Technical University of Munich

Chair of Hydromechanics

Analysis of the Performance of interFoam and shallowFoam Simulating the Steady Wave at the Floßlände

Bachelor Thesis

December 2018

Nikolas Carlos Manuel Vornehm

matriculation number: 03671714

attainment of the academic degree

"Bachelor of Science"

degree programme:

Environmental Engineering

Faculty for Civil, Geo and Environmental Engineering

Supervisor: Hao Zeng, M.Sc., Chair of Hydromechanics

Examiner of the Bachelor Thesis: Univ.-Prof. Dr.-Ing. Habil. Michael Manhart

Abstract

The main objective of this work is the evaluation of the performance of OpenFOAMs 3D solver `interFoam` and the newly developed 2D solver `shallowFoam` on simulating a steady wave. The phenomenon to be simulated is the steady surfing wave based at the Floßlände located in the south of Munich.

This thesis contains short introductions to OpenFOAM itself and the solvers `interFoam` and `shallowFoam`. A detailed description is given of how the simulations are established. The geometry of the channel in which the wave is situated is explained in detail. It serves the understanding of both, the 3D and the 2D mesh generation.

The 3D mesh is generated in 2 stages: the premesh generation with the utility `blockMesh` and the final mesh generation with the utility `snappyHexMesh`. The premesh generation with `blockMesh` builds a mesh in the shape of a block. That is then being shaped by `snappyHexMesh`. `snappyHexMesh` uses a pre-drawn model (in *.stl format) of the shape that needs to be cut out of the block-shaped-mesh. The implementation of boundary conditions and initial conditions follows. They are explicitly chosen to mimic the actual wave regarding the meshes properties. The numerical schemes used for the simulation in 3D are briefly introduced. A main problem of the simulation are wiggles that appear in the section of supercritical flow. Various schemes are used on the momentum flux and the mass flux and its results are compared. The most promising scheme turns out to be the very stable upwind scheme. Its disadvantage of dispersion is then be fixed by compressing the air-water interface. The two turbulence models k-epsilon and k-omegaSST are briefly explained and compared to identify which one performs better for this event. It results that the k-omegaSST produces outcomes that are more realistic.

The simulation is done with five different discharges that suit the discharges of the physical model done by Ruth Morandi for her Bachelor Thesis. The results are compared at three main locations: the flow depth at the lowest and highest point of the wave, which are important to evaluate for surfing suitability, and the depth of the outlet water. It concludes that the results are similar enough to deduce the simulation to be accurate.

The simulation is then analysed regarding the Total Energy. Unfortunately the simulation fails to produce useful results for the Total Energy calculated with Bernoulli's Equation. Some disrealities can be explained due to low resolution. The main reason, however, for why the calculations are unsuccessful remains unknown.

The last part is a description of the development of the 2D simulation. The solver `shallowFoam` neglects the vertical dimension. Therefore, it is possible that the solver has trouble to simulate flows that include vertical velocities such as waves. Nevertheless, the results are compared to those of a 3D simulation to evaluate the solvers accuracy in sections where the vertical velocity can be neglected.

Zusammenfassung

Die Hauptaufgabe dieser Arbeit ist die Evaluierung der Ausführung des 3D Solvers `interFoam` und des 2D Solvers `shallowFoam`. Beider solver sind Teil der OpenFOAM library. Simuliert wird die stehende Surfer welle an der Floßlände, in Thalkirchen, im Süden Münchens.

Diese Arbeit enthält eine kurze Einführung zu OpenFOAM und den solvern `interFoam` und `shallowFoam`. Es wird auch im Detail erklärt wie die Simulationen entwickelt wurden. Die Geometrie des Ländkanals in dem sich die Surfer Welle befindet wird ausführlich zur Schau gestellt. Dies dient dem besseren Verständnis der Entwicklung des 3D und des 2D Meshs.

Das um einiges kompliziertere 3D Mesh wird in 2 Phasen erstellt: die Erstellung des Pre-Mesh mit der Utility `blockMesh` und die Mesh Verfeinerung mit der Utility `snappyHexMesh`. Wie der Name verrät, erstellt `blockMesh` eine Mesh in Form eines Quaders oder Blocks. Dieser Quader wird anschließend mit `snappyHexMesh` verfeinert. Hierfür wird eine 3D Zeichnung eines Volumens im *.stl Format verwendet. Das Volumen wird dann vom Ursprünglichen Quader subtrahiert und übrig bleibt das Kontrollvolumen der Strömung. Die Anfangs- und Randbedingungen werden den Eigenschaften des Meshs angepasst um möglichst realitätsnah simulieren zu können. Hierfür sind die numerischen Schemen von größter Wichtigkeit. Eines der Hauptprobleme der 3D Simulation sind die Formierung kleiner Oberflächenwellen. Es stellt sich heraus, dass das upwind differencing am erfolgreichsten ist. Jedoch leidet dabei die Simulation unter Dispersion der Wasseroberfläche. Um dem entgegen zu wirken wird das Interface komprimiert. Die zwei Turbulenz Modelle die verglichen werden sind das k-epsilon und das k-omegaSST Modell, wobei letzteres bessere Ergebnisse produziert.

Um die Simulation mit den Ergebnissen des physikalischem Modells erstellt bei Ruth Morandi zu vergleichen wird die sie mit 5 verschiedenen Durchflüssen getestet. Verglichen wird die Fließtiefe an 3 verschiedenen Stellen: der tiefste und der höchste Punkt der Welle, welche wichtig der Evaluierung dessen Surfbarkeit ist, und die Fließtiefe des Unterwassers. Man erkennt, dass die Ergebnisse sich günstig ähneln und demnach die Simulation als realistisch angenommen werden kann.

Die Simulation wird nun anhand der Bernoulli Gleichung für Gerinne Strömungen analysiert. Die Ergebnisse erweisen sich jedoch als nicht-physikalisch. Einige der Ergebnisse lassen sich auf die Sampling Auflösung zurückschließen. Im allgemeinen bleibt der Grund jedoch unbekannt.

Der letzte Teil beschreibt die Entwicklung der 2D Simulation. Der 2D Solver `shallowFoam` vernachlässigt die vertikale Dimension. Daher könnte der Solver Probleme habe Strömungen zu simulieren welche vertikale Geschwindigkeiten beinhalten wie z.B. Wellen. Dennoch werden die Ergebnisse mit denen der 3D Simulation verglichen um klarzustellen in welchen Abschnitten der Strömung die Simulation physikalische Ergebnisse produziert.

Table of Contents

List of Figures

List of Tables

Notation

A	Cross-sectional area	[m^2]
Co	Courant number	[$-$]
f	Face centre	[$-$]
f	Any function	[$-$]
$f_{\sigma i}$	Surface tension	[$\frac{N}{m}$]
g	Gravitational acceleration	[$\frac{m}{s^2}$]
h	Flow depth	[m]
h_c	Critical flow depth	[m]
h_w	Wave height	[m]
I_s	Slope	[%]
k	User input	[$-$]
k_{st}	Strickler value	[$\frac{m^{\frac{1}{3}}}{s}$]
L	Length of a flow	[m]
l_w	Wave length	[m]
M	Mesh height	[m]
p	pressure	[$\frac{N}{m^2}$]
P	Cell centre	[$-$]
q	Specific discharge	[$\frac{m^2}{s}$]

Q	Discharge	$[\frac{m^3}{s}]$
$\boldsymbol{S}$**:**	surface vector	[–]
t	time	[s]
u	Velocity	$[\frac{m}{s}]$
$\boldsymbol{U}$	Velocity magnitude	$[\frac{m}{s}]$
V	Volume	[m^3]
V_{cell}:	Volume of cell	[m^3]
x	x-coordinate	[m]
z_b	z-coordinate bottom elevation	[m]
z_{cell}	z-coordinate of cell centre	[m]
z_w	z-coordinate of water surface	[m]

Greek Letters

α	Indicator function	[–]
Γ	Solution of issue	[varies]
κ	Count of timestep	[–]
ν'	Turbulent viscosity	$[\frac{m^2}{s}]$
ρ	Density	$[\frac{kg}{m^3}]$
τ	Duration of timestep	[s]

τ_{bi}	ith component of bed stress	$[\frac{kg}{m*s}]$
τ_{ij}	Lateral stresses in shallow water equations	$[\frac{kg}{m*s}]$
$\tau_{t_{ij}}$	Turbulent stress	$[\frac{N}{m^2}]$
$\tau_{v_{ij}}$	Viscous stress	$[\frac{N}{m^2}]$
υ	Solution for a certain time in Euler scheme	[–]
ϕ	Any quantity	[varies]
ϕ_p	Positive face flux	$[\frac{kg*m^3}{s}]$
ϕ_n	Negative face flux	$[\frac{kg*m^3}{s}]$

Abbreviations

2D	Two – dimensional
3D	Three – dimensional
alpha1	α
BC	Boundary condition
CFD	Computational Fluid Dynamics
epsilon	ε (rate of dissipation of kinetic energy)
FVM	Finite Volume Method
H	z_w (z-coordinate of water surface)
HU	q (specific discharge)

IC	Initial condition
k	Turbulent kinetic energy
kst	k_{st} (strickler value)
nut	ν' (turbulent viscosity)
omega	ω (specific rate of dissipation)
OpenFOAM	Open Source Field Operation and Manipulation
RANS	Reynolds – average Navier – Stokes
S	z_b (z-coordinate bottom elevation)
SST	Shear Stress Transport
U	u (velocity)
VoF	Volume of Fluid

1. Introduction

The Floßlände is a raft docking station located in the south of Munich (Thalkirchen). The Floßlände is originally designed to produce supercritical flow and then turn into a hydraulic jump. Such circumstances happen to be very suitable for surfing. Being the reason why in the 1970s river-surfing was invented at this location [1].

Figure 1: Surfers waiting for raft to pass the Floßlände [2]

Originally the Floßlände was built to serve the goods traffic and transporting of wood along the Isar. The rafts nowadays are a tourist attraction, where they are taken on a trip from Wolfratshausen to Thalkirchen with a brass band on board (Figure 1) [3].

Surfing conditions were at its peak between the years 2002 and 2006. The wave was easy to surf at all times from April to September. The change from a mechanic to an electric discharge control system and a change of subsoil caused the discharge to decrease [4]. In addition to that a hydropower plant located upstream the Floßlände now also consumes a significant amount of discharge. As a result, the wave was barely generated and decent to surf only on seldom occasions [3]. Some surfers attempted to improve the wave by narrowing the channel with a self-build sleeve construction to increase specific discharge. As the event of a raft passing the construction happens fairly frequently and requires the entire channel width, the sleeve

construction was supposed to be embedded and removed quickly. However, the construction wouldn't preserve for long. Due to the force of the current it would rupture and therefore, the wave stayed flat [5]. The effort done by personal individuals in their spare time shows the importance of the wave for the people. That's why in the year 2015 a stationary lip was installed. The installation of the lip was realized due to the Interest Group Surfing in Munich working closely together with the municipality of Munich. The stationary lip now provides a wave which is manageable to surf well even at low discharges [1].

