulation on a model with a low water tide to look whether that makes a big difference. The final simulation was with a changed geometry, where the inflow part of the sluice was rounded such that water would flow into the sluice without a lot of contraction. The rest of the time was making sure all the results were uniform presented and we used some time to work on the presentation.

Week 11
The last week was all about the report and presentation. I needed to write the last few parts like the discussion and abstract. Also I finished my presentation and held it at Thursday. The report was finished at Wednesday.

3 Setup of the model

The next chapter will be about the setup of a free surface CFD model in OpenFOAM. As a base we use most of the settings of an earlier model Witteveen+Bos made in COMSOL Multiphysics. COMSOL Multiphysics is a commercial CFD software that the company has a license for.

The first step of the setup is making the geometry. For that, we mimic the geometries from the COMSOL model. The geometry is relatively simple, where water will flow from a box of water through a sluice to the other box of water. After the geometry, we will inform you about the solver inside OpenFOAM that can deal with multiphase flows. This solver solves the Navier-Stokes equations alongside an extra equation for the interface between water and air iteratively. The boundary conditions have to be chosen appropriately to make the system of equations solvable. Besides, the boundary conditions have to belong to a free surface flow which will be different to a rigid lid model obviously. Together with the wide range of different boundary conditions you can choose from in OpenFOAM, makes choosing the appropriate boundary conditions a hard task.

At last we need to make the mesh for our model according to the set of boundary conditions we have imposed. This mesh is made with the OpenFOAM in-built utility snappyHexMesh and this result will be shown at the end of this chapter. We refer to appendix A to look how all these actions are incorporated into OpenFOAM.

3.1 Geometry

The process of constructing the geometry in SALOME will be explained in this section. As already mentioned, we used the same geometry as the earlier COMSOL model. This geometry is a simplified model compared to the actual situation in Ostend. The geometry describes a model where water flows from a big box of water through a pipe to another box.

The software to build the geometry is SALOME, version 9.3.0. The dimensions of the three parts of the geometry are given in table 1.

Part	Dimensions
Water basin	15m x 10m x 5m
Sluice	2,5m x 34.5m x 2m
Harbour	20m x 28.5 x 5m

Table 1: Dimensions of the different parts

We see that the area of the cross section of the sluice is 5m^2. The location of the sluice is 0,5 meters above the ground of the water basin. Also, the water basin lies one meter higher than the harbour, which means that the sluice outflow is 1,5 meter above the ground of the harbour. Furthermore, a talud will be placed at the outflow of the sluice. This means that the sluice and harbour overlap in our geometry. Figure 3.1 and figure 3.2 show the whole geometry from two perspectives, where you get a good view of how the harbour and sluice overlap in the latter figure. Note that we let the water flow in the Y-direction.

All patches belonging to the geometry are part of one of the six boundary groups. The front of the water basin is named the 'inlet'. The back of the harbour is the 'outlet'. The top faces of the water basin and the harbour is the 'atmosphere'. The slope is called the 'talud'. The walls of the sluice

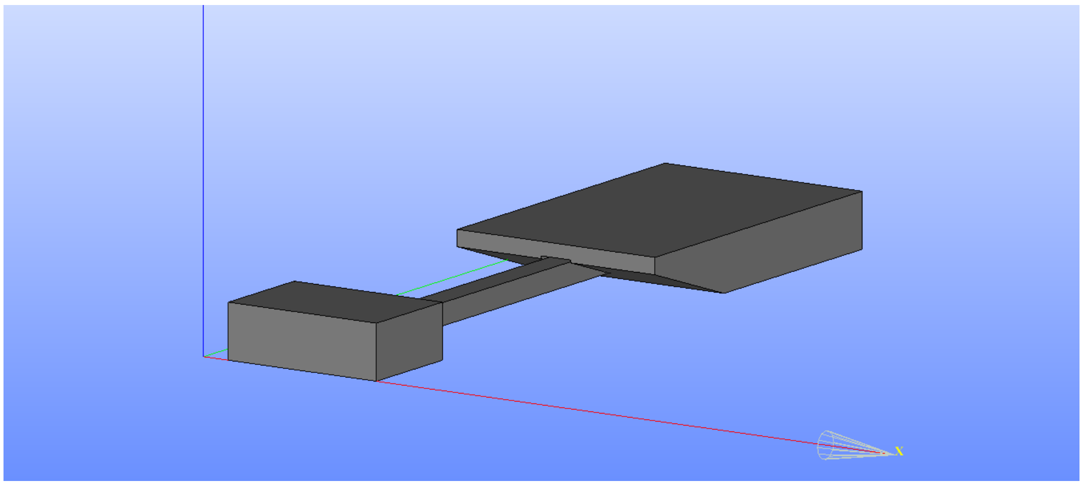


Figure 3.1: Geometry of the model viewed from the front

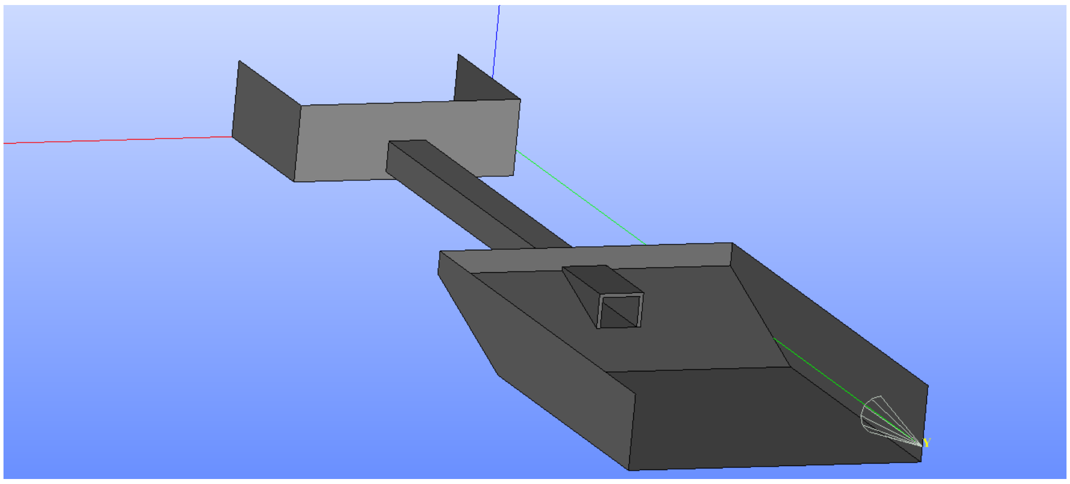


Figure 3.2: Geometry of the model viewed from the back without atmosphere, inlet and outlet faces

and the back of the water basin which is attached to the sluice is the 'concretewall' and all the other patches are just called 'wall'. With these boundary patches grouped, we can impose the appropriate boundary conditions on them.

3.2 The flow equations

The flow equations determine the character of the flow simulation. In this section we give the equations OpenFOAM uses to compute a free surface multiphase flow. Also which discretization methods and solvers it uses will be treated here.

The base of any fluid flow equation are the Navier-Stokes equations, which are given by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0 \tag{2}$$

and

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\, \boldsymbol{u}) = -\nabla p + \rho g + \nabla \cdot \tau. \tag{3}$$

Equation 2 describes the conservation of mass, whereas equation 3 describes the conservation of momentum. Here is ρ the density (kg/m^3), $\boldsymbol{u}$ the velocity vector (m/s), t is the time (s), p the pressure ($kg/m\,s^2$), g the gravitational acceleration (m/s^2) and τ the stress tensor ($kg/m\,s^2$).

In the incompressible case, density ρ is constant and equation 2 simplifies to

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4}$$

a divergence free velocity field condition.

Because we are modelling a free surface flow, we need to model the surface tension F (kg/m^2s^2) as well. We incorporate this into equation 3 as

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\, \boldsymbol{u}) = -\nabla p + \rho g + \nabla \cdot \tau + F. \tag{5}$$

This surface tension is defined as

$$F = \sigma \kappa \frac{\partial \alpha}{\partial x}, \tag{6}$$

where σ is the surface tension constant (kg/s^2) and κ is the curvature (m^{-1}), which can be approximated as $\kappa = -\frac{\partial n}{\partial x}$. The additional parameter α for the phase of the fluid is defined as

$$\alpha = \begin{cases} 0, & \text{if particle is in the other fluid} \\ 0 < \alpha < 1, & \text{at the interface} \\ 1, & \text{if particle is in the main fluid.} \end{cases} \tag{7}$$

In our case, the main fluid will be water and the other fluid will be air. OpenFOAM uses a particular method to capture the interface between the fluids, which is called the Volume Of Fluid (VOF) method. The VOF method calculates the ratio of each cell filled with the liquid phase by solving

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{u}) = 0. \tag{8}$$

With α we can define our density ρ as

$$\rho = \alpha \rho_0 + (1 - \alpha)\rho_1,$$

where ρ_0, ρ_1 are the densities of the water and air respectively. At last, equation 5 can be simplified if we consider a hydrostatic pressure contribution. The relation between the static pressure and the alternative pressure is

$$p' = p - \rho g h,$$

where p' is the new static pressure. In OpenFOAM, this pressure is named p_rgh. When we plug this relation in equation 5, we get

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\, \boldsymbol{u}) = -\nabla p' + \nabla \cdot \tau + F. \tag{9}$$

The algorithm OpenFOAM uses to solve equations 4, 8 and 9 is called interFoam.

Additionally, we want to visualize a turbulent flow. Therefore we need to incorporate a turbulence model into our OpenFOAM environment. We have several options here, for example a k-ϵ or k-ω turbulence model. We have chosen to use a combination of the two, namely $k - \omega$ SST model. SST stands for Shear Stress Transport. This model uses the k-ϵ in the free stream, which has good results overall. Near the wall, at the viscous boundary layers, it switches to a k-ω model. Both models work very good in their respective regions, so the $k - \omega$ SST model captures the best bits of both turbulence models [2].

It is commonly known that the Navier-Stokes equations can't be solved analytically. Therefore, OpenFOAM uses the Finite Volume Method to discretize all the terms in order to solve the equations. From there, we use the PIMPLE algorithm to solve for the variables in every time step. The PIMPLE method is a combination of the SIMPLE and the PISO algorithm and uses the advantages of both methods to create a more stable solution, even at larger time steps [3].

3.3 Boundary Conditions

After we have completed our geometry and assigned our boundaries, we can set our boundary conditions. For every variable in our equations, we need to put a boundary condition on each boundary group. OpenFOAM has a big list of boundary conditions from which you can choose from.

Some of those boundary conditions influence our choices we make for the mesh. This include mainly the no slip condition we want on the 'concretewall' patches and the talud. To really capture the effects of this no slip condition, we need to implement boundary layers at these boundaries such that the flow will not stick at the wall.

The boundary condition at the other 'walls' will be a free slip boundary. Furthermore we want to fix the flow rate of the water at the inflow boundary. Because we have so many options, it is hard to impose the right boundary conditions immediately. Therefore all the other boundary conditions were copied from already existing OpenFOAM tutorials [4]. In table B.1 of appendix B.1 are the boundary conditions we have chosen for this model setup.

Especially the search for the right boundary conditions at the outflow boundary will be a difficult task. The reason for this is that we need to force some kind of behaviour to stabilize the water level, as this outflow boundary must represent the sea. So one of our biggest challenges is to find a set of boundary conditions on this side that it does influence the behaviour of the flow as less as possible.

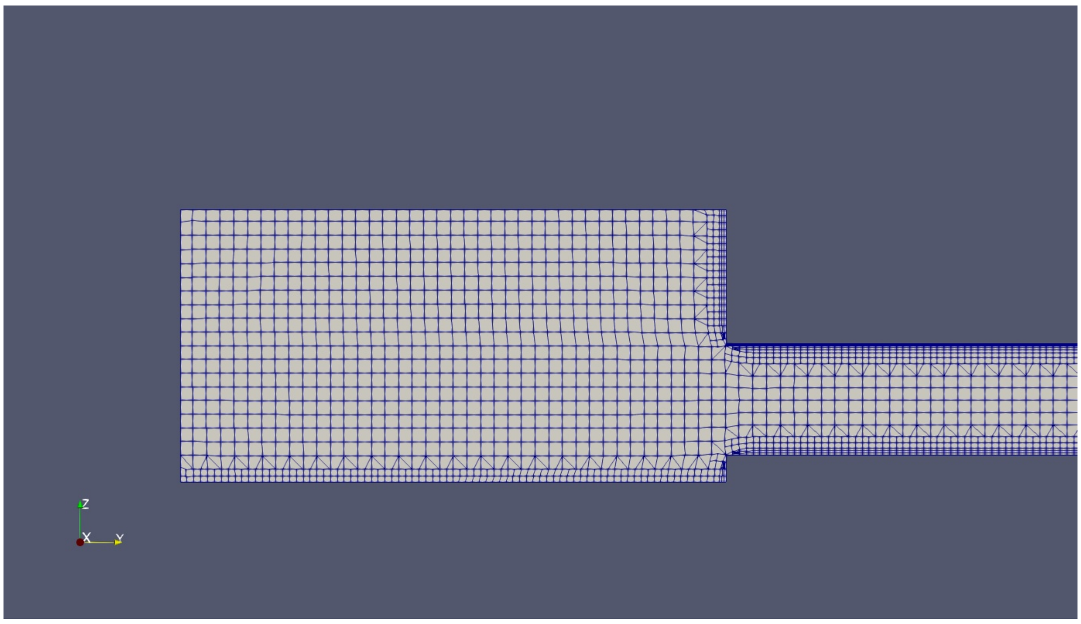


Figure 3.3: Slice of the mesh in the Y-Z plane at the inlet of the sluice

3.4 Meshing

The mesh is a very important part in obtaining a working and accurate solution. Meshing is always a trade-off between accuracy and computation time. Therefore we want to have a relative course mesh at regions that don't influence our accuracy very much, but a much finer mesh near some high influenced regions. An essential property we already mentioned is to include boundary layers around the concrete wall sections such that the water doesn't 'stick' to the wall. Furthermore we refined the mesh a bit more near the other walls. A slice of the mesh can be seen in figures 3.3 and 3.4. The mesh is created with the snappyHexMesh tool of OpenFOAM. Please note that the rendering