The history of this wave and the effort made by people involved shows the importance of such free time activities to keep a healthy society. People desire to surf without having to relocate closer to the sea shore. Munich has a total of three steady river waves to surf on which are very busy at all times of the day and at any time of the year. The demand for more waves exist and needs to be taken serious. This thesis is dedicated to all river-surfers. It has the objective of contributing to surfing wave technology and make surfing even more available for people living far away from a coastline.

2. Theoretical fundamentals

2.1 The stationary wave

The Theoretical fundamentals are extracted from Ruth Morandis Bachelor Thesis, “Maßstabsähnliche Untersuchungen zur stationären Welle an der Floßlände”(Scale-like investigation of the stationary wave at the Floßlände), also written at the Chair of Hydromechanics at the Technical University of Munich. Ruth Morandi has done research on the steady wave at the Floßlände based on a physical model with a ratio of 1:18.

A steady wave usually used to be a phenomenon of inaccuracy in hydraulic engineering. When supercritical flow is generated after a weir for example, a hydraulic jump is needed to turn it back into subcritical flow. From the hydraulic engineering point of view an efficient energy conversion in a stilling basin is most important. That efficient energy conversion can be deadly for humans due to the massive counter current. Nevertheless, when the combination of flow velocity, water depth and height of the sill at the end of the stilling basin match, a steady wave can arise [7].

Not only a hydraulic jump can be suitable for surfing but a so called “sheet flow”, too. The sheet flow occurs when overflowing a corresponding bottom geometry. The Floßlände is a combination of both. It has a stationary lip for the water to flow over and causes a hydraulic jump just afterwards. That combination creates its wave geometry [7].

The wave of issue is a “facing wave”, which is a continuous wave with a smooth surface. It is very suitable for surfing. The opposite is a “white water wave” which mostly consists of foam [7].

2.2 Suitability for Surfing

The goal of surfing is to balance on the wave. That state of balance is reached when the surfers gravitational force and the waters momentum are in equality. Due to a waves smooth surface between its valley and peak a surfer is able to glide downhill, whilest the water runs uphill, contrary to the surfers movement. That holds him in abeyance.

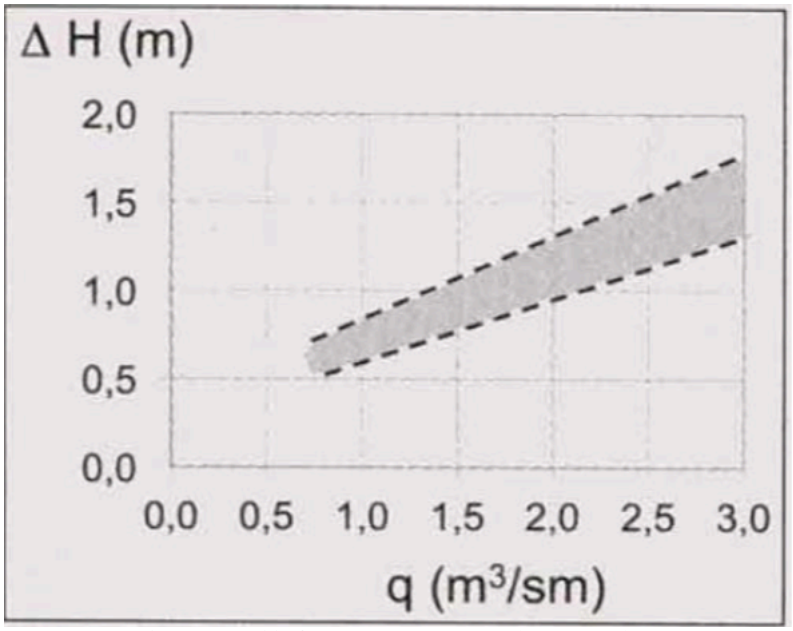

Figure 2: Conditions suitable for a wave generation (grey space).

The criteria needed to generate a steady wave are: A cross construction i.e. a ramp with a rough surface, a height difference of the inlet and outlet water surface of $0.5\,m < \Delta h < 1.2\,m$ and enough specific discharge $q > 1\,\frac{m^3}{sm}$. The domain for a steady wave is quite narrow. As shown in Figure 2 the Δh and q ratio needs to match the requirements accurately [7].

A wave can be described by its size and shape, which are measured by the wave height h_w and the wavelength l_w, which are illustrated in Figure 3. The wave height should be between 50 and 150 cm of height and should have a slope between 30%-80% to be suitable for surfing.

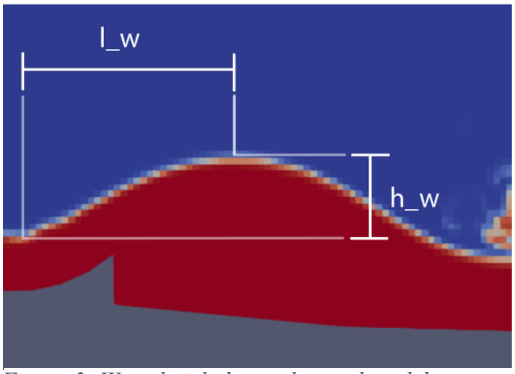

Figure 3: Wave height h_w and wave length l_w.

Another indicator surfing suitability is the Froude number Fr:

$$Fr = \frac{v}{g*y} \tag{2.1}$$

If $Fr < 1$, the flow is subcritical, which means that surface waves can spread faster than the water flows. If $Fr > 1$ the flow is in supercritical state, where no surface wave can move upstream. Fr should be between 1.7 and 4.5 at the waves lowest point, otherwise a normal hydraulic jump occurs instead.

3. Simulation setups

This section will present the setups for the simulations as well as the OpenFOAM framework which is used in this study. OpenFOAM is one of the most popular open source CFD-Software platforms. It was created in 2004 by the company OpenCFD Ltd and is still being professionally updated on a regular basis including contributors from the community. Engineers and scientists from commercial companies as well as academic institutes form a very engaged user base. The OpenFOAM library contains a wide range of solvers and utilities to resolve complex problems such as fluid flows including turbulence, chemical reaction and heat transfer as well as solid mechanics, acoustics and electromagnetic problems [15].

All solvers unsed in this thesis are developed with OpenFOAM.

3.1 Mesh

3.1.1 Geometry

In Figure 4 in can be observed, that the Floßlände forms the end of the raft channel. After the water passes it, it flows into a lake. Only the Floßländes geometry will be considered for the simulation. That includes the 8 m wide and 24.1 m long channel. All coordinates used to model the channel are measured by BSc. Ruth Morandi [6][7]. A model of the channel is displayed in

Figure 4: 3-D view on Floßlände (Google Maps).

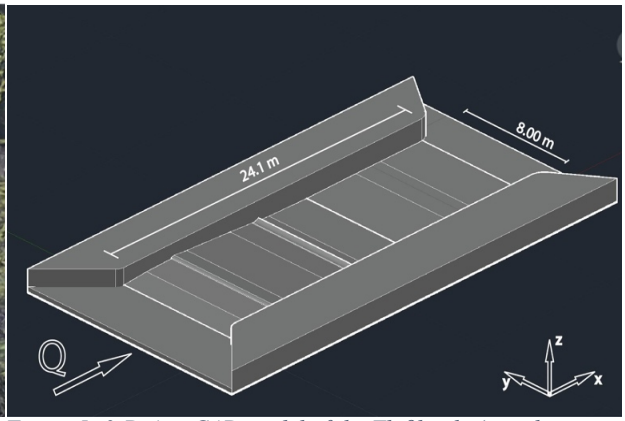

Figure 5: 3-D AutoCAD model of the Floßlände (coordinates from [6][7]).

Figure 5.

The channels slopes are the crucial features of its geometry. As shown in Figure 6, the channel consists of nine slopes and two features. The first feature is a sill that separates slope 2 and 3. The sills quarter elliptical geometry is elevated from slope 2 and connects smoothly to slope 3. The second feature is the stationary lip located on slope 5 [6].

The sill causes the flow to become supercritical and the stationary lip has the purpose of generating the stationary wave. The hydraulic jump occurs after the stationary lip in slope 6.

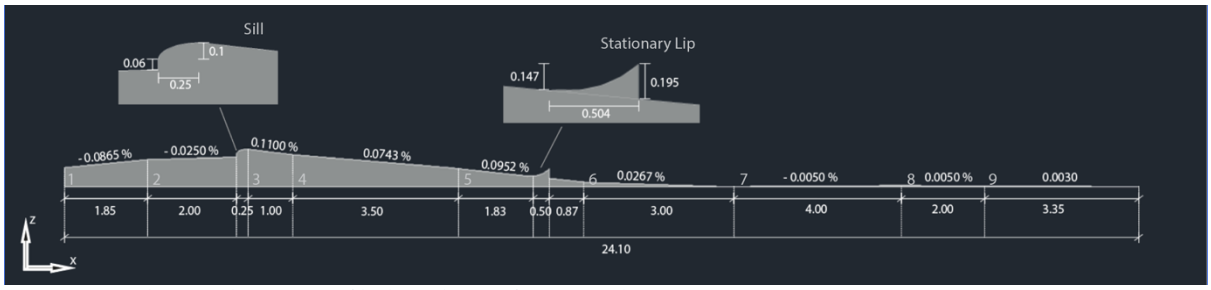

Figure 6: 2-D bottom elevation profile (coordinates from [6][7]).

3.1.2 3D – Mesh

The mesh for this case will be created with two OpenFOAM utilities: `blockMesh` and `snappyHexMesh`. At first a background mesh will be created with `blockMesh` and subsequently shaped with `snappyHexMesh` into its final structure. At first a prototype mesh is generated that is further refined in the stage of mesh conversion in chapter 3.2.6.

blockMesh

`blockMesh` is a very basic mesh generation utility in the OpenFOAM library. All its inputs are written down in one file only called “blockMeshDict”. As an exception it is usually kept inside the folder *constant/polyMesh/blockMeshDict* instead of in *system/*, where the dictionary files are usually kept. Several things should be defined in the `blockMesh`-dictionary [8]:

1. all reference vertices of the mesh:
   (0 /1 / 0), (24.1 / 1 / 0), (24.1 / 1.2 / 0), (0 / 1.2 / 0), (0 / 1 / 2), (24.1 / 1 / 2), (24.1/ 1.2 / 2), (0 / 1.2 / 2) [7].
2. all blocks forming the entire mesh (each consisting of 8 vertices), the grading definition and the amount of cells in x, y and z direction:
   the 3D mesh only consists of one block, simpleGrading (1 1 1) and the cell count(x,y,z) = (120,1,10) (for the prototype mesh)
3. special edge definitions such as arcs and splines: not used here
4. faces of blocks assigned to patch names:
   inlet, outlet, left, right, bottomWall, atmosphere
5. special face merging: not used here

[8]

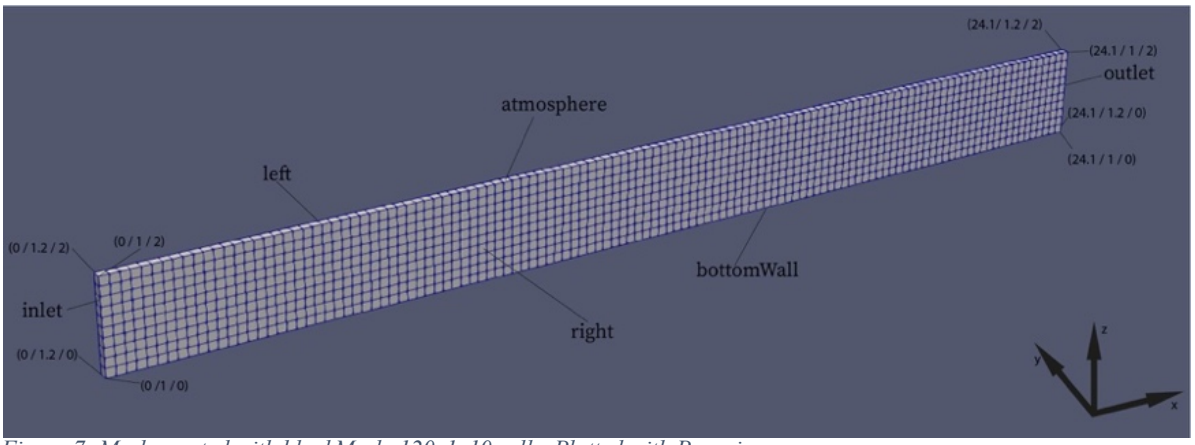

Figure 7: Mesh created with blockMesh. 120x1x10 cells. Plotted with Paraview.