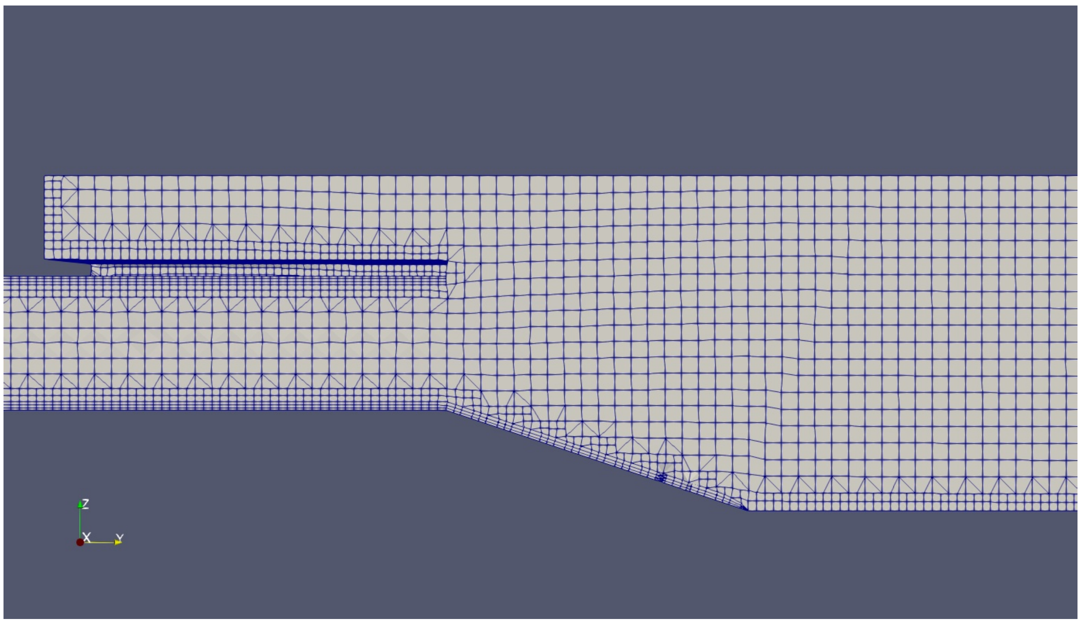


Figure 3.4: Slice of the mesh in the Y-Z plane at the outlet of the sluice

of the mesh in ParaView can show some strange shapes or discontinuities at a few points.

4 Default model

The model setup from the previous chapter will now be simulated. We will provide the initial settings of our computations and present the results of the solution. We will then discuss the points we will need to focus on in our future models.

4.1 Initial settings

After all numerical settings have been set, we have to initialize the controls and settings for the simulation. First of all, we define the initial field for the fluid phases. For simplicity, we assume a uniform water level over the whole domain at the start of the simulation. The water level we chose is 3.5 meter above the ground such that the sluice is completely under water, but both the water basin as the harbour has a layer of air at the top of their respective domains.

We chose to run the simulation for 60 seconds. The time step is made adjustable to the mesh size and maximum Courant number, which is restricted to 1 in our case. Although we use the stable PIMPLE algorithm, which consists of the SIMPLE algorithm that formally doesn't need a CFL-condition, it is a safe restriction.

4.2 Results

Figure 4.5 and 4.6 show the results of the simulation at the first and last time step concerning the water phase α. The water level drop is something that immediately stands out. The main reason for this is the boundary condition at the outlet boundary. Because we can't control the water height and we have a zero gradient condition for the velocity, water flows out but is compensated with air coming into the model. This stimulates the water to flow out even harder, such that the harbour

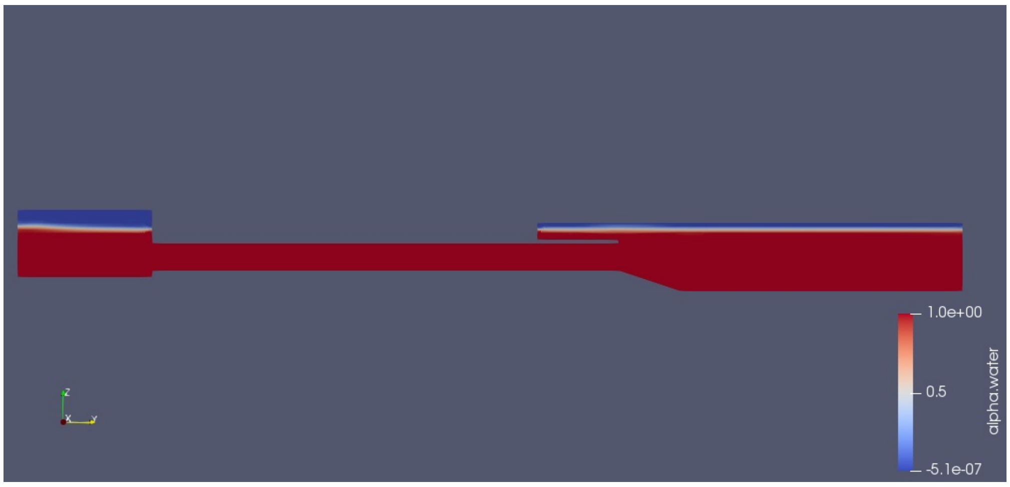


Figure 4.5: The water level at t = 1s

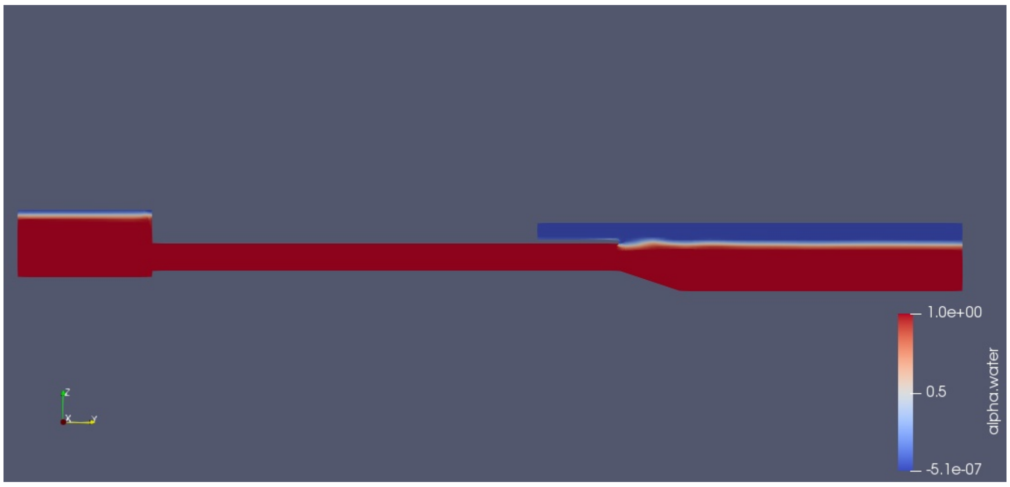
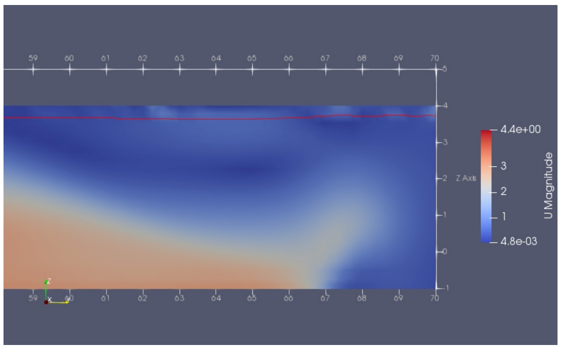


Figure 4.6: The water level at t = 60s

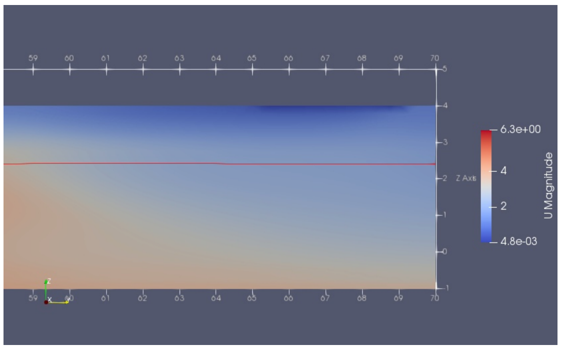
loses all its water.

When looking closely to the moment this happens, we see that from the moment the velocity jet hits the outflow boundary, the water level is dropping. In figure 4.7 we see two different moments of the simulation, where the red line is the water level. We see that at 30 seconds, the velocity jet hasn't reached the end of the domain yet and the water level is at a normal level. Between this time and the end of the simulation, the flow has reached the end of the domain and has been flowing out. Due to the water drop, we chose not to do an analysis on the loss coefficient of this model.

Looking forward to what we can do differently to maintain the water level, an option would be to fix the water level by splitting the outflow boundary into two separate boundaries of the two different phases. Then we can impose boundary conditions on both boundaries separately which will hopefully lead to a stable solution with a solid water level. This is what we are going to study in the next model.



(a) Velocity jet at $t = 30$s



(b) Velocity jet at $t = 60$s

Figure 4.7: Water level drop when the velocity jet hits the outflow boundary

5 Model with separate outlet boundaries

Based on our previous model, we created a new, more realistic model and used our prior obtained knowledge to compute a better simulation. We took the main focus points learned from our initial OpenFOAM simulation to try to get a more stable solution of our flow.

At first, we adapted the geometry of the model which is more realistic to the situation in Ostend [1]. The biggest difference is a few changes in the dimensions. Moreover, we changed the shape of the talud a little. We also make the geometry 30 meters longer to let the flow damp out more.

Next, we have to pay attention to the outflow boundary conditions. As seen in our initial model, the current conditions create such instabilities that it has a big impact on the water level. The first adaptation we did was the splitting of the outflow boundary in a separate air and water boundary. With two outflow boundaries instead of one, we could fix the value of the phase to their respective boundary. At last, we played a little bit with some other boundary conditions to end up with a stable solution.

In the results section, we give an overview of the results of this simulation and discuss them. We analyse the convergence of the loss coefficient ξ by means of the other variables.

5.1 Geometry

To increase our chances of a stable solution, we made some changes in the geometry of the model. Also we made the geometry look more similar to the actual construction in the harbour of Ostend.

At first we changed the dimensions of whole region. We made the boxes that represent the water basin and the harbour much higher, such that the air component is more present in the model. Also we made the outflow region a bit bigger. In this way, the outflow boundary lies further away from the end of the sluice and we hope therefore that the velocity jet has been damped more near the outflow boundary than in the previous model. In table 2 are the dimensions of the three blocks that represent the three parts of our model.

Part	Dimensions
Water basin	15m x 10m x 8m
Sluice	2m x 30m x 2,5m
Harbour	20m x 79m(top)/59m(bottom) x 10m

Table 2: Dimensions of the new model

The face of the harbour that intersects with the sluice is now a talud with slope 1:2 and the bottom of the harbour is at the same height of the bottom of the water basin. Therefore the sluice lies 0.5 meters above the ground at both sides.

5.2 Boundary conditions and mesh

The boundary conditions at the outflow boundary are the main issue in order to get a stable, correct solution. We therefore have to make adjustments in the boundary conditions.

At first we split the outlet boundary in a boundary for the air and a boundary condition for the water. The aim is to fix the water level at the outflow boundary with this change. Next, we had to figure out which boundary conditions suits best for these separate boundaries. We tried some boundary conditions and eventually we found a stable, good looking solution.

We had to decide which water height we want to have at the harbour side, such that the border between the two outlets will be at that level. We made the decision to do a simulation with the average water height of the harbour, which is 2.33 TAW [1]. This is equal to 5.83 meters above the ground. We rounded it up to 5.85 meters as our mesh coincides with this height. In chapter 8 we do another simulation where the initial water height is equal to the Lowest Astronomical Tide (LAT).

Because we made the geometry a lot bigger, having the same mesh size and mesh refinements will make our computations a lot heavier. Therefore we chose to refine the model at two sub-regions and have a courser mesh on the rest of the domain. These regions are a box over the whole width and length of the geometry, but a height from the ground to 0.5 meter above the top of the sluice and a box around the water level of the harbour, with a height of 0.3 meter. We again make use of boundary layers at the patches belonging to the barrier. We see the mesh in figures 5.1 and 5.2.

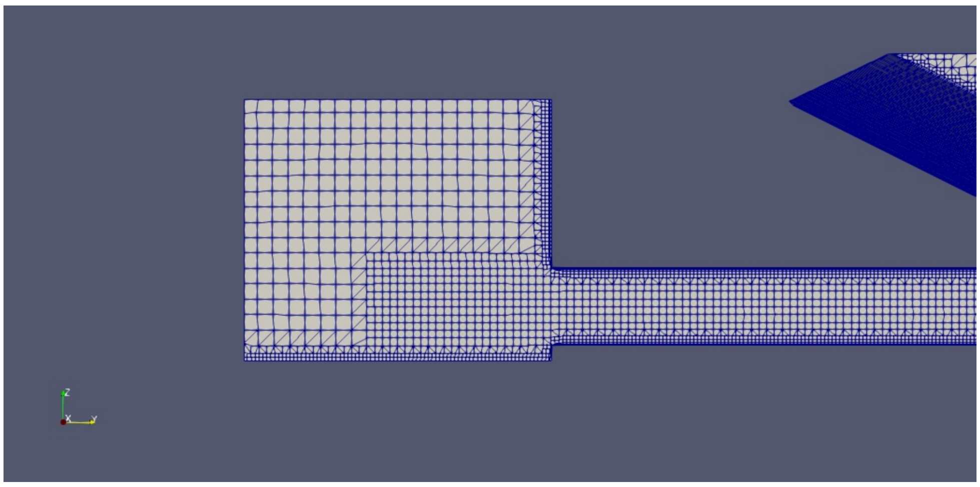


Figure 5.1: Slice of the mesh in the Y-Z plane at the inlet of the sluice

After the meshing we start choosing our boundary conditions, especially for our two outflow boundaries. Let us recall what the boundary conditions for the outflow boundary were in our initial model (table B.1). The first observation we made was the zero gradient boundary condition for the velocity resulted in inflow of air and outflow of water which made the water level drop. Now we have split the outlet, we set a zero gradient condition for the water boundary and a small fixed velocity for the air boundary directed out of the domain. Moreover, the phase fraction α can be fixed to the appropriate values, i.e. 0 for air and 1 for water.

Although it seems a good selection of boundary conditions, the simulation stops after around 17 seconds. Further inspecting why this is the case, we see a water level drop next to the outlet. The

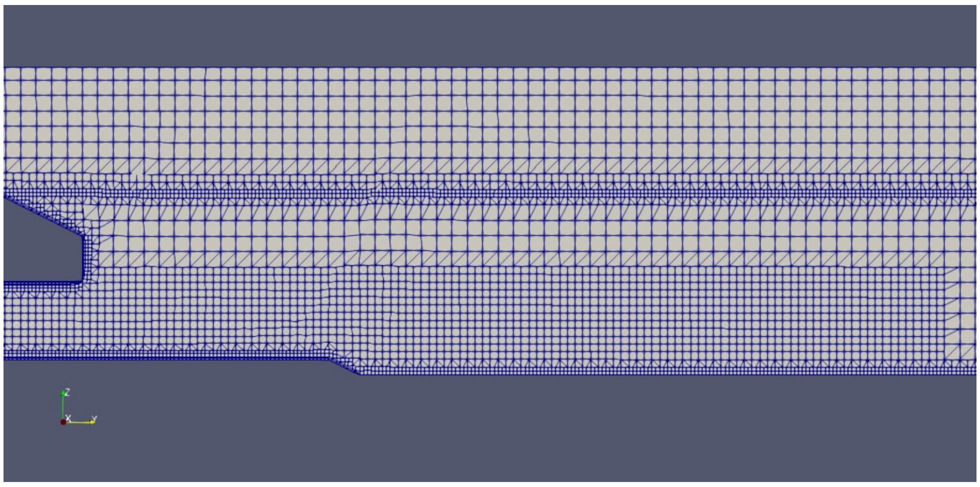


Figure 5.2: Slice of the mesh in the Y-Z plane at the outlet of the sluice

computations get disturbed because the water outlet prescribed only water and therefore lead to very small time steps.

To overcome this problem, we try a subtle difference in boundary conditions. Instead of just a fixedValue for α at the outlets, we try the OpenFOAM inletOutlet condition similar to the atmosphere patches. This condition makes it possible for the water to flow out, but makes sure that the backflow is always water. These conditions seem to work properly and don't lead to a simulation error. The boundary conditions at the outlet boundaries for our three main variables are schematically given in table 3 and a total overview of the boundary conditions are in appendix B.2.

Variable	Outlet air	Outlet water
U	fixedValue, uniform (0, 0.1, 0)	zeroGradient
p_{rgh}	fixedFluxPressure, uniform 0	fixedFluxPressure, uniform 0
α	inletOutlet, uniform 0	inletOutlet, uniform 1

Table 3: The final boundary condition choices for the outlet boundaries.

In the next section, we will give and discuss the results of the simulations with this geometry and these boundary conditions.

5.3 Results

The simulation had a simulation time of 180 seconds. After we ran interFoam, we got a few interesting results. In figure 5.3 we see the flow in the Y-direction at $t = 100s$. The red line is the location of the free surface ($\alpha = 0.5$). A nice first observation is that the water level is not dropping over time. We see that the water damps out pretty nicely and the water has a sharp contraction at the beginning of the sluice due to our boundary layers.

When looking somewhere further in time, at $t = 170s$, we see something strange happening. In figure 5.4, water flows back at the middle of the outflow boundary and pushes against the velocity jet coming from the sluice. Although we expect backflow due to the vortices of the jet (and setting the boundary condition for the velocity at zero gradient), they should appear at the sides of the outflow boundary. Looking at the streamlines of the flow in figure 5.5, we see that backflow pushes the velocity jet away to the sides and takes over the last part of our domain.

So we can conclude that the outflow boundary still gives us trouble. Because we want to keep the

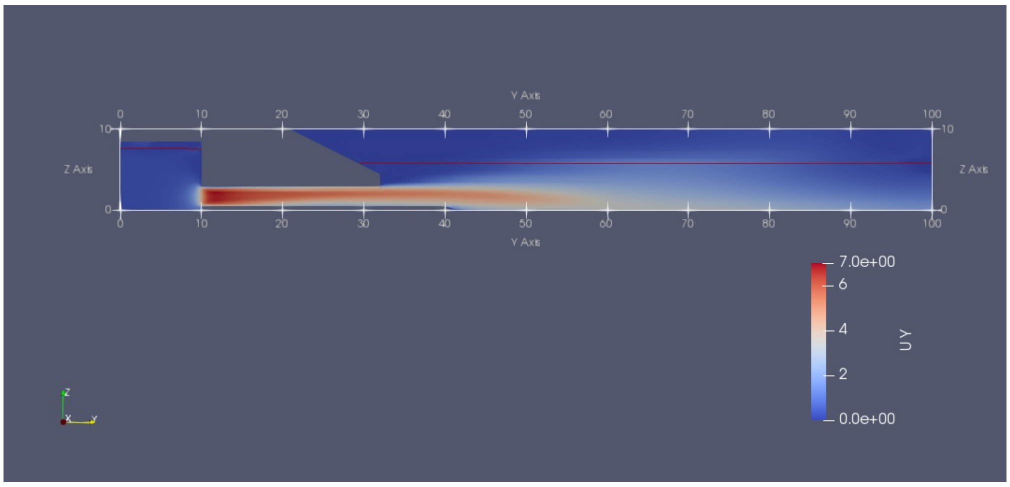


Figure 5.3: Visualisation of the water flow in Y-direction on the Y-Z plane at $t = 100s$

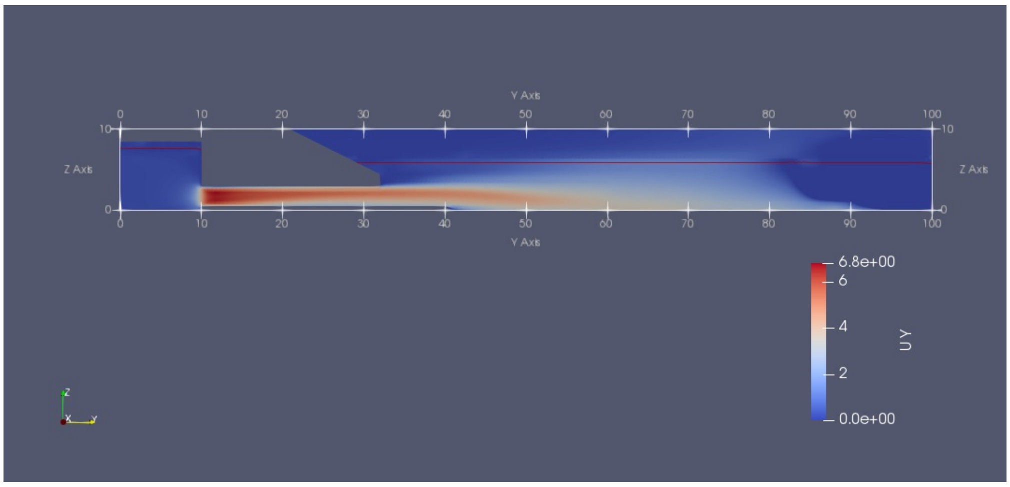


Figure 5.4: Visualisation of the water flow in Y-direction on the Y-Z plane at $t = 170s$

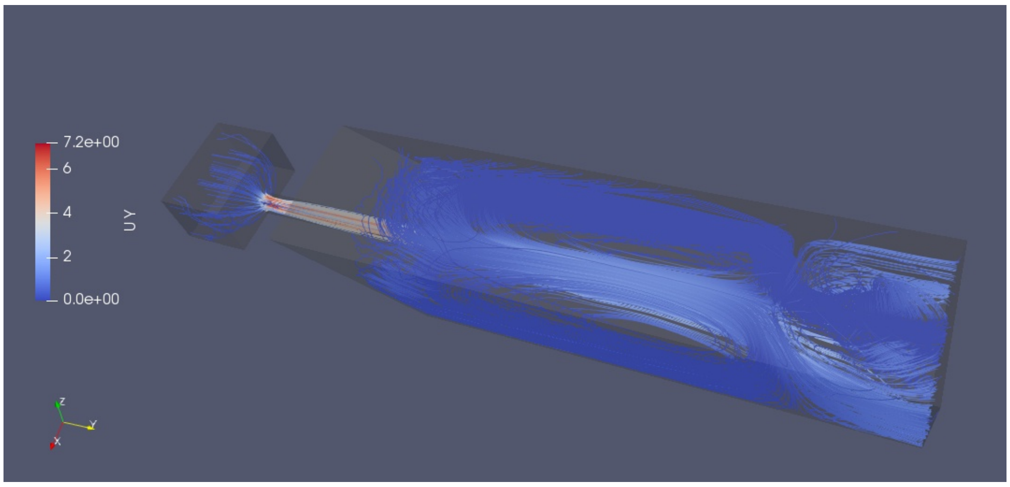


Figure 5.5: Visualisation of the streamlines at $t = 170s$

water level the same we have to introduce backflow. We have not found a way yet to control it though, as it leads to big disturbances at the end of our domain. However, we can still analyse this model when we negate the last meters of our model. The flow pattern before the disturbances looks pretty good actually, so we will take only data points from that part of the domain.

A solution to this behaviour can be to change the geometry and narrow the outflow boundary to a small patch. When we place for examples some pillars before the narrow outflow, the water gets damped hopefully enough to minimize the effects of backflow at the boundary. Another solution would be to force the water to flow out with minimal force with a fixed small pressure gradient at the outflow boundary. These solutions will be discussed in chapter 7.

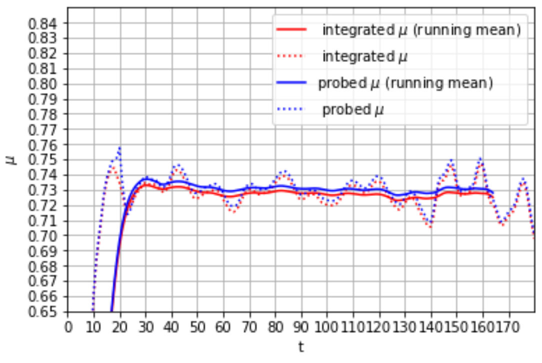
5.3.1 Loss coefficient

We are very interested in the loss coefficient over this model. Recall that the loss coefficient ξ is calculated with the relation $\xi = (1 - \frac{1}{\mu})^2$. To calculate the contraction coefficient μ we reformulate equation 1. We will use that $\Delta h = \frac{\Delta p}{\rho g}$ [5] and we obtain

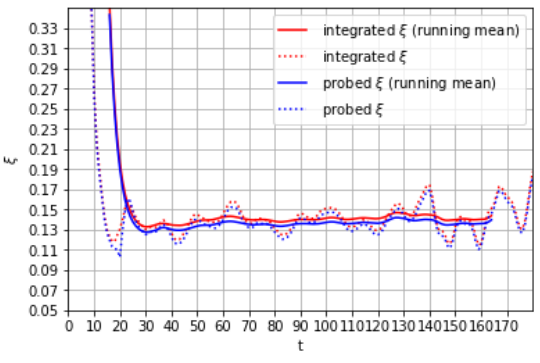
$$\mu = \frac{Q}{A\sqrt{2\Delta p/\rho}}. \tag{10}$$

For using this formula, we need to extract some data from our model, namely the discharge Q through the sluice and p_{rgh} at the two sides of the sluice. For the calculation of Q at each time step, we use the OpenFOAM post-processing utility `volFlowRateSurface`. This will cut a plane in the y-normal direction into the sluice at $y = 20m$ and calculates the discharge at every time step. For both pressures we use two methods. The first one is to use the `probes` utility from OpenFOAM, where we place one probe on each side of the sluice and it calculates the pressure at every time step. The other one is to let ParaView collect the integrated data at each time step over two vertical slices, one in front of the sluice and one after the sluice (y = 5m and y = 60m). To get the mean value of the pressure at both sides we divide the total integrated pressure by the integrated value for α.