The result is a 3D block-shaped mesh. It can be observed in Figure 7. All visual results of this work are plotted with the third party software Paraview. In this case the assumption is made, that the velocities in the y-direction can be neglected. As stated in chapter 3.1.1 the observed channel is straight and does not contain any curves. Hence, the velocities in y-direction are insignificantly small and it is sufficient to have only one cell in the y-direction. As consequence the total amount of cells of the prototype mesh is reduced from 48000 (for a real 3D mesh with the dimensions: 24.1 m * 8 m * 2 m) to 1200 (pseudo 3D mesh with the dimensions: 24.1 m * 0.2 m * 2 m). The mesh is reduced to 2.5% of the size and saves copious amounts of computation time. This will also be an advantage during the stage of convergence in chapter 3.2.6.

snappyHexMesh

For the mesh to become useful it needs to be shaped into the geometry described in chapter 3.1.1. This will be done with the utility `snappyHexMesh`.

`snappyHexMesh` is a mesh shaping utility that is also included in the OpenFOAM library. Based on the very well defined dictionary named snappyHexMeshDict, which is usually kept in *system/snappyHexMeshDict*, it shapes an already existing mesh into a detailed geometry. The geometry itself is defined in a *.stl or an *.obj file usually kept in the folder *constant/triSurface/*.stl* [9]. The geometry file used here is called riverSlope.stl. It is drawn with AutoCAD 2018 (Figure 8). The path to find the slopes drawing is *constant/triSurface/riverSlope.stl*.

`snappyHexMesh` has problems with shaping sharp edges. In the case of it becoming a serious problem one can extract feature-edges with a utility called `surfaceFeatureExtract`, which is based on the simple dictionary file surfaceFeatureExtractDict, kept in

system/surfaceFeatureExtractDict. The way the extracted features will be treated in `snappyHexMesh` is defined in the snappyHexMeshDict [9]. However, for the Floßlände it is not necessary to extract any features.

`snappyHexMesh` proceeds by subtracting the geometry from riverSlope.stl from the original mesh, it can be specified whether the subtracted or the leftover part is going to be used for further proceeding [9]. For this simulation the geometry of the slope, which is shown in Figure 8, is subtracted from the mesh created with `blockMesh`. This is done in the first of three stages of the meshing process called `castellatedMesh`.

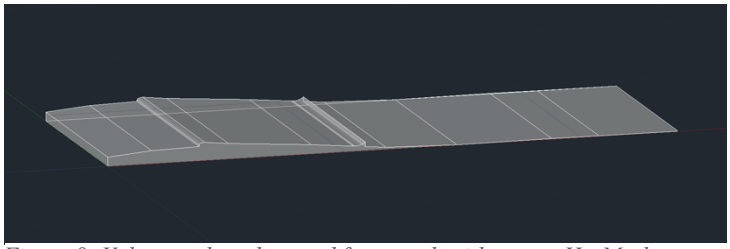

Figure 8: Volume to be subtracted from mesh with snappyHexMesh.

`castellatedMesh` refines the mesh at the surface intersection of the original mesh and the geometry of riverSlope.stl. As mentioned the slope volume is removed and the volume through which the water flows is kept. The result is shown in Figure 9. It is a mesh that still only consists of the original hexahedra and the refined cells. No cells are added or reshaped at this stage [10].

The second stage is called `snap`. Cell vertex points are moved onto the surface geometry to replace the jagged surface generated by `castellatedMesh` with a smooth one. This happens in the following steps [10]: At first the vertices in the jagged boundary are shifted onto the riverSlope.stl surface. Secondly using the latest shifted boundary vertices, snap resolves the relaxation of the internal mesh. Thirdly determine the vertices that trigger the mesh quality parameters to be tampered with. Lastly diminish the displacement of those vertices from their initial value, which is 1, and continue to repeat from 2 until mesh quality is of satisfaction [10]. Figure 10 shows its result.

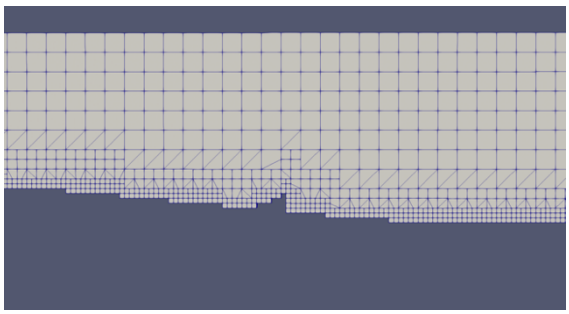

Figure 9: Mesh after castellatedMesh.

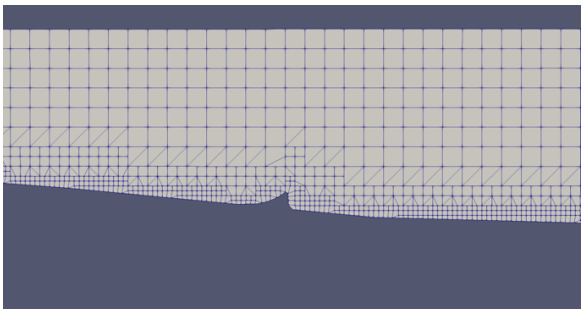

Figure 10: Mesh after snap.

The `addLayers` process is the last stage of `snappyHexMesh` and is an optional process that inserts additional layers consisting of hexahedral cells along the boundary surface. The snapping stage produces irregular cells by the boundary surface which is improved by adding those new cell

layers [10]. Since here the boundary layer is not of importance, the output mesh of the snapping stage serves the purpose of this study, which is the reason why the `addLayers` process is not executed.

After finishing all relevant stages `snappyHexMesh` runs a quality control, whose parameters are also set in the snappyHexMeshDict [10].

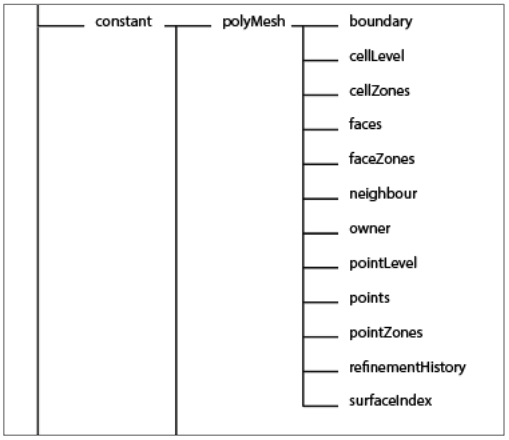

Figure 11: File structure inside polyMesh after snappyHexMesh.

For every stage of `snappyHexMesh` (`castellatedMesh`, `snap` and `addLayers`) a new folder is created containing a *polyMesh/* directory. As mentioned above `addLayers` is not used in this study, which means only two new folders are created: *0.002/polyMesh/* and *0.004/polyMesh/*. The name of the folders are defined by the timestep setting in *system/controlDict/*, which in this case is set to 0.002 (this will be further explained in chapter 3.2.3). Each timestep represents one of the `snappyHexMesh` stages [10]. The mesh used for further proceeding is the snapped one which is found in the second directory, *0.004/polyMesh/*. The *0.004/polyMesh/* directory will now be replaced with the one in *constant/polyMesh*. After it has been copied the directories *0.002/* and *0.004/* are of no longer use and can be deleted. All files included in the new *constant/polyMesh/* can be observed in Figure 11. The new mesh is shown in Figure 12. Since the patch bottomWall is now completely replaced by the patch riverSlope_riverSlope, all setting for bottomWall must be deleted out of the other all filesinside *0/* directory.

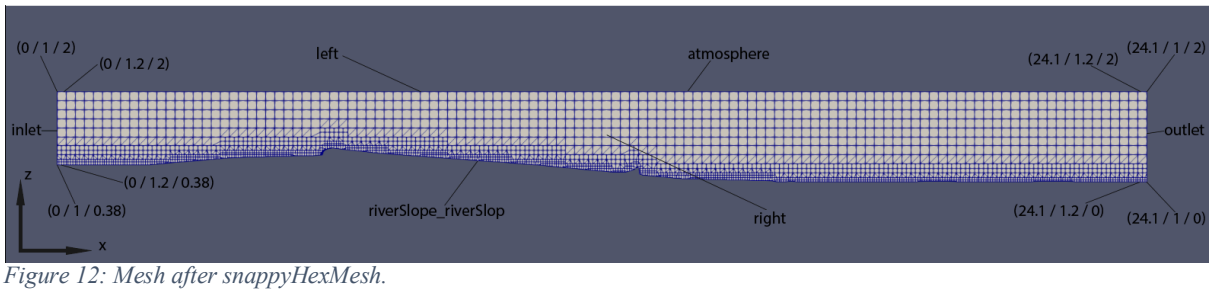

Figure 12: Mesh after snappyHexMesh.

3.1.3 2D – Mesh

The mesh for the 2D simulation is exclusively done with `blockMesh` and can be observed in Figure 13. The y-dimension can be neglected, due to the channels geometry. Therefore, the width of the mesh is kept at $0.1\ m$. Technically speaking it becomes a 1D simulation. Since the vertical dimension is neglected, the mesh should have an overall height of $1\ m$ [29]. The length of the mesh is as long as the channel itself: $24.1\ m$. The mesh is split into 6 different

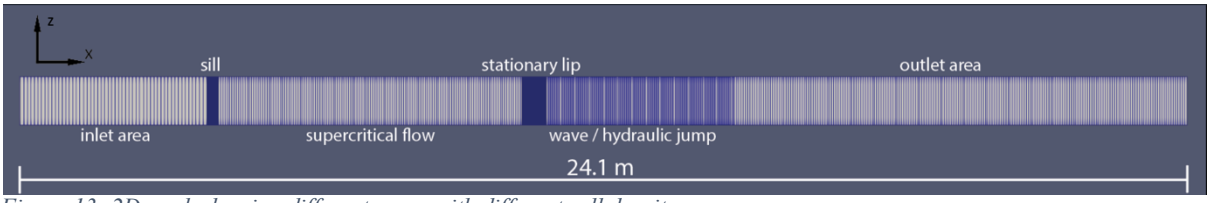

Figure 13: 2D mesh showing different areas with different cell density.

areas with different cell densities. The cell density of an area depends on the complexity of the flow and the bottom geometry e.g. the cells by the sill and the stationary lip are smaller than elsewhere.