All this data is then processed in a Python script, that eventually calculates the loss coefficient ξ. Additionally, we calculate the moving average of the contraction coefficient, which takes the 15 previous and the 15 upcoming values of the contraction coefficient and averages it. The python script can be found in appendix C. The graph of the moving average of the contraction coefficient μ and loss coefficient ξ over time is given in figure 5.6.



(a) Graph of μ over t of the separate outlet model



(b) Graph of ξ over t of the separate outlet model

Figure 5.6: Analysis of the contraction and loss coefficient over time for the separate outlet model

We are of course very curious to the results of a rigid lid model. That is what we are going to simulate next.

6 Rigid lid model

In order to determine the effects on the loss coefficient by modelling a free surface flow, we want to make a comparison with a rigid lid model. In this chapter we will give a brief overview of the different settings belonging to simulating a rigid lid model in OpenFOAM. We need to change from a multiphase flow to a single phase flow and we will therefore make use of a different solver. At the end we will present the results.

6.1 Differences with a free surface model

For the setup of this model we have to change quite a bit of files in the OpenFOAM directory. Because we can simulate a single phase incompressible flow now, we don't need the interFoam solver anymore. The best option now is to use the pimpleFoam solver, which just solves the regular Navier-Stokes equations for incompressible flows with the PIMPLE algorithm, which we have used with interFoam as well.

Another difference is that we can ignore the gravitational effects on the fluid. Therefore we will just work with pressure p instead of p_{rgh} and we can remove the file for the gravitational acceleration. The boundary conditions will not change much. We will put a slip condition on the top boundary and we can set the pressure equal to zero at the outflow boundary. Furthermore, we need to initialize an extra variable $\tilde{\nu}_{turb}$, which is the modified turbulent viscosity. Although we use a $k-\omega$ SST turbulent model and we actually don't need that variable in solving the equations, OpenFOAM gives an error message when that file is not present. An overview of all the boundary conditions for this model is given in appendix B.3.

At last we need to change the discretization and solving methods of the different terms in our equations, as they are a little bit different with the use of p instead of p_{rgh}.

All the other settings like the computation time, adjustable time step and the turbulent conditions are unchanged. The results are presented in the next section.

6.2 Results

In this section we will show the flow pattern of the model and the corresponding analysis on the loss coefficient.

At first we want to see if the flow pattern looks stable and whether we have some strange behaviour going on. One of the best ways to visualize this is again with streamlines. They are shown in figure 6.1. We notice that the pattern of the flow looks really nice. Despite the fact that the flow became a little less symmetric, the velocity jet moves through the middle of the domain and the backflow appears at the sides. No other strange backflow occurs.

Next we will calculate the loss coefficient over this model. This is shown in figure 6.2. We have calculated this coefficient just with the probed data, because we deal only with one fluid and after inspecting the results in ParaView we observed that the differences between the values of the points on a whole slice (as we did in the previous analysis) were negligible.

The first thing we notice is that the value of the loss coefficient stabilizes very fast. After approximately 60 seconds the amplitude of the oscillations in the values are almost gone. The rigid lid model therefore attains a steady solution pretty fast. Moreover, we see that the loss coefficient is a little bit less than the loss coefficient of the free surface model in the previous chapter.

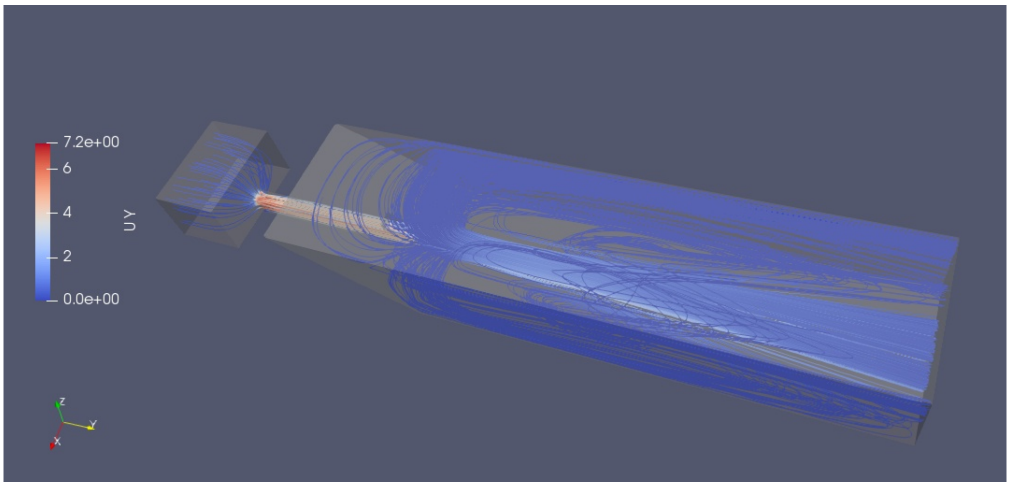
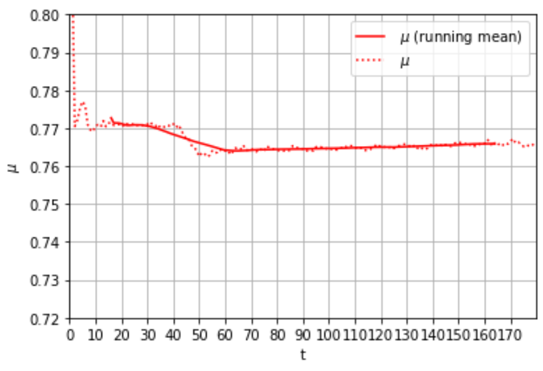
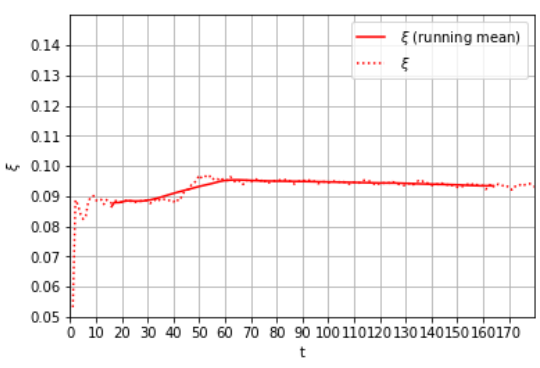


Figure 6.1: Streamlines of the rigid lid flow at $t = 170s$



(a) Graph of μ over t of the rigid lid simulation



(b) Graph of ξ over t of the rigid lid simulation

Figure 6.2: Analysis of the contraction and loss coefficient over time for the rigid lid model

Because we were not completely satisfied with the previous free surface flow, we will move on to finding ways to get a stable solution around the outflow boundary.

7 Models to stabilize backflow

We saw in chapter 5 that we had some strange behaviour of the flow near the outflow boundary, where backflow takes over that part of the domain and forces the velocity jet to move out of the way. In this section we propose two ways to overcome this problem.

The first way is a change in geometry, where we narrow the end of the domain such that the outflow boundary is a much smaller patch. In addition to that, we place some pillars in front of that boundary to damp the velocity jet even more. This will lead to less backflow instabilities, as water won't get out of the domain very easy.

The second way is to impose a small fixed pressure gradient condition at the outflow boundary. This forces the water to flow out of the domain, such that we don't have the backflow issues. The gradient needs to be sufficiently small, otherwise it can take over the whole flow pattern as it will suck the water out of the domain.

7.1 Narrow outflow boundary

For this model, we need to make some changes in the geometry with SALOME. We 'cut out' two boxes of 7m x 10m x 10m at both sides of the end of the harbour. Furthermore, we place five pillars with a width of 0.6 meters at the opening of the narrow part of the harbour. The geometry can be seen in figure 7.1. To visualize the flow pattern we use again the streamlines function in ParaView.

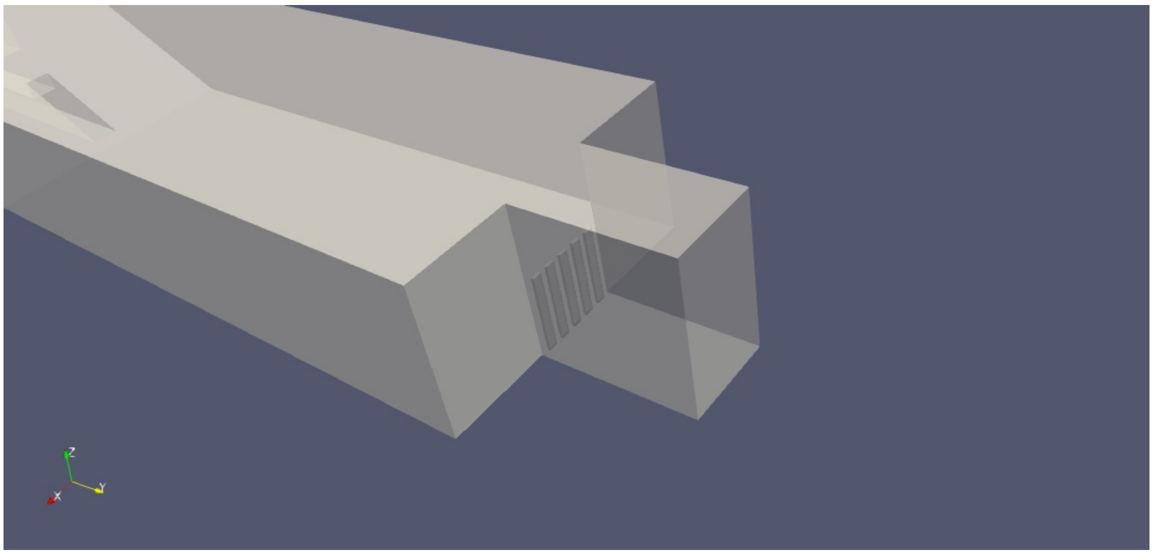


Figure 7.1: Geometry of the narrow outflow part with pillars

This is showed in figure 7.2. The flow pattern looks much better than the free surface flow we had earlier. What immediately catches the eyes is that the flow doesn't get past the pillars at all. Although you can't really see this in the streamlines picture, the water level starts to rise and the discharge rate through the sluice decreases.

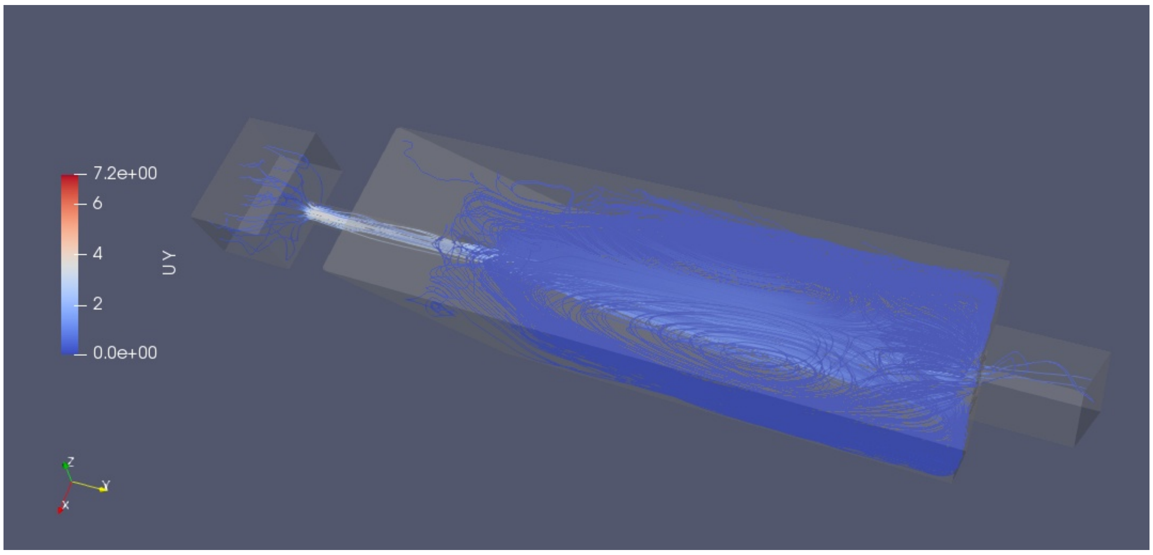
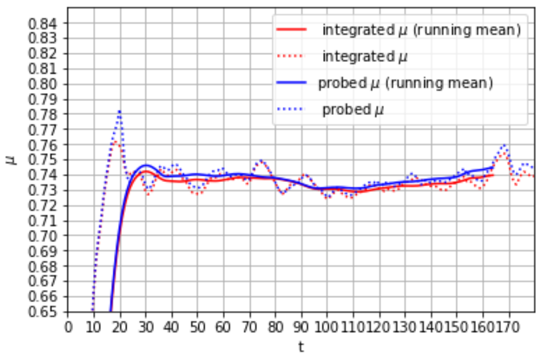
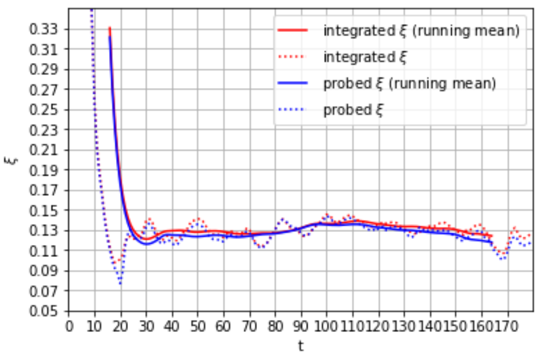


Figure 7.2: Streamlines of the model with a narrow outflow boundary

This water level rise will definitely not mean that we can't do a good analysis on the loss coefficient. On the contrary, we see in figure 7.3 that the value of the loss coefficient over time gets very stable at an early stage of the simulation.



(a) Graph of μ over t of the model with a narrow outflow boundary



(b) Graph of ξ over t of the model with a narrow outflow boundary

Figure 7.3: Analysis of the contraction and loss coefficient over time for the model with a narrow outflow boundary

7.2 Fixed pressure gradient

Another possible solution to the backflow problem is to prescribe a pressure gradient on the outflow boundary. The only change we therefore made here is the boundary condition for p_{rgh} at the water outlet boundary. Thus

```
    outlet_water
    {
        type                    fixedFluxPressure;
        value                   uniform 0;
    }
```

becomes

```
    outlet_water
    {
        type                    fixedGradient;
        gradient                uniform 5;
    }
```

We also tried gradient values of 0.2 and 1, but some discontinuities occurred at $t = 140s$ and $t = 160s$ respectively. These discontinuities lead to a blow up of the simulations, but we haven't found the exact reason for the discontinuities.

In figure 7.4 we can see the streamlines of this model. It looks like this method also achieves a nice flow pattern. We see the circulations at the side and we have the main jet flowing through the middle towards the outflow boundary.

Now we will look at the effects on the loss coefficient with this boundary condition. The graphs are seen in figure 7.5. The first observation we make is that it takes a lot of time to get a stable value for the loss coefficient. Even at the end of the simulation the values do vary quite much. Also at the beginning of the simulation, we see that the contraction coefficient attains values that are higher than our previous simulations. This is something that we can explain. At the first minute the flow gets pulled towards the outflow boundary which leads to less contraction. When this 'pulling effect' is taken over by the regular stream we see that the value become less. This also explains why it takes longer for the loss coefficient to converge.

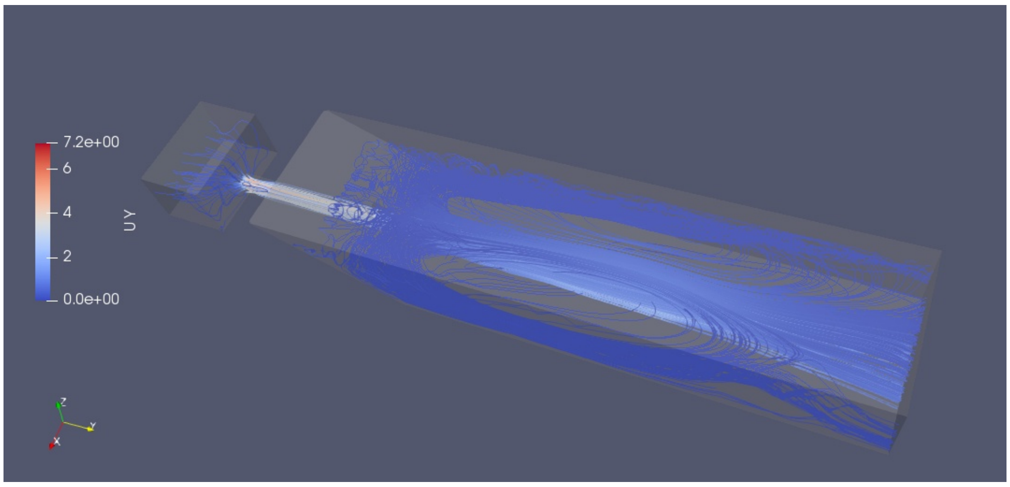
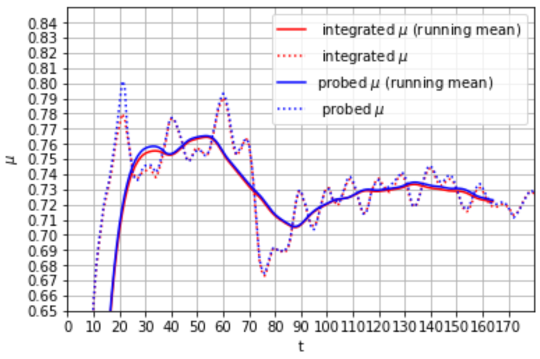
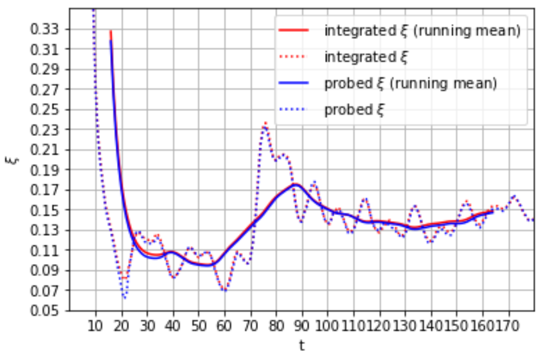


Figure 7.4: Streamlines of the flow at $t = 170s$ with a fixed pressure gradient at the outflow



(a) Graph of μ over t of the fixed pressure gradient simulation



(b) Graph of ξ over t of the fixed pressure gradient simulation

Figure 7.5: Analysis of the contraction and loss coefficient over time for the fixed pressure gradient model

8 Model of the Lowest Astronomical Tide

Now we have successfully stabilized the flow around the outflow boundary, we use the pressure gradient method to model the flow during LAT. The water height at the LAT is 0.05 TAW, whereas the average tide is at 2.33 TAW. Therefore we need to change the initial field of α, where

```
    boxToCell
    {
        box (0 10 0) (20 100 5.85);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
```

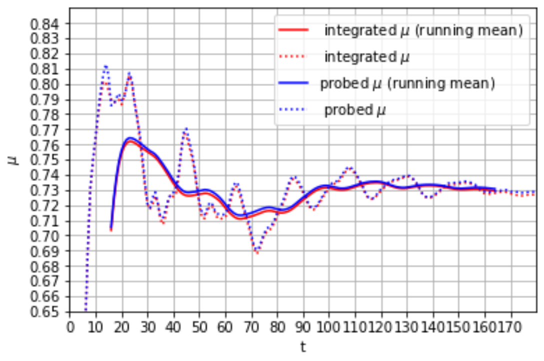
becomes

```
    boxToCell
    {
        box (0 10 0) (20 100 3.55);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
```

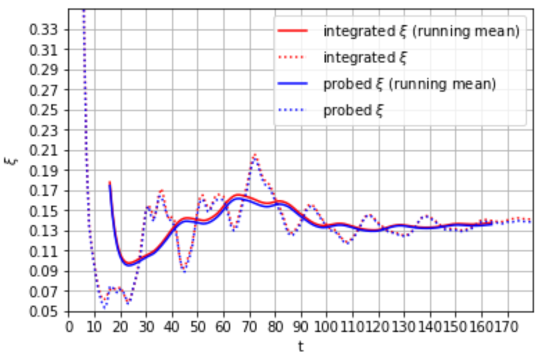
in the setFields file in OpenFOAM.

We also had a refinement region around the interface, so we need to rebuild the mesh with a refinement region around the height of 3.55 meter. This is done with a simple adjustment in the snappyHexMesh dictionary file.

We use again the pressure gradient condition at the outflow boundary of the water. We have seen that it works and we like this option over the narrow outflow solution, because the water level will not rise in this case. Although it will take longer for the loss coefficient to stabilize, we think that this is a more reliable solution for the flow as the discharge rate through the sluice will stay closer to the wanted discharge rate. In figure 8.1 we see the graphs for the contraction and loss coefficient for the LAT model.



(a) Graph of μ over t of the model with LAT



(b) Graph of ξ over t of the model with LAT

Figure 8.1: Analysis of the contraction and loss coefficient over time for the model with LAT

We thought that maybe the air could lead to a more difficult analysis on the loss coefficient, as the height of the sluice is almost as high as the water tide and air could maybe come into the sluice. But from the results we see that the values of the loss coefficient are very stable after around 100 seconds. Before that, the flow needs again some time to stabilize the effects from the pressure gradient condition.

9 Model with optimized inflow conditions

To see what the effects of a different design are, we change the inflow design of the sluice. Instead of the sharp edges at the connection of the water basin to the sluice, we place a fillet with a radius of 2 meters at the top and side edges of the sluice. This should lead to less contraction as water will flow into the sluice more fluent. The construction can be seen in figure 9.1.

We still make use of the boundary condition of the pressure gradient at the water outflow boundary. The results with respect to the loss coefficient are given in figure 9.2.

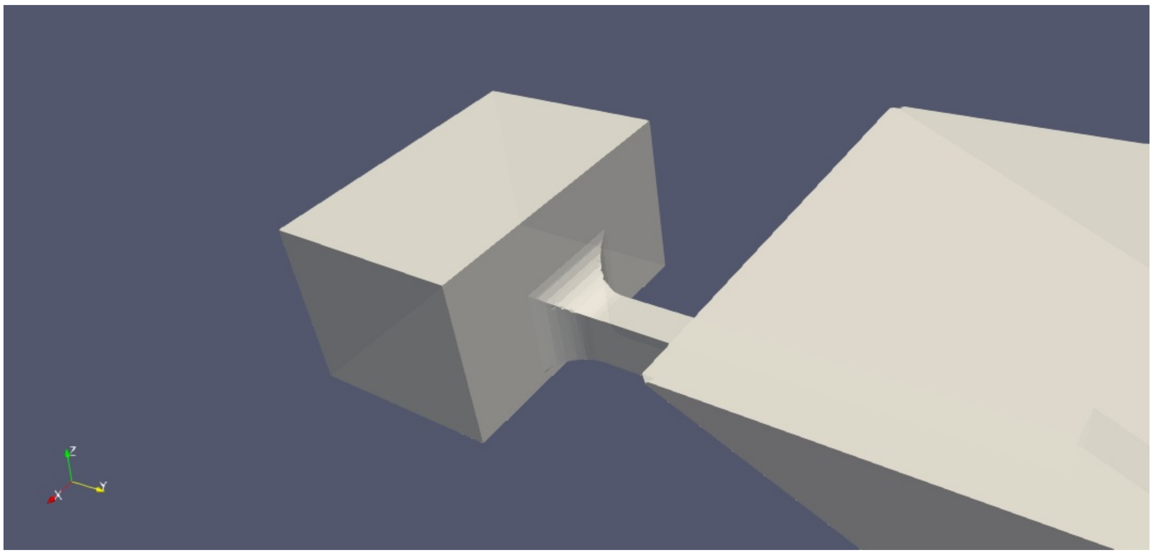
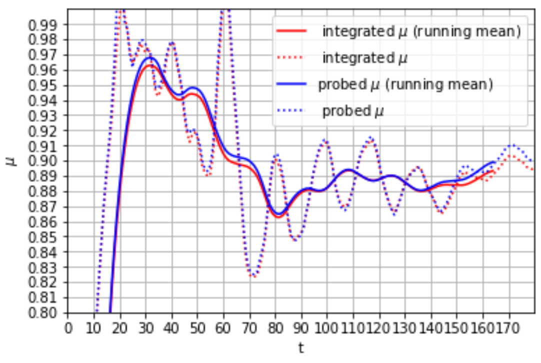
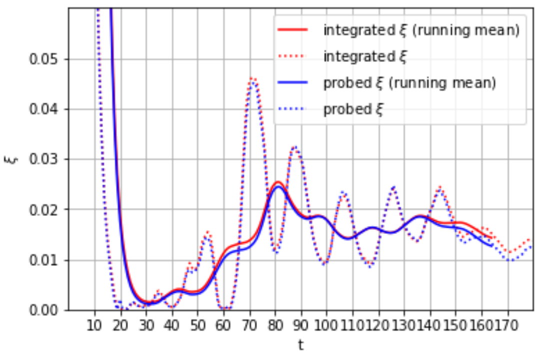


Figure 9.1: Geometry with a fillet at the inflow part of the sluice



(a) Graph of μ over t of the model with round inflow edges



(b) Graph of ξ over t of the model with round inflow edges

Figure 9.2: Analysis of the contraction and loss coefficient over time for the model with round inflow edges

The loss coefficient over this model is a lot less than the previous models, as we have expected. The inflow construction on the sluice leads to almost no contraction. So this confirms that the construction of the sluice really matters in the loss coefficient. We see again the high peak at the beginning of the simulation due to the pressure gradient condition at the outflow boundary. At the second half of the simulation, the values of the loss coefficient vary much less.

10 Discussion and conclusion

We will now discuss all the obtained results and discuss whether modelling a free surface flow is needed to derive the loss coefficient over the design of the sluice. We also discuss the capabilities and performances of OpenFOAM, especially in free surface flows.

We will first start with the differences between the values of the loss coefficient over all the different models we have analysed. For each model we looked at all the values of the loss coefficient after the simulation was halfway, which was at $t = 90s$. This is mainly because for some models it took more time to get to more stable values of the loss coefficient, especially the models where we prescribed a pressure gradient on the outflow boundary. We then looked at the running mean of ξ to determine the upper and lower bound of this graph. These range of values of ξ of all the models is given in table 4.

Model	Chapter	Values of ξ
Separate outlets	Chapter 5	$0.13 < \xi < 0.15$
Rigid lid	Chapter 6	$0.09 < \xi < 0.10$
Narrow outflow boundary	Chapter 7.1	$0.11 < \xi < 0.15$
Fixed pressure gradient	Chapter 7.2	$0.13 < \xi < 0.18$
Lowest Astronomical Tide	Chapter 8	$0.13 < \xi < 0.15$
Optimized inflow conditions	Chapter 9	$0.01 < \xi < 0.02$

Table 4: The range of values of ξ of each model

When we ignore the results of the last model with the better design of the sluice, we see that the values of ξ are all pretty close to each other. The rigid lid model got to a value that is a bit less than all the others and has also a sharper estimation on the loss coefficient. Although it seems like it is a small difference, the differences are still significant in our opinion.