This mesh has 2539 cells in x-direction and 1 cell in y-direction and z-direction. The inlet and the outlet boundary are defined as patches. All the other boundaries are provided with the boundary condition `empty`.

3.2 3D – solver interFoam

The 3D-Simulation is done with OpenFOAMs multiphase flow solver `interFoam`. The solver is included in OpenFOAMs C++ libraries and utilities and continues to gain popularity in the multiphase research community [11].

The Navier-Stokes-equation is solved by `interFoam` for two incompressible, isothermal, immiscible fluids. Therefore, the fluids properties are persistent within the entire space occupied by one of the two fluids. An exception is extant at the interface where the two fluids meet within a cell. Hence `interFoam` is making use of the Volume-of-Fluid-Mothod (VoF) [12].

VoF is a method used in numeric fluid mechanics to model two-phase flow problems. In this method an additional variable α is introduced, which determines the proportion of volume of a fluid in one cell, whereas α is 1 within the first fluid and 0 inside the second fluid. Only at the interface does its value vary between 0 and 1 [13].

3.2.1 Governing Equations

The Conservation of Mass Equation for constant density (incompressible):

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{5.1}$$

The Momentum Equation for laminar and turbulent flow:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho u_j u_i\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\tau_{v_{ij}} + \tau_{t_{ij}}\right) + \rho g + f_{\sigma i} \tag{5.2}$$

Whereas the density ρ is calculated as

$$\rho = \alpha\rho_1 + (1-\alpha)\rho_2 \tag{5.3}$$

The two fluids are immiscible, which results in a sharp interface. Due to the VoF the last equation needs to be solved to grant the interfaces location to be found. It is the transport equation for α:

$$\frac{\partial \alpha}{\partial t} + \frac{\partial(\alpha u_j)}{\partial x_j} = 0 \tag{5.4}$$

[12]

Since `interFoam` is an isothermal solver, no energy equation is solved [12].

A 3D-simulation made with `interFoam` consists of three main directories: *0/*; *constant/* and *system/*. The simulation can be pictured as a video, but instead of taking many pictures with minimal time difference, the solver is given the first "picture" of the "video" out of which it computes the second out which it computes the third and on. The state at the time $t = 0$ could described as the first picture. All parameters describing the initial state are defined in *0/*. It includes all boundary conditions (BC) and initial conditions (IC). All parameters which remain constant throughout the whole simulation are determined in *constant/*, such as: geometry, turbulence model, transport properties and the gravitational acceleration. All files that dictate the system on how to work are saved in *system/*. These files can either dictate a utility (e.g. `snappyHexMesh`, `decomposePar`) or command `interFoam` itself on how to calculate, e.g. the timestep

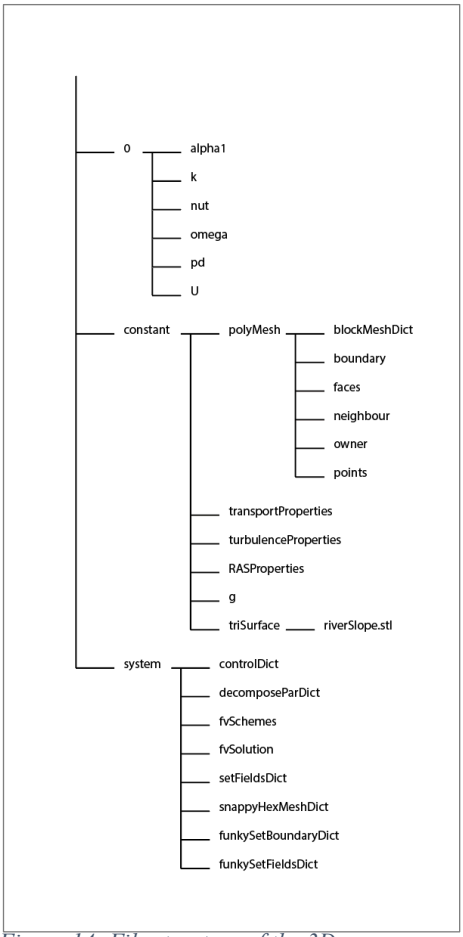

Figure 14: File structure of the 3D-simulation.

between the “pictures” [18]. An overview of the file structure at this stage is displayed in Figure 14. All files are attached to this thesis.

3.2.2 Boundary Conditions (BC) and Initial Conditions (IC)

The IC and BC are defined for every cell and every boundary face for α (described in, u (velocity), ν' (turbulent vicousity), k, ω/ε ($k-\omega$SST /$k-\varepsilon$ turbulence model) and pd (pressure). By setting IC and BC one tries to achieve two goals: firstly create an environment which is as close to physical reality as possible. If this is not the case it is probable that the simulation will stop due to a rapid rising Co, which consequence is a hasty timestep decrease, since `adjustableTimeStep` is set. This setting adjust the timestep during runtime to keep the simulation accurate and shorten computation time (will be explained further in chapter 3.2.3). If the timestep becomes to small `interFoam` will stop computing, otherwise the computation time becomes enormously long. As this is an analysis about a steady wave, the second goal is to set the IC and BC in a way to reach steady state as soon as possible to save computational effort. Once steady state is reached there is no need to keep computing, which means the sooner steady state is reached, less computation time is required.

All of the above mentioned parameters (α, ν', k, ω, p, u) are defined in individual files. At the top of each file is an option to define the internalField (includes all cells) as `uniform` or `nonuniform`. By using the `uniform` option one value can be set which will be applied on every cell throughout the entire mesh, excluding the boundary patches. When using the `nonuniform` option `interFoam` expects a list containing values for all cells. The list can be made by an OpenFOAM utility called `setFields`. In this case the variables for which internalField is set to `nonuniform`, the list is made with the swak4Foam utility `funkySetFields`. It is a better, more accurate version of `setFields`.

The boundaries are usually defined as constant throughout a boundary patch. If this needs to be defined more accurately, similar to the internalField, a list is required. The lists that are needed for the BC are made with another swak4Foam utility called `funkySetBoudary`.

Swak4Foam is an additional C++ library for the OpenFOAM library that contains a number of utilities, boundary conditions and function objects. These enable the user to set a simulation more accurate without having to program oneself [14].

The IC/BC that require a list and cannot stay uniform are α and u. `funkySetFields` permits the user to manipulate any scalar or vector field defined in a file and `funkySetBoudary` can set on a boundary patch any kind of field [14]. Together they allow to set any scalar field and vector

field throughout the entire mesh including the boundaries. The expressions that define the fields are set in *system/funkySetFieldsDict/* and *system/funkySetBoudaryDict*.

The discharge is determined by α and u. Since α determines which of the cells contain water, it also determines the cross sectional flow area. The discharge Q is calculated as

$$Q = A * u \tag{5.5}$$

The flow depth-discharge-relation has been previously investigated by Dipl.-Ing Johann Obert, from which it is possible to derive the IC and BC for α and u [6]. It is displayed in Figure 15.

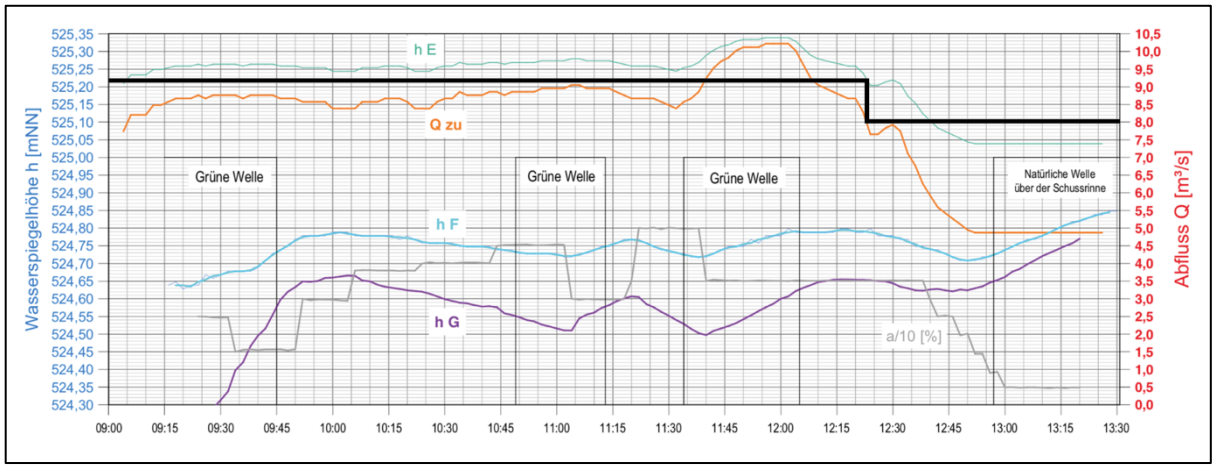

Figure 15: Diagram picturing flow depth-discharge-relation over time (h E is the location at the beginning of the Floßlände channel) [6] (translated from German).

For the prototype mesh a discharge of $8.003\ m^3/s$ is set. As shown in the diagram at this discharge the interface at the channel entry is located at $525.22\ m\ a.s.l.$ [6]. The channels bottom at this location is located at $524.20\ m\ a.s.l.$ [7]. Therefore,

$$A = b * h = 8\ m\ (525.22\ m - 524.20\ m) = 8\ m * 1.02\ m = 8.12\ m^2 \tag{5.6}$$

and

$$u_x = \frac{Q}{A} = \frac{8.003\ m^3/s}{8.12\ m^2} \approx 0.9856\ m/s \tag{5.7}$$

Reading off the diagram is rather inaccurate. An alternative option are the measurements taken of the physical model by Ruth Morandi. Her measurement converted to the real size geometry results in an inlet flow depth of $h = 1.0168\ m$ [7]. Therefore, the horizontal inlet velocity u_x results in

$$u_x = \frac{Q}{b*h} = \frac{8.003\ m^3/s}{8\ m*1.0168\ m} \approx 0.9839\ m/s \tag{5.8}$$

Since the objective of this work is the comparison to the physical model, one would agree to use its data instead of the measurements taken by Dipl. Ing. Johann Obert.

alpha1

Since alpha1 contains the information about the division of water and air, it is the first file to be manipulated. As mentioned above, one attempts to reach steady state as fast as possible and therefore, tries to place water in the mesh as close to what steady state would be achieved as. To set up the IC with a scalar field α `funkySetFields` needs an expression written in C++ [16]:

```
((pos().x <= 4.1  &&  pos().z < 1.3968) ||…

(pos().x > 4.1  && pos().x <= 12  && …

pos().z < 1.3968 - (pos().x - 4.1) * 0.09) ||…

(pos().x >  12 && pos().z < 0.75) ? 1 : 0)
```

Where 1 corresponds to water and 0 to air.