The sharper value of the loss coefficient can be explained by the absence of waves. The waves will lead to a varying height of water over time and the oscillations are also seen in the graphs of all the other models. We can also state that in this particular geometry, the rigid lid assumption gives us a lower bound estimate for the loss. We think that the air and waves have some impact at the pressure difference between the two sides of the sluice. Of course we don't know if that still holds when we make changes to the geometry. The fact that the geometry is pretty general for this type of water construction makes us suspect that the models that want to estimate the loss over such a design, are pretty similar to this one.

We also got the loss coefficient estimation of the COMSOL model Witteveen+Bos made earlier. The loss coefficient there is around $\xi = 0.07$, much lower than our results. This significant difference is maybe due to the different geometry we have applied. The geometry of the COMSOL model was the same as we used in our first simulation. After that, we changed the geometry to a more similar situation of the harbour in Ostend. Also a different turbulence model was used and there are maybe a few differences in the discretization, in which we didn't look further into.

The construction of the sluice really makes a difference in the amount of loss over that design, as is seen in the last model. With the round edges there is almost no loss. The water can flow easily into the sluice without almost any contraction.

We can conclude that a rigid lid model does a good job in a first approximation of the loss coefficient. In this case, we don't have any experimental data to calculate the real value of the loss coefficient and compare it with our simulations, but there are significant differences between the rigid lid and free surface models. From literature we have found that 90 degree edges lead to a contraction coefficient $\mu = 0.6$ or $\mu = 0.7$ [5]. These contraction coefficients correspond to a loss coefficient of $\xi = 0.44$ and $\xi = 0.18$ respectively. So if you really want a fast estimation on the loss coefficient, one could opt for a rigid lid model to get a lower bound. If the goal is also to make a realistic model, the more complex free surface flow is the best choice. It does take more time to get to more stable values of the loss coefficient, but you can't negate the influence of the air and the free surface in models like these.

We chose the software program OpenFOAM for the simulation of our CFD models. We have seen that it has some very good options to simulate free surface flows. The advantage of open source is that you can really dig deeper into the code to see what is happening and make changes. Also there are a lot of CFD communities online that improve OpenFOAM codes and once or twice per year, a new version is released with more different cases and options.

Although the internship lasted for 10 weeks only, we could really see the many possibilities we have with OpenFOAM. These possibilities also makes it sometimes hard to find the right settings and conditions right away. The best example for that was the choices of boundary conditions we had, especially when trying to find the right condition on the outflow boundary which is always a challenge.

Apart from that, we had a really positive experience with OpenFOAM. We think that when we find the right people that share a common goal in modelling free surface flows, further improvements can be made to optimize these kind of simulations. It was too bad that we didn't have any experimental data to compare the OpenFOAM results with. I think that these are the challenges Witteveen+Bos may want to invest on to improve this study about the performance of OpenFOAM regarding free surface flows.

11 Epilogue

I really enjoyed my time at Witteveen+Bos. This internship gave me a lot of opportunities that I am really thankful of. I got the chance to get to know the very useful CFD software OpenFOAM, which will be beneficial for my career in the future. Also the fact that OpenFOAM was not really used before inside Witteveen+Bos gave me the feeling that I did contribute to the company.

I want to thank Witteveen+Bos for a great experience in the working field. All the employees were really helpful and the work environment was pleasant. I want to also thank two people form Witteveen+Bos in particular.

At first, I want to thank Wim Ridderinkhof for the internship project. The meetings with him were really useful and I hope he can use my results in the project. His critical thinking helped me to improve the model I made in OpenFOAM. Secondly, I want to thank my supervisor Arie de Niet. He really guided me inside Witteveen+Bos and helped getting to know the company well. The weekly meetings with him gave me a lot of adequate feedback, which was very pleasant. Also the opportunity to follow an OpenFOAM course in Delft together was nice to have.

At last, I want to thank my supervisors from the RuG, Roel Verstappen and Roel Luppes, for assessing my internship. My meetings with the first supervisor Roel Verstappen gave me new insights on how to tackle some problems I encountered during the internship.

References

[1] Witteveen+Bos. *Alternatieven Rapportage 'Overstromingsmaatregelen achterhaven Oostende - Ontwerp'*, 17 juli 2019. kenmerk 105618/mome/045.

[2] Florian R. Menter. *Improved two-equation k-omega turbulence models for aerodynamic flows.* NASA, 1992.

[3] Tobias Holzmann. *Mathematics, Numerics, Derivations and OpenFOAM®*. 2019. doi: 10.13140/RG.2.2.27193.36960. URL `https://Holzmann-cfd.de`.

[4] *OpenFOAM - Spillway Tutorial.* URL `https://www.hpc.ntnu.no/display/hpc/OpenFOAM+-+Spillway+Tutorial#OpenFOAM-SpillwayTutorial-system`.

[5] J.A. Battjes. *Vloeistofmechanica b70.* TU Delft, 1990.

[6] A.E.P. Veldman. *Computational Fluid Dynamics.* Rijksuniversity Groningen, 2012.

A CFD modelling of free surface flows with OpenFOAM

OpenFOAM is a free open source CFD software and is an abbreviation for Open Source Field Operation and Manipulation. It is largely used in the fields of engineering and science, for both commercial as academic purposes. This documentation is made for Witteveen+Bos regarding the computation of CFD models, with free surface flows in particular.

A.1 Introduction

We start this document with giving an overview of how to create an OpenFOAM simulation. All the steps you need to follow are mentioned here. Then we will show you how a typical OpenFOAM project directory looks like. Every important file in these directories are explained briefly. Next, we show your options in creating geometries for your model and how to construct a suitable mesh on it. These options range from built-in OpenFOAM utilities to using another open source software like SALOME.

Then we will dive deeper in the settings you need to run a free surface model. We will talk about the interFoam solver that OpenFOAM uses to compute a free surface flow. Furthermore we present a list with the most important boundary conditions used for this flow.

At last we will explain all the post-processing utilities we can make use of. For the visualization, we use the software ParaView which can be directly called from OpenFOAM. The software OpenFOAM itself has also a big list of post-processing tools to gather useful data that can be analysed.

A.2 Setup of an OpenFOAM model

The OpenFOAM installation contains more than 200 tutorials with all kind of flow models. From (in)compressible flows to heat transfers, many example codes are available in the OpenFOAM package. People using OpenFOAM often will pick one of the example flows that has many similarities with the CFD model they want to simulate and adjust the settings in the files to their problem.

At first, the geometry of the model needs to be made. OpenFOAM is able to build relative simple geometries with not much difficulty. With that geometry, OpenFOAM uses a builtin routine to create the mesh. For more complex geometries people make use of a Computer-Aided Design (CAD) system. With such a CAD system we can build a geometry that matches the real life situation. From there we can create a mesh and export the file to the folder with your OpenFOAM files. The CAD system we used is SALOME. SALOME is also an open source program which is widely used to create geometries and meshes for numerical models.

With the geometry and mesh created, all the mathematical and solving properties can be set according to your model. First we need to think about which properties we want to impose for our flow and which kind of tutorial suits best. At every type of boundary we need to impose a condition for the different variables like the pressure and velocity. Also an initial field can be set. If we have for example a multiphase flow, we can set which part of the geometry initially consists of one type of fluid and which part consists of the other fluid. Then we need to tell the software which discretization schemes we want to use for the different terms in our equations (depending on the flow we want to simulate) and how we want to solve these equations.

At last we need to set the parameters for our computations. Which timestep do we want for example and do we allow it to vary during the computation. For large computations we are also able to compute the CFD calculations parallel.

When everything went well, we are able to look at the simulation with Paraview. Paraview comes with the OpenFOAM package and offers useful utilities in the post-processing part of the model.

A.3 OpenFOAM organization

An OpenFOAM project consists normally of three folders in your project directory. In those folders are the appropriate C++ files that describe your flow and simulation. The first folder is called `0`, which consists of all the boundary conditions of your flow. Every variable in your equations needs to be prescribed by some boundary condition at each boundary patch. The second folder is called `system`. This folder consists of all the dictionary files that for example set up your geometry, the initial field and the parameters for the computations. Furthermore, the settings of the discretization of the equations and how to solve these are present in this folder. At last, we have the folder `constant`, which is the home of some constant properties of the flow, like the turbulence properties and constant variables like the gravitational constant. Moreover, all the data of the geometry and the mesh is stored here in the sub-folder `polyMesh`. Figure A.1 gives an overview of the directory structure of an OpenFOAM project. In the next paragraphs, we will give a short overview of the most important files in each folder.

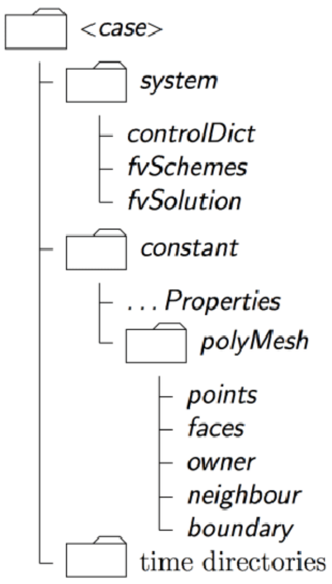


Figure A.1: OpenFOAM project directory structure

A.3.1 System

The `system` folder is the base of each OpenFOAM project. With the dictionaries for the geometry, meshing and solvers in it, we can create any flow we want. Here is an overview of the most important files:

blockMeshDict
The first step in a numerical model is making the geometry and a good looking mesh. The `blockMeshDict` file is responsible for the design of the blocks and shapes present in your model and generates a simple mesh on it. The outline of the file is as follows: First you generate the vertices of your shapes. After that you tell which vertices are part of which shape and you can impose a mesh on those shapes. Then you need to tell which patches of the shapes belong to which boundary. This is quite important, as you will need those boundaries later on when you impose all the boundary conditions.

The `blockMeshDict` file is mostly for simple shapes, e.g. block structures or spheres. For more complex geometries, there are other tools and software programs to create these shapes and import it to the OpenFOAM environment. Meshing can then be done with the `snappyHexMesh` tool of

OpenFOAM. It is important to note that you still need this `blockMeshDict` file to create the mesh. We will discuss this later on.

controlDict
The `controlDict` file consists of the parameters of your computation. Here you will tell the software how long your model needs to run, at which time steps it needs to write all the data and whether it has a fixed or varying time step. Especially the time step is really important for getting a stable solution. That is why the variable time step is set to true most of the time. Then you need to give up also the maximum Courant number, which is an important condition between the mesh size and the time step in order to get a stable solution. OpenFOAM will adjust the time step according to the given maximum Courant number.

fvSchemes
In the `fvSchemes` you will tell OpenFOAM how it needs to discretize all the terms in your equations. So you give up the time integration scheme, the discretization of the divergence and gradient parts of the equation and how the interpolation needs to be done.

fvSolution
How to solve for all the variables in the discretized model is written down in the `fvSolution` file. You will tell which algorithm you want to use in order to solve the variant of the Navier-Stokes equations you use for your model. Then every variable is solved at each time step by its own solver, with a certain tolerance, possible smoothing processes and corrector algorithms.

Other files
The above files are all necessary in an OpenFOAM project in order to run your model. However, OpenFOAM has more functionalities in order to get a better or faster simulation. To run your simulation in parallel, you need to include a `decomposeParDict` file, which will divide the domain equally over the available CPU cores. In the case of a multiphase flow, such as a free surface flow, you can set the initial field in a `setFieldsDict` file to specify which part of the domain consists of which fluid. Also, if you want to impose an initial velocity on a certain part of the domain, you can use this file to do so.

For importing a complex geometry and meshing it with the `snappyHexMesh` tool of OpenFOAM, you need the files `surfaceFeatureExtractDict` and `snappyHexMeshDict` files. These files will be explained in paragraph A.4.2.

A.3.2 0

The `0` directory is actually our initial time directory which contains all the initial and boundary conditions for the different variables of the equations. During the simulation all the other time directories will be made with the variables and their respective values. Below is an example of the boundary condition for the velocity vector field U.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Website:  https://openfoam.org                  |
|   \\  /    A nd           | Version:  7                                     |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    0;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            variableHeightFlowRateInletVelocity;
        flowRate                40;
        alpha                   alpha.water;
        value                   uniform (0 0 0);
    }
    outlet_air
    {
        type                    fixedValue;
        value                   uniform (0 0.1 0);
    }
    outlet_water
    {
        type                    inletOutlet;
        inletValue              uniform (0 0 0);
        value                   $internalField;
    }
    concretewall
    {
        type            noSlip;
    }
    talud
    {
        type            noSlip;
    }
    wall
    {
        type            slip;
    }
    atmosphere
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

// ************************************************************************* //
```

The file is constructed in a particular way. At the top of the file we see the header, which is present in all the OpenFOAM C++ files with the version of OpenFOAM we use. In the `FoamFile` section we see the type of variable and where it is located.

An important part is that you put the right dimensions belonging to the variable. The order of the dimensions from the SI unit system are `[kg m s K mol A cd]`. So make sure that the right dimension is given to the field.

Next you are asked to give up an internal field. So for example an initial velocity or pressure can be defined here. If you don't want a uniform initial field for the whole domain, you can use the `setFields` utility, which will be discussed later. This utility can create initial values on sub-domains of the model. So if you want to start with a geometry which is already partly filled with water, you can use `setFields` to do that. This will then automatically change the value of the internal field to a non-uniform list in the corresponding file in the `0` folder.

At last the boundary conditions need to be set for your variable in the `boundaryField` part. At every boundary patch you can choose the specific type of boundary condition and which value it has to have. Some boundary types need more input values. A list with the most common boundary conditions is given in A.5.2.

A.3.3 Constant

The `constant` folder is the home of the standard properties of the flow. Also files with some constant fields like the gravitational acceleration g are stored here. Furthermore, during the meshing part a folder `polyMesh` will be created which contains the data of all the cells in the domain.

transportProperties
The properties of the fluids are set in the `transportProperties` file. Properties as the type of fluid (Newtonian etc.), its density and viscosity are found here. In the case of a multiphase flow, you need to set these properties for both fluids and also determine which fluid you want to correspond to $\alpha = 1$ and which fluid corresponds to $\alpha = 0$. Also the surface tension σ of the fluids is given here, which is almost always set to 0.07 N/m.

turbulenceProperties
In `turbulenceProperties` we choose the type of turbulence simulation we want to do and which model we should implement. For the simulation type, one can choose between `laminar` for no turbulence, `RAS` for the Reynolds-Averaged Simulation modelling and `LES` for the Large-Eddy Simulation modelling. Depending on this choice, you can choose an appropriate turbulence model. For the `RAS` simulation, a k-ϵ model or k-ω SST model can be used for example.

polyMesh
The folder `polyMesh` is made after the meshing and contains the data of all the cells and their corresponding properties. The most important file is the `boundary` file which contains all the faces that belong to the boundary. Make sure that these boundaries have the same name as the boundaries you used in defining the boundary conditions in all your files in the `0` directory. Also the type of the boundary can be important, i.e. if you want to use a boundary condition belonging to a wall, you need to define that boundary also as a wall.

A.4 Geometry and meshing

A mesh of a simple geometry, i.e. a box, can be made pretty easily with the OpenFOAM utility blockMesh. In the system folder of your project you will have to make the file blockMeshDict. Again, the best option is to copy one of the blockMeshDict files of the existing tutorials and adapt it. In this file you will assign all the vertices of the blocks in your model to a location. The order in which you put these vertices is very important! With these vertices you can assign the blocks that present in your geometry. In the case of a single box, you will clearly need only 8 vertices.

Now you have assigned the blocks, it is time to tell the program which faces of the blocks belong to what kind of boundary. Again, the order of the vertices that define a patch is crucial. You can name every type of boundary the way you want, but it is advised to keep the names straightforward, such as inlet, outlet, wall etc. Every boundary has also to be provided with a certain type. The most common are 'patch' for the inlet, outlet and atmosphere and 'wall' for the wall boundaries. As you have assigned the right patches to the right type of boundary OpenFOAM can create the mesh for you. This is easily done by typing

```
blockMesh
```

in your terminal.

A.4.1 SALOME

For more involved geometries, there are several CAD systems that can be used. SALOME is one of them and is able to create the geometry and build a mesh. In this section we will outline the most important functions creating a geometry and mesh. At first we need to build our geometry. The best way to start is to insert the vertices of your blocks. By simply changing the coordinates of each vertex and applying them we create our corners of each block. After that we can create a box from the insert panel and just by clicking 2 opposite vertices (opposite in each direction), a box is created. Defining all the vertices has the advantage over just creating a box and drag it to the right place as we are able to adapt our geometry much easier if we want so.

After you have created all separate shapes, we can merge them with the fuse function. By doing so, all the parts are connected and all the overlapping faces are removed. Now we're able to let a fluid flow through the whole geometry.

The next step is to 'explode' the total geometry. Then we see all the different faces of the geometry and we have the option to group and export them. You want to group them per boundary (wall, atmosphere etc.) to prescribe the same boundary conditions on such a group.

Now there are two options to create the mesh from this geometry. One option is to use SALOME for this as well. We don't discuss this part, as we only experimented with this option briefly.

The other option is to export all the different boundary groups + the whole geometry in STL-format to your OpenFOAM directory and use the `snappyHexMesh` tool. This procedure will be discussed in the next session.

A.4.2 SnappyHexMesh

The mesh is one of the most important parts in the process of getting a stable simulation. With the `snappyHexMesh` utility, OpenFOAM is able to make a suitable uniform grid with necessary refinement regions and boundary layers. Furthermore it is able to check the quality of the generated mesh. To perform meshing with the snappyHexMesh tool, we need to follow a sequence of steps which we will discuss here.

To begin, we need to make a box with the blockMesh tool that contains the model we made in SALOME. Define the mesh size inside this box. This mesh size will be the maximum mesh size in our mesh that we are going to generate. After you are done, type

```
blockMesh
```

in your terminal. You then see that it created your `constant/polyMesh` folder.

The next step is to tell OpenFOAM which surfaces of the model it needs to snap to and are available for refinement. In order to do that, it is important that you place all your STL-files into the constant/triSurface folder in your OpenFOAM environment. Then we need to use a file called surfaceFeatureExtractDict. It is again easiest to use an already existing file from one of the available tutorials and adjust it accordingly. In this file you need to specify all your boundary STL files (So not your whole geometry STL file). When you are done, you perform

```
surfaceFeatureExtract
```

and you will see that it created eMesh files of your boundary STL files in the triSurface folder. Also an additional extendedFeatureEdgeMesh folder is created with all the data of the edges belonging to the boundary faces in object format.

Then we arrive at the part we are most interested in: the actual meshing part. At first, copy an already existing snappyHexMeshDict file from one of the tutorials and place it in your system folder. The meshing in this dictionary file consists roughly speaking of three parts: castellatedMesh, snap and addLayers. Before we start, we need to define our geometry and surfaces to make them available for refinement.

The first step is then the castellatedMesh part. In this part, you can refine all your regions and surfaces. So if you want to have a finer mesh at the walls, you can add some refinement levels to those surfaces. Also if you want to create a refinement region because you expect that area to be highly submissive to changes, you can add such a region to the geometry at the top of the file. Then you refine that region in the refinementRegions section.

At last you need to tell OpenFOAM which part of the geometry it needs to keep. So if you wan to analyse a flow inside your complex geometry, you enter a point in the domain of that geometry. It could of course also be the case that you want to analyse the flow around some body, such as air around a vehicle. Then you need to select a point outside the geometry, but inside the box created with blockMesh.

For the snap section you usually don't need to change much regarding to other snappyHexMeshDict files of other problems. On the other hand, addLayersControls has some important features to look after. If you want boundary layers at some walls, you can add those surfaces to the layers parameter. Then you can enter the amount of boundary layers and their expansion ratio.

When you're done, you enter

```
snappyHexMesh -overwrite
```

or, in the case you want to run it in parallel,

```
mpirun -np <cores> snappyHexMesh -overwrite -parallel
```

and OpenFOAM will start meshing. In order to do parallel computations, you need an extra file decomposeParDict in your system folder which consists of parameters how you will subdivide your domain over the CPU cores. The commands to decompose and reassemble your domain are

```
decomposePar
```

for decomposing the domain before you run snappyHexMesh and

```
reconstructParMesh -constant
```

for reconstructing the mesh afterwards.

At the end of the file is a list with a lot of parameters to check the quality of the mesh that just have been created. When the output tells you that some conditions have not been met, you know you need to refine the mesh more at some complex surfaces.

A.5 Mathematical settings for free surface flows

A.5.1 Solver

OpenFOAM has a different solver for almost every different type of flow. Free surface flows belong to the group multiphase flows. When considering incompressible, isothermal fluids, the most convenient choice of solver is the interFoam solver. This solver will solve the Navier-Stokes equations, together with an additional equation for capturing the position of the interface. The Navier-Stokes equations it will solve are

$$\nabla \cdot \boldsymbol{u} = 0$$
$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\,\boldsymbol{u}) = -\nabla p' + \nabla \cdot \tau + F,$$

where

$$p' = p - \rho g h.$$

The interface is captured by the VOF method interFoam uses. It solves

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{u}) = 0,$$

where α is the phase of the flow defined by

$$\alpha = \begin{cases} 0, & \text{if particle is in fluid 1} \\ 0 < \alpha < 1, & \text{at the interface} \\ 1, & \text{if particle is in fluid 0.} \end{cases} \tag{11}$$

With this α we can define our density ρ as

$$\rho = \alpha \rho_0 + (1-\alpha)\rho_1,$$

where ρ_0, ρ_1 are the densities of fluid 0 and fluid 1 respectively.

It is commonly known that these equations can't be solved analytically. Therefore, OpenFOAM uses the Finite Volume Method to discretize all the terms in order to solve the equations. It then uses the PIMPLE algorithm is applied to solve for the variables in every time step. The PIMPLE method is a combination of the SIMPLE and the PISO algorithm.

Additionally, if you want to simulate a turbulent flow you can implement a turbulence model. This is previously described in the part about the file `turbulenceProperties` in the folder `constant`.

A.5.2 Boundary Conditions

The choice of boundary condition can be a hard task, as OpenFOAM has a lot of boundary conditions to choose from. For all variables, type of boundaries and types of flow there are custom made boundary conditions. This sounds really nice, but sometimes you can't see the differences between a set of boundary conditions and it is hard to pick the right one. Moreover, not all boundary conditions are documented very well, so it can take some time to figure out what some of them actually do.

In table A.1, we see the most common boundary conditions for free surface flows. There are a lot more boundary conditions to experiment with, so take a look at the OpenFOAM website to found out which boundary conditions you can make use of.

Name of b.c.	Description	Type of boundary
slip	$\frac{\partial \phi}{\partial n} = 0$	Wall
noSlip	$U = 0$	Wall
fixedValue	$\phi = c$	Various
zeroGradient	$\frac{\partial \phi}{\partial x} = 0$	Various
fixedGradient	$\frac{\partial \phi}{\partial x} = c$	Various
fixedFluxPressure	Sets the pressure gradient such that the flux is equal to that specified by the velocity b.c.	Various
calculated	This b.c. will not be evaluated, it is calculated by the field values	Various
inletOutlet	zeroGradient for outflow, fixedValue for in/backflow	Outlet
outletInlet	Same principle as inletOutlet, but reversed	Inlet
flowRate	Fixes the mass or volumetric flow rate	Inlet/Outlet
variableHeightFlowRate-InletVelocity	Fixes the volumetric flow rate proportional to the phase value α	Inlet
variableHeightFlowRate	Used in combination with the previous b.c. to to constrain the values of α	Inlet
totalPressure	fixedValue for the total pressure ($p + \frac{1}{2}U^2$)	inflow/outflow patches
pressureInletOutletVelocity	A velocity b.c. that can be imposed on boundaries where the pressure is prescribed. ZeroGradient for outflow and same value as the value of the normal component of the internal cell for inflow	inflow/outflow patches

Table A.1: Overview of some of the most common boundary conditions

A.5.3 Computation

With the meshing done and the boundary condition chosen, we can almost start our simulation. We need to set a few parameters first, such as our initial field and the settings for our computation.

At first, we need to specify which part of our domain consists of water and which part of air at our starting time. This is done with a `setFieldsDict` file. In this file you start with setting one of the two phases as default phase. Then you can include a certain shape in your domain, such as a box or sphere and set the value of the phase belonging to the other fluid. Also other variables can initially set by this feature, such as the initial velocity.

For example, suppose your total domain is a box of 10m x 10m x 10m and you want to have the bottom 4 four meters consisting of water. Then you can set the default phase as air with the lines

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);
```

Then for our water phase, we make a box of 10m x 10m x 4m and set the value of alpha to 1:

```
regions
(
    boxToCell
    {
        box (0 0 0) (10 10 4);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
)
```

Now, before you start your solver, you type

```
setFields
```

in the terminal and your initial field will be set.

The last step is to adjust some parameters in the `controlDict` file. You tell OpenFOAM how long your computation needs to be, when it needs to write your data and whether you want an adjustable time step or not. When you are done with all of this, you can start the simulation by typing

```
interFoam
```

or, in the case you want to run it in parallel,

```
mpirun -np <cores> interFoam -parallel
```

in your terminal. Again, when you run your simulation in parallel, you need to to enter the following commands:

```
decomposePar
```

for dividing your domain before you run interFoam and

```
reconstructPar
```

for reconstructing your solution afterwards. During the simulation you see that OpenFOAM writes the folders with the solution according to the time steps you have chosen in your `controlDict` file. When it is finished, you can view your results with ParaView.

A.6 Post-processing

The visualization of the data is an essential part in the analysis of the flow. OpenFOAM pays much attention to this part of the CFD and has many utilities that it can rely on.

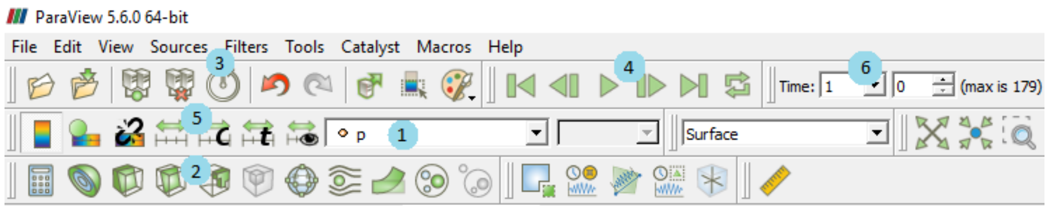
A.6.1 ParaView

The first thing is the strong connectivity with the open source software ParaView. ParaView is one of the most used visualization software programs in the field of CFD. When you install OpenFOAM, a version of ParaView is installed with it. To open your solution in ParaView you can use the OpenFOAM command

```
paraFoam
```

It will create a temporarily file with the extension '.foam' in your OpenFOAM project folder which will be opened in ParaView. This file is empty, but because it is located in your project directory, ParaView can read all your time solutions. Click on 'apply' in the `properties` panel of ParaView to view your solution

In the menu bar on the top of the window, there are a lot of usable options to get a better understanding of your solution. You can see the options menu bar below: We will explain the most



important features.