This expression is based on the meshes cartesian coordinate system. The origin of the meshes coordinate systems is located at $523.82\ m\ a.s.l.$ The interface at the inlet is placed at

$$z_w = z_b + h = (524.20\ m\ a.s.l. - 523.82\ m\ a.s.l.) + 1.0168 = \quad (5.9)$$

$$= 1.3968\ m$$

The result of the expression for α can be observed in Figure 16.

The BC for α are set as follows:

Boundary patch	**Boundary Condition**	**Definition**
inlet	`pos().z < 1.3968 ? 1 : 0`	water below z = 1.23 m
outlet	`zeroGradient`	applies a zero-gradient on patch
atmosphere	`uniform 0`	value 0 throughout the entire patch
Left	`slip`	provides a slip constraint
right	`slip`	provides a slip constraint
riverSlope_riverSlope	`uniform 1`	value 1 throughout the entire patch

(Table 1: Boundary conditions for alpha1.) [17]

The inlet patch is set with `funkySetBoudary` to $1.3968\ m$ water height to match the interfaces location inside the mesh. The outlet is provided with a zero-gradient to match the water surface flowing towards it. The atmosphere patch has a constant value of 0 (for air), since it is constantly dry and the slope being a constant value of 1 (for water). The left and right patch are provided with a slip condition to model just one strip anywhere within the channel.

U

To match a discharge of $8.003\ m^3/s$ the velocity at the inlet patch needs to be set to $(0.9839, 0, 0)^T$. As an IC the water within the internal field can be set to the same velocity. The expression for the velocity field within the mesh set in `funkySetFields` is:

```
alpha1 == 1 ? vector(0.9839,0,0) : vector(0,0,0)
```

It sets all water excluding the boundaries to the velocity $(0.9839, 0\ , 0)^T$. The expression for the inlet patch set in `funkySetBoudary` is also

```
alpha1 == 1 ? vector(0.9839,0,0) : vector(0,0,0)
```

It sets all the incoming water to the above calculated velocity. The result can be observed in Figure 16. The remaining boundary patches are defined with the following BCs:

Boundary patch	**Boundary condition**	**Definition**
outlet	`zeroGradient`	applies a zero-gradient on patch
atmosphere	`uniform 0`	value 0 throughout the entire patch
Left	`slip`	provides a slip constraint
right	`slip`	provides a slip constraint
riverSlope_riverSlope	`uniform 0`	value 0 throughout the entire patch

(Table 2: Boundary condition for U.) [17]

Again, to not be a subject of influence, the outlet patch is provided with a zero-gradient condition. Likewise to alpha1 the left and right patch are provided with a slip condition. The velocity at the slope needs to be 0 to create a boundary layer and the atmosphere patch is also set to 0.

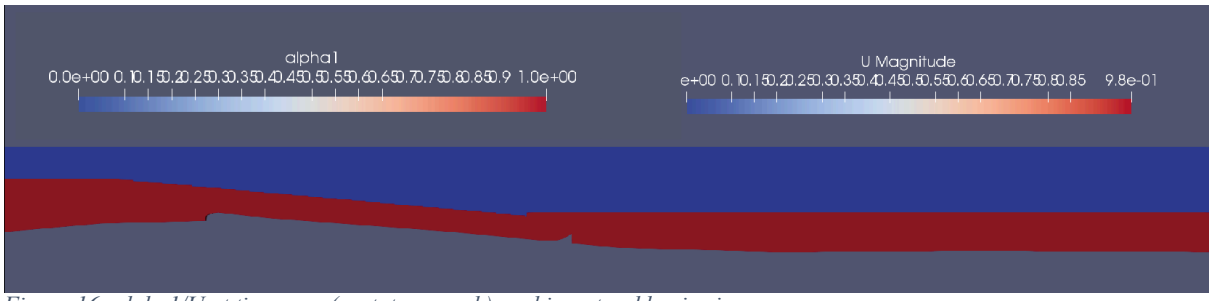

Figure 16: alpha1/U at time zero (prototype mesh); red is water, blue is air.

pd

This file contains the information about the total pressure calculated as follows:

$$pd = p - \rho g(z_w - z_{cell}) \tag{5.10}$$

All the boundaries, except for the atmosphere patch, are defined with the condition `buoyantPressure`. For this boundary condition an initial atmospheric pressure gradient needs to be set, out of which `interFoam` sets a fixedGradient pressure [19]. All values and gradients, including the internalField can be set to 0. They adjust themselves during the simulation. The atmosphere is set to `totalPressure`. It provides a total pressure condition based on the properties of air [17].

3.2.3 ControlDict

As any other OpenFOAM solver, `interFoam` commences every run by building a database. The controlDict, located in *system/controlDict/*, contains all essential database controls such as input/output and time settings, these are essential, as during computing the output data is typically given out at time intervals. The controlDict can be observed in Figure 17. All crucial keyword entries will be explained below [18].

The `startFrom` entry defines when the resulting simulation is supposed to start. In this case it starts at `startTime`, which is specified in the row below as 0. Analogical to `startFrom` the end of the simulation is determined at `stopAt` and `endTime`, which is set to 20 s. `deltaT` is the time interval in which `interFoam` will process the iterations. `writeInterval` sets the time steps (in seconds) in which a directory with the results for the given time is created. Here it is set to 0.5,

which means every $0.5\ s$ the simulation data at the time of simulation is saved in a new directory. That data describes the state of the flow at the time equal to the directories name. The chronological succession of all the flow states show the flows movement. If `writeControl` is set to `adjustableTimeStep`, it will adjust the time interval during the run, so that the Courant number (Co) stays at its defined maximum ($maxCo\ =\ 0.5$) [18].

The Co basically describes how many times a fluid particle will be monitored during its passage through one cell. The equation for Co is:

$$Co = \frac{\delta t |\boldsymbol{U}|}{\delta x} \tag{5.11}$$

Whereas:

$\boldsymbol{U}$: velocity magnitude

δt: time step

δx: cell length in direction of $\boldsymbol{U}$

controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     3.1                                |
|   \\  /    A nd           | Web:         http://www.extend-project.de       |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         20;

deltaT          0.002;

writeControl    adjustableRunTime;

writeInterval   0.5;

purgeWrite      0;

writeFormat     ascii;

writePrecision  7;

writeCompression uncompressed;

timeFormat      general;

timePrecision   7;

runTimeModifiable yes;

adjustTimeStep  on;

maxCo           0.5;

maxAlphaCo      0.1;

maxDeltaT       1;
```

Figure 17: controlDict file for the 3D-simulation.

Since velocity and cell size vary within the mesh, in each time interval every cell has its own Co. Therefore a mean and maximum Co can be established and are displayed in the output data in the terminal during the run. That the maximum Co is set 0.5 means that inside the smallest cell with the fastest velocity a fluid particle is being monitored twice. The mean Co might be remarkably lower. To ensure an accurate simulation the Co should be kept below the value of 1. That ensures that any cell within a streamline captures every fluid particle flowing through at least once.

The `adjustableTimeStep` setting keeps the simulation on a certain quality in case the timestep `deltaT` is too big and reduces the computation time when `deltaT` is too low [18].

3.2.4 Numerics

The *fvSchemes* and *fvSolution* determine the calculations and discretisation and therefore, influence the convergence of `interFoam`. Although the settings in the template files found in the `damBreak` tutorial can work well for some simulations, the awareness of their execution is highly important. `interFoams` performance may differ significantly by changing the discretisation schemes and/or solver settings.

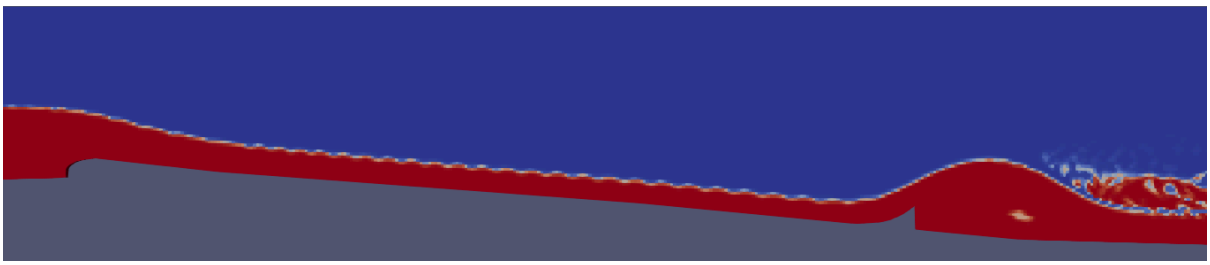

Figure 18: Simulation run with template schemes. Produces wiggles at the interface.

The series of physical terms necessary to calculate a flow are allocated to numerical schemes to solve those. The fvSchemes file, located in *system/fvSchemes/*, contains all numerical schemes that are used in a simulation [20].

The simulation result shown in Figure 18 was generated with the numerical schemes given with the template files from the tutorial. The overall result looks fine except for the wiggles at the interface in the section of supercritical flow. These would influence later calculations. This chapter explains the crucial schemes used to achieve useful results. They are listed in Table 3.

Even though OpenFOAM offers a few time schemes the one used here is the first-order forward implicit Euler scheme and is specified with the `Euler` entry.

$$v_{\kappa+1} = v_{\kappa} + \tau * f(t_{\kappa+1}, v_{\kappa+1}) \tag{5.12}$$

Whereas:

v_{κ}: solution for the time κ

τ: length of one timestep κ

κ: count of the timestep

and

$$f(t_\kappa, v_\kappa) = \dot{v}_\kappa \tag{5.13}$$

group	term	applied scheme
timeScheme (ddt) $\frac{\partial}{\partial t}; \frac{\partial^2}{\partial t^2}$	default	Euler
gradSchemes ∇	default	Gauss linear
	grad(U)	Gauss linear
	grad(alpha1)	Gauss linear
divSchemes $\nabla \cdot$	div(rho*phi,U)	Gauss linear
	div(phi,alpha)	Gauss upwind
	div(phirb,alpha)	Gauss interfaceCompression
	div(phi,k)	Gauss upwind
	div(phi,omega)	Gauss upwind
laplacianSchemes ∇^2	default	Gauss linear corrected
interpolationSchemes (cell to face interpolation value)	default	linear
	interpolate(HbyA)	linear
snGradSchemes (component of gradient normal to a face)	default	corrected

(Table 3: Numerical schemes.) [30]

The keyword `Gauss` denotes to the standard finite volume discretisation of Gaussian integration, which is the integration over the faces and is based on summing the face values [20].

$$\int_{V_{cell}} \nabla \phi \; dV = \oint_{\partial V_{cell}} d\boldsymbol{S} \; \phi = \sum_{face} \boldsymbol{S}_{face} \; \phi_{face} \tag{5.14}$$

[31]

The face values need to be interpolated from the cell centre values. The interpolation scheme is specified by the keyword following the `Gauss` keyword. There is are many interpolation schemes to choose from. Some are even specially designed for the divergence terms [20].

The face values for the gradient schemes of this simulation are all being linearly interpolated. It is specified with the keyword `linear` [20].

The first attempt to fix the wiggles is the change of the interpolation scheme for the momentum flux, `div(rho*phi,U)`, from a linear to limited scheme classified as a TVD (Total Variation Diminishing) scheme. The `phi` keyword refers to face flux. The keyword for the limited scheme is `limitedLinearV 1`. It interpolates linearly but limits the gradient to a certain degree. The scheme itself is specified with the entry `limitedLinear` and limits with the following equation:

$$\Gamma = \max[\min\left(\frac{2r}{k}, 1\right), 0] \tag{5.14}$$

with

$$r = \frac{\phi_{j+1} - \phi_j}{\phi_j - \phi_{j-1}} \tag{5.15}$$

Whereas:

Γ: solution of issue

r: ratio of successive gradients

k: user input

The “V” at the end is entered when the scheme is applied on a vector field. The scheme then takes into account the direction of the steepest gradient of ϕ. When calculating the face momentum flux r represents the velocity. The user input ϕ is specified at the very end of the entry and can obtain values between 0 and 1. In this case $k = 1$. The solution of issue is the face momentum flux [21].

The result is identical to the previous one. As the interpolation was limited and the results don’t differ, it allows the assumption that the scheme for the momentum flux is not the reason for the wiggles.

The interpolations for the face mass flux, `div(phi,alpha)`, were originally solved with the Van Leer scheme, which is also a TDV scheme that promises smooth variations and no abrupt switches. It is specified with the `vanLeer01` keyword.

$$\Gamma = \frac{r+|r|}{1+r} \tag{5.16}$$

Where r is calculated as in equation 5.15 [32]. The `01` at the end of the keyword limits the result strictly between 0 and 1 [20].

However, to attain better results it was replaced by the first order upwind differencing scheme specified to with the entry `upwind`. It has the great advantage of always achieving physical results, but it produces a diffusive solution [26].

Firstly the upwind scheme compares the negative flux with the positive flux to find the flow direction. Then it projects the value in the cell centre P on the face centre f downstream the of P [33]. Figure 19 illustrates the projection.

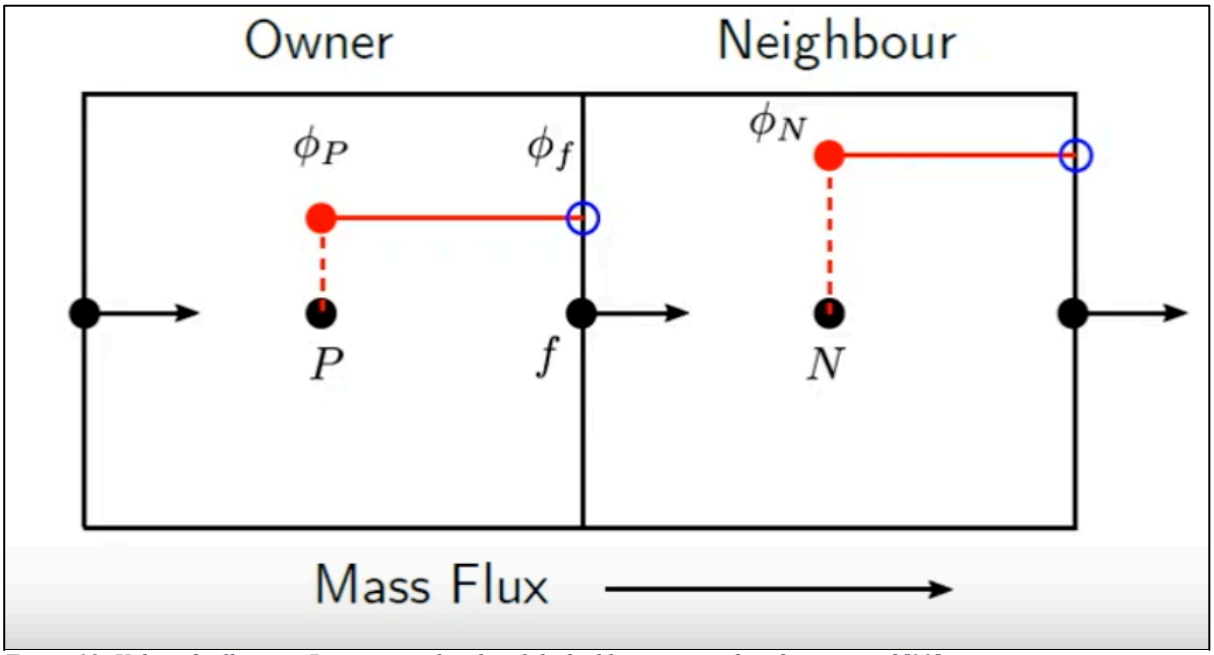

Figure 19: Value of cell centre P represented with red dashed line projected on face centre f [33].

The result is shown Figure 20. It is satisfying as all the wiggles have disappeared. However, comparing it to the previous result the interface has diffused and become much broader.

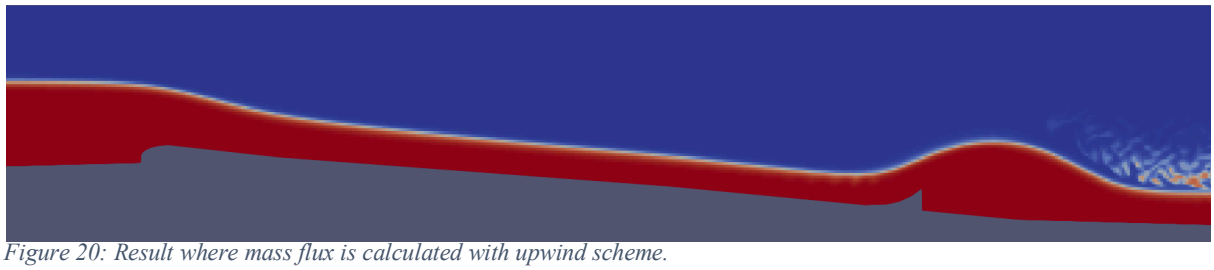

Figure 20: Result where mass flux is calculated with upwind scheme.

The equation is extracted of the interfaceCompression.H header file and the Phi.H header file from the OpenFOAM library.

The solution and algorithm control is determined in *fvSolution*, located in *system/fvSolution*. It controls the solvers, its tolerances and the algorithms. In the solvers section all linear solvers used for discretised equations i.e. U, pd, omega, are listed together with its tolerance, relative tolerance and option for preconditioning of matrices in the conjugate gradient solver [22].

A rather small correction is done by lowering the tolerances for the pd and pdFinal solver from $10^{-7}$to 10^{-9}. It sharpens the interface a little better at the beginning of the wave (Figure 21a and 21b).

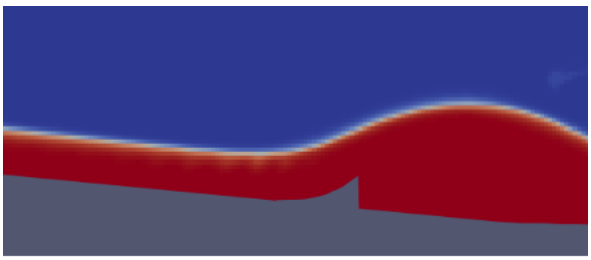

Figure 21a: Simulation calculated with higher tolerance.

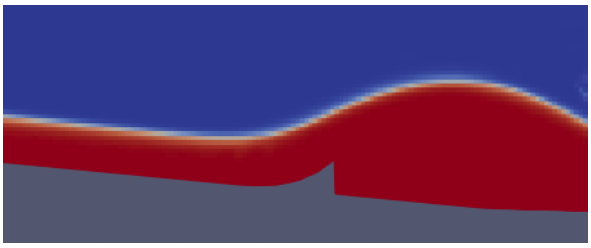

Figure 21b: Simulation calculated with lower tolerance

Apart from the solver section the file contains the settings for the two solving algorithms that `interFoam` uses: PISO and PIMPLE. PIMPLE solves with outer and inner iterations. Within every outer iteration the inner iterations are done. Once convergence for the timestep is reached or the maximum number of iterations is reached the algorithm moves on to the next time step. This simulation uses one outer iteration and a maximum of four inner iterations [35]. The PISO algorithm was originally developed as an extension of the SIMPLE algorithm used by the solver `simpleFoam`. It is responsible for solving the pressure-velocity calculation for the Navier-Stokes equation [34]. The `cAlpha` option in the PISO controls determines the interface compression. Setting it to 0 results in no compression and 1 resembles conservative compression. Everything larger conveys to increased compression [36].

Due to upwind differencing of the mass flux, the interface is too broad. Therefore, `cAlpha` is set to 4. This results in a sharper interface that can be observed in Figure 22, but also creates

Figure 22: Simulation with compressed interface (cAlpha = 4). Instabilities in subcritical flow.

instabilities. The simulation is now unstable in the sections of subcritical flow, whilst before it was unstable in the section of supercritical flow. Although this problem wouldn't be as significant, it is not worth sacrificing stability. Therefore, the simulation will remain with conservative compression.

3.2.5 Turbulence

Since the turbulence model used for a simulation can influence the results drastically, the choice of the right turbulence model is highly important. The Floßlände is a very fortunate case, because it exists already, which means that by trying different turbulence models and comparing the results with the actual wave of the physical model in Figure 23c one can find out which model to use for further proceeding.

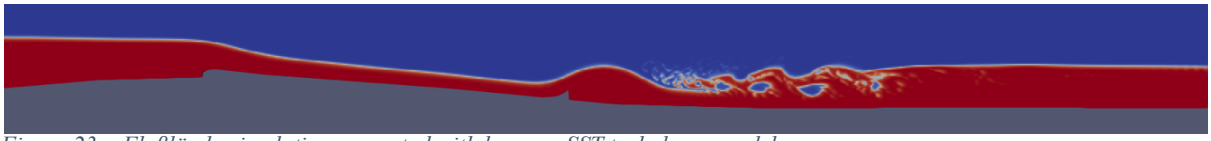

Figure 23a: Floßlände-simulation computed with k-omegaSST turbulence model.

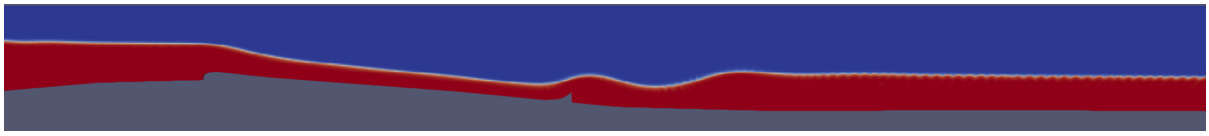

Figure 23b: Floßlände-simulation computed with k-epsilon turbulence model.

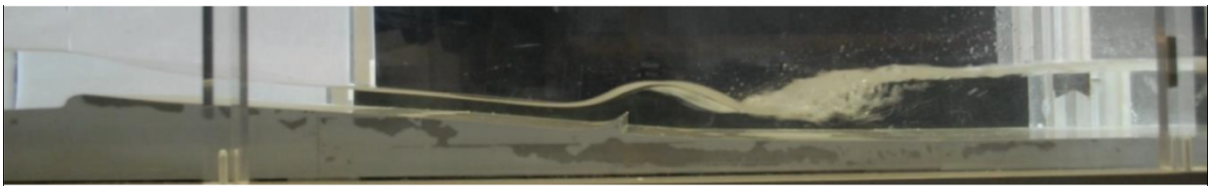

Figure 23c: Physical model of the Floßlände in ratio 1:18 [7].