1: You can select the vector or scalar field you want to visualize in this drop down menu. So every variable that OpenFOAM has calculated at each time step you can find here.

2: It is hard to visualize the whole domain when you use a 3D model. The features in this part of the menu make the visualization process easier. For example, you can create a 2D slice in any direction to visualize the values of a certain variable or you can use the 'contour' option to visualize water height. Another very useful visualization features for flows are the 'glyphs' and 'streamlines' functions. These capture the pattern of the flow very well.

3: The 'filter' drop down menu contains, next to the features treated previously, other ways to visualize and analyse the flow. Especially under the 'Data Analysis' menu are useful ways to plot and calculate data.

4: The animation control panel you use to make a movie of you flow over time. Note that if you have a lot of data points, this might work pretty slow. You can also render it as a video when you save it as an animation (under 'file').

5: This panel is to control the color bar and the scaling of the values of your selected variable. You can also play with opacity or try out different color bars that fits best for your flow.

6: This makes it easier to switch between solutions at different time steps. So you don't need the animation control panel to get to your desired time step.

A.6.2 PostProcess utility

Although ParaView does a decent job in extracting data from your flow, OpenFOAM has its own post processing utilities for this. There are a lot of features in the `postProcess` utility. The whole list can be viewed with the command

```
postProcess -list
```

One of these postProcess functions is `probes`. This function places a probe at a given point in your domain and it will write away the values of the field you give up at each time step at that point. To use that function, you need that file in your `system` folder by typing

```
foamGet probes
```

In this file you can initialize the location of the probe(s) and say which fields you want it to store data from. Then you are almost done. You only need to add the following lines to your `controlDict` file

```
functions
{
    #includeFunc probes
}
```

If you didn't run your simulation yet, the function will gather the data during the simulation. If you already have all your solution files, you can just call the post processing function alone with the command

```
postProcess -func 'probes'
```

and it will gather the data. The data is stored in the newly made folder `postProcessing`. With all these data, you can write a script that loads the data and creates plots for example.

A.7 Final remarks

This appendix does not reflect the full capabilities of OpenFOAM in CFD calculations. It contains the standard steps and most common settings, but there are a lot more options of OpenFOAM that can be exploited. These options can all be found online, whereas all the information is freely accessible. For every version of OpenFOAM, a user guide is published to look into. Although almost every utility of OpenFOAM is mention here, the description of those utilities is not always there or is a bit vague.

To learn OpenFOAM, one can best follow some of the tutorials you can find online or the ones that are in the user guide. If you are stuck with some problem, you can always use the many CFD online discussion websites to ask you question to the online community.

B Boundary conditions

In this section you can find the boundary conditions used for the different models. Note that you can find the meanings of the different boundary conditions in appendix A.5.2.

B.1 Initial boundary conditions

In the table below are the boundary conditions used for the default model (chapter 4).

α	**Inlet:**	variableHeightFlowRate
	Outlet:	zeroGradient
	Atmosphere:	inletOutlet
	Concretewall:	zeroGradient
	Talud:	zeroGradient
	Wall:	zeroGradient
U	**Inlet:**	variableHeightFlowRateInletVelocity of (0, 20, 0)
	Outlet:	inletOutlet
	Atmosphere:	pressureInletOutletVelocity
	Concretewall:	noSlip
	Talud:	noSlip
	Wall:	slip
p_{rgh}	**Inlet:**	fixedFluxPressure, uniform 0
	Outlet:	fixedFluxPressure, uniform 0
	Atmosphere:	totalPressure, uniform 0
	Concretewall:	fixedFluxPressure, uniform 0
	Talud:	fixedFluxPressure, uniform 0
	Wall:	fixedFluxPressure, uniform 0
k	**Inlet:**	fixedValue, uniform 0.001
	Outlet:	zeroGradient
	Atmosphere:	inletOutlet, uniform 0.001
	Concretewall:	kqRWallFunction, uniform 0.001
	Talud:	kqRWallFunction, uniform 0.001
	Wall:	kqRWallFunction, uniform 0.001
ω	**Inlet:**	fixedValue, uniform 0.1
	Outlet:	zeroGradient
	Atmosphere:	inletOutlet, uniform 0.1
	Concretewall:	omegaWallFunction, uniform 0.1
	Talud:	omegaWallFunction, uniform 0.1
	Wall:	omegaWallFunction, uniform 0.1
ν_{turb}	**Inlet:**	fixedValue, uniform 0
	Outlet:	zeroGradient
	Atmosphere:	calculated
	Concretewall:	nutkRoughWallFunction
	Talud:	nutkRoughWallFunction
	Wall:	nutkWallFunction, uniform 0

Table B.1: Boundary conditions for our initial setup

B.2 Improved boundary conditions

In the table below are the boundary conditions used for the models of chapters 5 and 7-9.

α	**Inlet:**	variableHeightFlowRate
	Outlet air:	inletOutlet, uniform 0
	Outlet water:	inletOutlet, uniform 1
	Atmosphere:	inletOutlet, uniform 0
	Concretewall:	zeroGradient
	Talud:	zeroGradient
	Wall:	zeroGradient
U	**Inlet:**	variableHeightFlowRateInletVelocity of (0, 22, 0)
	Outlet air:	fixedValue, uniform (0, 0.1, 0)
	Outlet water:	zeroGradient
	Atmosphere:	zeroGradient
	Concretewall:	noSlip
	Talud:	noSlip
	Wall:	slip
p_{rgh}	**Inlet:**	fixedFluxPressure, uniform 0
	Outlet air:	fixedFluxPressure, uniform 0
	Outlet water:	fixedFluxPressure, uniform 0
	Atmosphere:	totalPressure, uniform 0
	Concretewall:	fixedFluxPressure, uniform 0
	Talud:	fixedFluxPressure, uniform 0
	Wall:	fixedFluxPressure, uniform 0
k	**Inlet:**	fixedValue, uniform 0.001
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	inletOutlet, uniform 0.001
	Concretewall:	kqRWallFunction, uniform 0.001
	Talud:	kqRWallFunction, uniform 0.001
	Wall:	kqRWallFunction, uniform 0.001
ω	**Inlet:**	fixedValue, uniform 0.1
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	inletOutlet, uniform 0.1
	Concretewall:	omegaWallFunction, uniform 0.1
	Talud:	omegaWallFunction, uniform 0.1
	Wall:	omegaWallFunction, uniform 0.1
ν_{turb}	**Inlet:**	fixedValue, uniform 0
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	calculated
	Concretewall:	nutkRoughWallFunction
	Talud:	nutkRoughWallFunction
	Wall:	nutkWallFunction, uniform 0

Table B.2: Improved boundary conditions

B.3 Rigid lid boundary conditions

In the table below are the boundary conditions used for the rigid lid model of chapter 6.

U	**Inlet:**	flowRateInletVelocity of 22
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	slip
	Concretewall:	noSlip
	Talud:	noSlip
	Wall:	slip
p	**Inlet:**	fixedFluxPressure, uniform 0
	Outlet air:	fixedValue, uniform 0
	Outlet water:	fixedValue, uniform 0
	Atmosphere:	fixedFluxPressure, uniform 0
	Concretewall:	fixedFluxPressure, uniform 0
	Talud:	fixedFluxPressure, uniform 0
	Wall:	fixedFluxPressure, uniform 0
k	**Inlet:**	fixedValue, uniform 0.001
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	kqRWallFunction, uniform 0.001
	Concretewall:	kqRWallFunction, uniform 0.001
	Talud:	kqRWallFunction, uniform 0.001
	Wall:	kqRWallFunction, uniform 0.001
ω	**Inlet:**	fixedValue, uniform 0.1
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	omegaWallFunction, uniform 0.1
	Concretewall:	omegaWallFunction, uniform 0.1
	Talud:	omegaWallFunction, uniform 0.1
	Wall:	omegaWallFunction, uniform 0.1
ν_{turb}	**Inlet:**	fixedValue, uniform 0
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	nutkWallFunction, uniform 0
	Concretewall:	nutkRoughWallFunction
	Talud:	nutkRoughWallFunction
	Wall:	nutkWallFunction, uniform 0
$\tilde{\nu}_{turb}$	**Inlet:**	fixedValue, uniform 0
	Outlet air:	zeroGradient
	Outlet water:	zeroGradient
	Atmosphere:	zeroGradient
	Concretewall:	zeroGradient
	Talud:	zeroGradient
	Wall:	zeroGradient

Table B.3: Boundary conditions for the rigid lid model

C Python script

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#Import integrated p_rgh at first time step at the outflow side
path = "D:/Users/SCHI6/Documents/CFD_berekeningen/180s_model/"
df_right = pd.read_csv(path+"tdata/y60/p_rgh_y60.0.csv")

#Select the alpha water and p_rgh columns
df_right = df_right[['alpha.water','p_rgh']]
df_right['p_avg_right'] = np.NaN

#Create a for loop to put all the other time steps in the dataframe
for x in range(1,180):
    df_right2 = pd.read_csv(path+"tdata/y60/p_rgh_y60."+str(x)+".csv")
    df_right = pd.concat([df_right,df_right2[['alpha.water','p_rgh']]],
                         ignore_index=True, sort = False)

df_right = df_right.rename(columns = {'p_rgh':'p_rgh_right'})
df_right = df_right.rename(columns = {'alpha.water':'alpha.water_right'})

#Divide p_rgh by alpha water to find the mean p_rgh over
#the part consisting of water of the whole slice
df_right['p_avg_right'] = df_right['p_rgh_right']/(
        df_right["alpha.water_right"])

#Do the same for the inflow part
df_left = pd.read_csv(path+"tdata/y5/p_rgh_y5.0.csv")
df_left = df_left[['alpha.water','p_rgh']]
df_left['p_avg_left'] = np.NaN

#Create a for loop to put all the other time steps in the dataframe
for x in range(1,180):
    df_left2 = pd.read_csv(path+"tdata/y5/p_rgh_y5."+str(x)+".csv")
    df_left = pd.concat([df_left,df_left2[['alpha.water','p_rgh']]],
                         ignore_index=True, sort = False)

df_left = df_left.rename(columns = {'p_rgh':'p_rgh_left'})
df_left = df_left.rename(columns = {'alpha.water':'alpha.water_left'})

#Divide p_rgh by alpha water to find the mean p_rgh over
#the part consisting of water of the whole slice
df_left['p_avg_left'] = df_left['p_rgh_left']/(df_left["alpha.water_left"])

#load the data of the discharge Q
discharge = pd.read_excel(path+"tdata/discharge.xlsx")

#Merge the dataframes into one dataframe
df = pd.concat([discharge,df_right,df_left], axis = 1)

#Variables
Area = 5
```

```
rho = 1000

#Calculate the contraction coefficient with the integrated data
df['mu1'] = df.Q / (Area*np.sqrt(2*(df.p_avg_left - df.p_avg_right)/rho))
df['mu1_rmean'] = np.NaN

#Calculate the moving average of the contraction coefficient with the
#integrated data
for row in range(15,164):
    df['mu1_rmean'].iloc[row] = df['mu1'].iloc[row-15:row+16].sum()/31

#Calculate the contraction coefficient with the probed data
df['mu2'] = df.Q / (Area*np.sqrt(2*(df.probe_left - df.probe_right)/rho))
df['mu2_rmean'] = np.NaN

#Calculate the moving average of the contraction coefficient with the probed
#data
for row in range(15,164):
    df['mu2_rmean'].iloc[row] = df['mu2'].iloc[row-15:row+16].sum()/31

#Calculate the loss coefficient over the model
df['xi1'] = np.square((1-1/df.mu1))
df['xi1_rmean'] = np.square((1-1/df.mu1_rmean))
df['xi2'] = np.square((1-1/df.mu2))
df['xi2_rmean'] = np.square((1-1/df.mu2_rmean))

#Plot the two graphs
plt.plot(df['t'],df['mu1_rmean'], color = 'red',
         label = r' integrated $\mu$ (running mean)')
plt.plot(df['t'], df['mu1'],linestyle = 'dotted', color = 'red',
         label = r' integrated $\mu$')
plt.plot(df['t'],df['mu2_rmean'], color = 'blue',
         label = r'probed $\mu$ (running mean)')
plt.plot(df['t'], df['mu2'],linestyle = 'dotted', color = 'blue',
         label = r' probed $\mu$')
plt.xlabel('t')
plt.ylabel(r'$\mu$')
plt.axis([0, 180, 0.65,0.85])
plt.xticks(np.arange(0,180,step = 10))
plt.yticks(np.arange(0.65,0.85,step = 0.01))
plt.grid()
plt.legend()
plt.show()
plt.plot(df['t'],df['xi1_rmean'], color = 'red',
         label = r' integrated $\xi$ (running mean)')
plt.plot(df['t'],df['xi1'], color = 'red', linestyle = 'dotted',
         label = r' integrated $\xi$')
plt.plot(df['t'],df['xi2_rmean'], color = 'blue',
         label = r' probed $\xi$ (running mean)')
plt.plot(df['t'],df['xi2'], color = 'blue', linestyle = 'dotted',
         label = r' probed $\xi$')
plt.xlabel('t')
plt.ylabel(r'$\xi$')
plt.axis([0, 180, 0.05,0.35])
plt.xticks(np.arange(0,180,step = 10))
plt.yticks(np.arange(0.05,0.35,step = 0.02))
plt.grid()
plt.legend()
plt.show()
```