In this study two turbulence models are tested: k-omegaSST and k-epsilon. They can be observed in Figure 23a and 23b. Both of them are RANS Two-Equation Models. The k-epsilon model solves for the variables k (turbulent kinetic energy) and epsilon (rate of dissipation of kinetic energy). It is no low-Reynolds model which means that it calculates velocities in the viscous sublayer, close to the wall, analytically applying wall functions. Unlike low-Reynolds models, wall functions do not apply a no-slip condition at the walls surface, even though that corresponds to physical reality. However, in this simulation the boundary layer is not monitored and can be approximated with wall function to reduce computational effort [23].

The k-omegaSST (shear stress transport) derives from the k-omega model, both can be used as low-Reynolds models. K-omegaSST has similarities with the k-epsilon in the freestream section and is more alike the k-omega towards the boundary layer. As explained above the

boundary layer is not monitored which is the reason why wall functions will be used for this model, too [23].

As shown in Figure 17a and 17b the two turbulence models provide very different results. The wave that is formed using k-omegaSST is bigger than the one using k-epsilon. The k-epsilon simulation also doesn't produce a hydraulic jump as the k-omegaSST simulation does. However, both are compared to the result of a physical experiment run with a discharge of $Q = 8.003\ \frac{m^3}{s}$. As observed the shape and size of the wave in the simulation done with k-omegaSST is more alike the experiment and therefore, will be used for further analysis.

3.2.6 Mesh Convergence

During the stage of mesh-convergence the overall mesh will be refined by levels in order to analyse the accuracy of the simulation. Every cell is split in half by all spatial dimension per refinement level. That means that in a true three dimensional mesh every cell is split into $2^3 = 8$ parts per refinement level. In a pseudo 3D mesh, as the one developed in chapter 3.1.2, one of the three dimensions is neglected. Therefore, every cell is only split into $2^2 = 4$ parts per level of refinement. This means that the relative amount of cells compared to a true 3D mesh, decreases with increasing number of refinement level. This again shows the imperative advantage of using a pseudo 3D mesh.

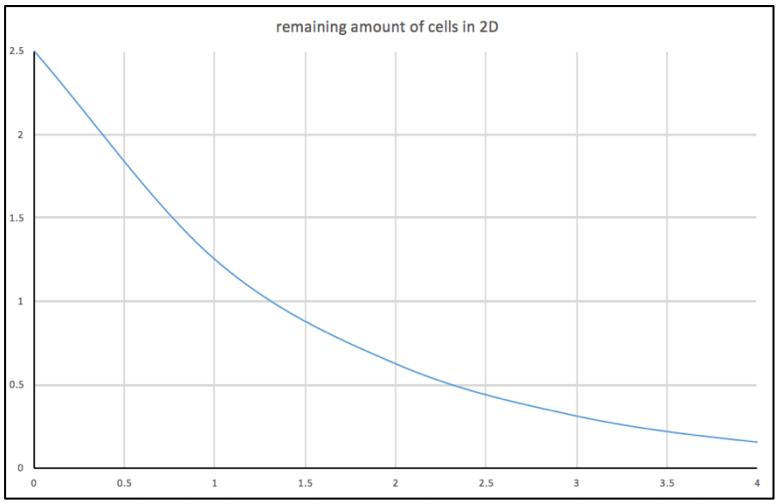

Figure 24: Chart displaying cell saving with increasing refinement level.

The chart in Figure 24 shows the percentage of cells in one strip (pseudo 3D mesh) of the channel in a block mesh (where 100% is the amount of cells in a block with the dimensions: 24.1 m * 2 m * 8 m). Refinement level 0 is the mesh that has been developed in chapter 3.1.2 (1200 cells). As a true 3D mesh it would have 48000 cells. Therefore, the cells amount has been reduced to

$$\frac{1200}{48000} = 0.025 = 2.5\% \tag{5.16}$$

One must take in count that this calculation only applies on the block mesh. After running `snappyHexMesh` some cells are further refined which distorts the calculation on which the diagram above is based on. Nevertheless, it still works as an indicator for computation-time saving.

The goal of mesh convergence is to see if the simulations results converge towards one value. A key parameter needs to be chosen and monitored at one specific location of the mesh. The result of the chosen parameter at the selected location will be recorded and plotted against a measure of the mesh density. The arising curve, consisting of at least three runs of convergence, clarifies when full convergence is accomplished. Full convergence occurs when two results on different mesh density measures are identical. If full convergence is not accomplished yet, it indicates where the result is converging to and how far away the finest mesh is from full convergence [24].

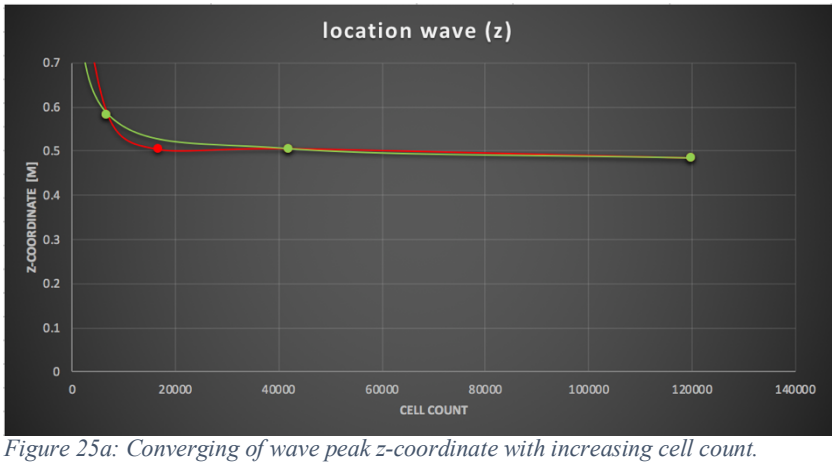

Figure 25a: Converging of wave peak z-coordinate with increasing cell count.

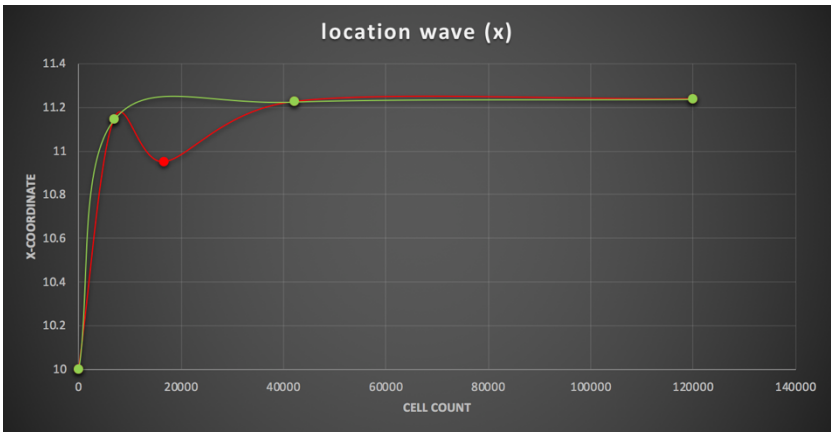

Figure 25b: Converging of wave peak x-coordinate with increasing cell count.

The monitored parameter is the location of the wave peak. The location is defined by its x- and z-coordinates. These are both plotted against the cell count, which is the used measure of mesh density. The charts can be observed in Figure 25a and 25b.
Four runs of mesh convergence are done, which means that the prototype mesh is refined three times (refinement level 3) to create the finest mesh. Its cell count is 119 952. Although some irregularities occur at low cell count (red line), both, the x- and z-coordinates converge well. Skipping refinement level 1 (green line) a much smoother curve arises. The green line consists of 3 simulation runs and is therefore, still a valid convergence. Hence the simulation at refinement level 3 can be assumed to be accurate enough for the use of the wave analysis.

To keep the condition

$$Co < 1 \tag{5.17}$$

duration of timesteps needs to decrease with increasing amounts of cells. Therefore, the computational effort raises due to both, higher mesh density and shorter timestep. At refinement level 4 the computation time becomes enormously long which is why it is not done. Figure 26 shows the results of the different stages of mesh convergence.

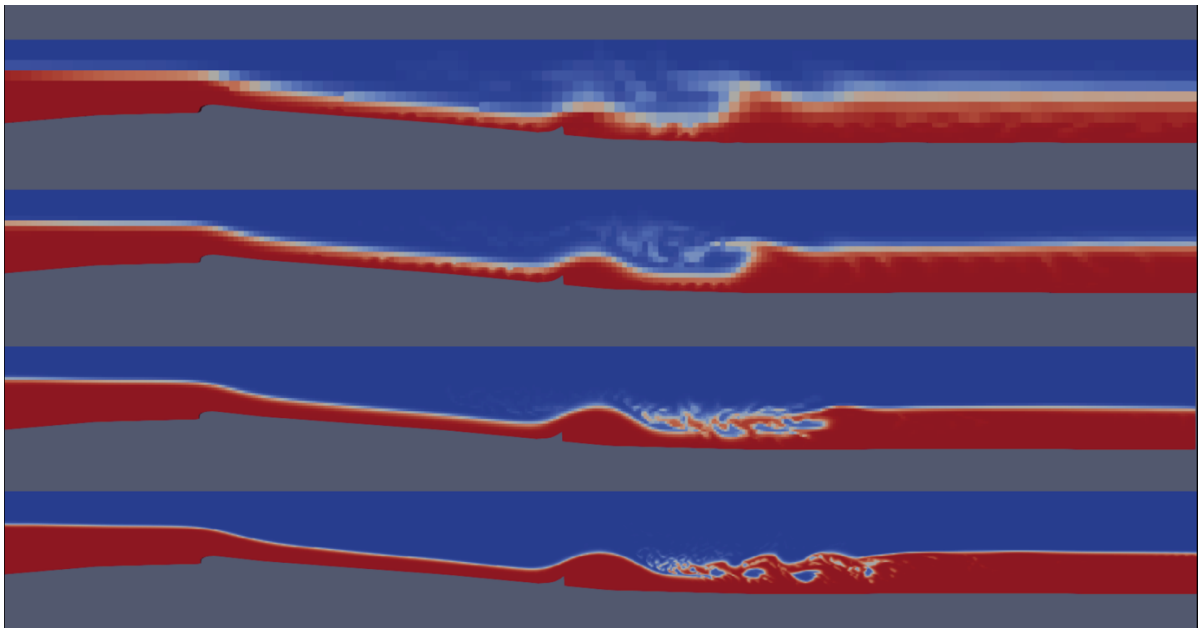

Figure 26: result of simulation with increasing level of refinement. Top to bottom: Level 0, level 1, level 2, level 3. $Q = 8.003\ \frac{m^3}{s}$.

3.3 2D – solver shallowFoam

The 2D simulation is done with the solver `shallowFoam`. Although the 2D simulation fails and does not produce useful data for the steady wave, the solver will be introduced. It will also be explained why it is not suitable for the phenomenon of issue.

`shallowFoam` is a newly developed solver. Its Authors are KM-Turbulenz GmbH (2009) and Florian Mintgen (2012) from the Technical University of Munich [29]. Since `shallowFoam` is a 2D solver it works with very little amount of computation time compared to a 3D solver. Therefore, it is recommendable to choose a 2D solver in cases where it is possible to use it.

3.3.1 Shallow Water Equations

The assumption of shallowness is $h << L$, whereas L is the horizontal length of the flow in x-direction. The vertical dimension is neglected and therefore, there is only one block in z-direction [25].

The mass continuity equation is downsized to only two dimension: it is assumed that the vertical acceleration is equal to zero: $\frac{\partial w}{\partial z} = 0$.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \qquad \Rightarrow \qquad \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{7.1}$$

As a consequence the vertical component of the momentum equation is reduced to the hydrostatic pressure:

$$\frac{\partial p}{\partial z} = \rho g \tag{7.2}$$

The depth integration of Equation (7.2) results in the hydrostatic pressure distribution and is one of the basic assumptions of shallow flows:

$$p(z) = \rho g(z_w - z) + p_a \tag{7.3}$$

[25]

For building the 2D model the mass conservation equation and the horizontal components of the momentum equation need to be depth-averaged. Likewise to the Reynolds decomposition a quantity is divided into a depth average $\bar{\phi}$ and a deviatoric $\tilde{\phi}$ component:

$$\phi = \bar{\phi} + \tilde{\phi} \tag{7.4}$$

The depth average is defined as

$$\bar{\phi} = \frac{1}{h}\int_{z_b}^{z_w} \bar{\phi}\, dz \tag{7.5}$$

Depth-averaging the transport equation yields

$$\frac{\partial h}{\partial t} + \frac{\partial q_i}{\partial x_i} = 0 \quad (i = 1,2) \tag{7.6}$$

and the depth-averaged momentum equation results in

$$\frac{\partial q_i}{\partial t} + \overline{u_j}\frac{\partial q_i}{\partial x_j} = -\frac{g}{2}\frac{\partial h^2}{\partial x_i} - gh\frac{\partial z_b}{\partial x_i} - \frac{\tau_{bi}}{\rho} + \frac{1}{\rho}\frac{\partial h\overline{\tau_{ij}}}{\partial x_j} \qquad (i = 1,2) \tag{7.7}$$

[25]

3.3.2 Boundary Conditions (BC) and Initial Conditions (IC)

The BC and IC that are specified for the 2D solver are displayed in the following Table.

BC	file name	inlet	outlet	atmosphere/ bottomWall/ right / left
specific discharge q $[\frac{m^2}{s}]$	HU	`flowRateInlet`	`zeroGradient`	`empty`
location of water surface z_w $[m]$	H	`criticalFlowDepthInlet`	`zeroGradient`	`empty`
Strickler value k_{st} $[\frac{m^{\frac{1}{3}}}{s}]$	kst	`zeroGradient`	`zeroGradient`	`empty`
turbulent viscosity ν' $[\frac{m^2}{s}]$	nut	`zeroGradient`	`zeroGradient`	`empty`
bottom elevation z_b $[m]$	S	`zeroGradient`	`zeroGradient`	`empty`

(Table 5: Boundary conditions for the 2D simulation.)

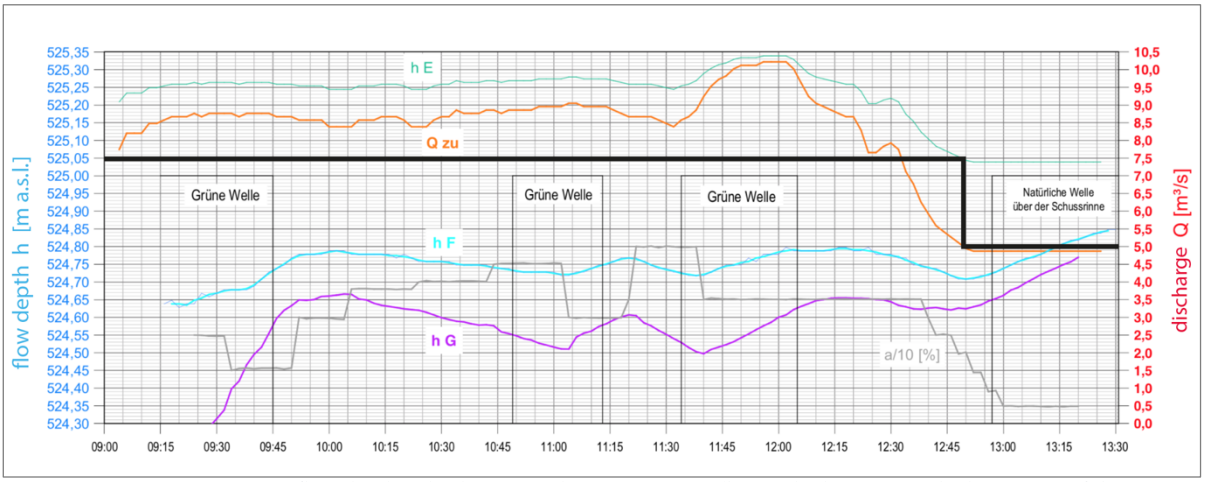

Figure 27: Diagram picturing flow depth – discharge – relation over time (h E is the location at the beginning of the Floßlände channel) [6] (translated from German); Line showing the flow depth at sea level for $5\frac{m^3}{s}$.

A prototype 2D simulation is done with a discharge of 5 $\frac{m^3}{s}$. The specific discharge q is simply calculated as

$$q = \frac{5\frac{m^3}{s}}{8\,m} = 0.625\,\frac{m^2}{s} \tag{7.8}$$

The location of the water surface by the inlet is extracted of the chart made by Dipl. Ing. Johann Obert, Planungsbüro für Wasserbau [6] as follows:

$$z_w = 525.05\ m.a.s.l. - 523.82\ m.a.s.l. = 1.23\ m \tag{7.8}$$

The chart can be observed in Figure 27. This mesh, too, has its base at $523.82\ m.a.s.l.$ For the BC assigned to the inlet water surface (`criticalFlowDepthInlet`) the critical flow depth h_c needs to be specified, too:

$$h_c = \sqrt[3]{\frac{q^2}{g}} = \sqrt[3]{\frac{0.625^2}{9.81}} = 0.34\ m \tag{7.9}$$

The initial water surface within the mesh is set with `funkySetFields`. The expression used sets a similar IC as in 3D and defined with the following code:

```
(pos().x <= 4.1) ? (1.23) : ((pos().x > 4.1) && …

(pos().x < 11) ? (1.23 - (pos().x - 4.1) * 0.09) : ((pos().x >= 11) ? (0.65)))
```

The turbulent viscosity is set to be $0.001\ \frac{m^2}{s}$ all over the mesh. The Strickler value is assumed to be $85\,\frac{m^{\frac{1}{3}}}{s}$ for the wooden planks of which the biggest part of the channel consists. The bottom of inlet area consists partly of gravel and partly of concrete. For the gravel a Strickler value of $40\,\frac{m^{\frac{1}{3}}}{s}$ is assumed and for the concrete $65\,\frac{m^{\frac{1}{3}}}{s}$. The bottom elevation requires a list of the height of the bottom at every cell. The values are linearly interpolated for the individual cells.

4. Results

4.1 3D - results

For the Analysis of the wave the 3D simulation previously introduced is repeated five times on different discharges. The obtained data is then compared to data from Ruth Morandis Bachelor Thesis. To be able to compare the data, the discharges for which the steady wave is simulated need to match the ones measured by Ruth Morandi.

4.1.1 Data extraction

After running a simulation one obtains time folders i.e. 0.5, 1, 1.5, 2, … which are saved in the main project folder. As mentioned in chapter 3.2.3. it depends on the `writeInterval` setting in the *system/controlDict* in which time intervals the results are written down. Each of those folders contain files for every parameter that has been set in the *0/* folder. Each file within a time folder holds one value of its parameter for every cell in the mesh. E.g. the apha1 file inside the time folder *11.5/* contains the α-values for every cell at the time $t = 11.5s$. Since the mesh contains 119 952 cells the apha1 file holds the same amount of values.

The essential measure for the comparison to the physical model is the flow depth $h(x)$, which is not directly displayed in the results. To obtain the flow depth one first has to calculate the z-coordinate $z_w(x)$ of the water surface. The bottom height $z_b(x)$ will then be subtracted from $z_w(x)$. The z-coordinate and the bottom height have its base at 523.82 m.a.s.l. The result is the flow depth. Figure 28 exemplifies the calculation.

The following equation is given to attain $z_w(x)$:

$$z_w(x) = \tag{6.1}$$

$$= M - \sum_{i=1}^{n} (1 - \alpha_i(x)) * dz$$

Whereas:

$M = 2$: mesh height

n: number of cells in a column

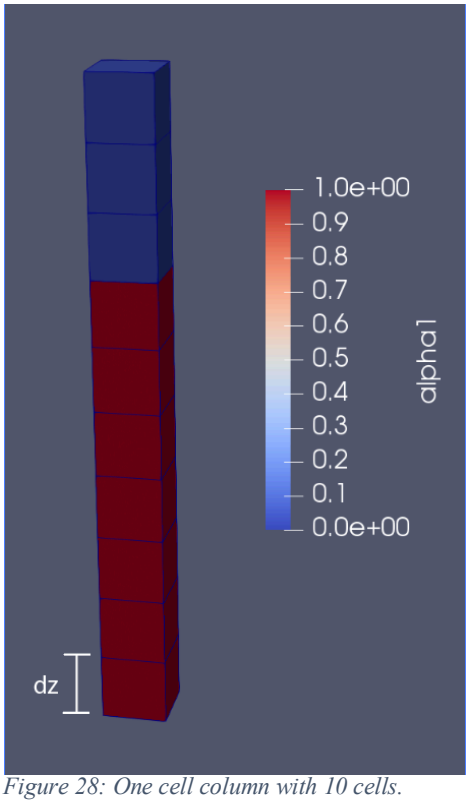

Figure 28: One cell column with 10 cells. Red: $\alpha = 1$. Blue: $\alpha = 0$.

dz: height of one cell ($= 0.025$)

$x \coloneqq [0, 24.1]$: x-coordinate in the mesh

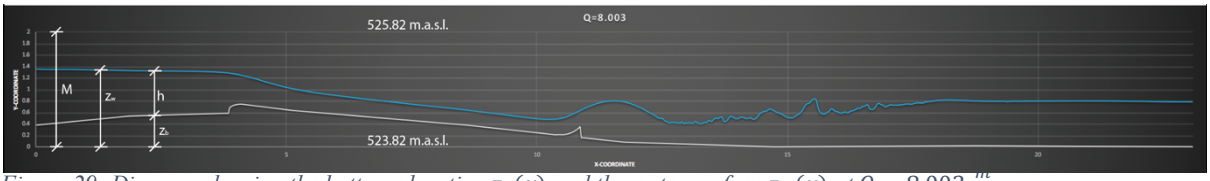

Figure 29: Diagram showing the bottom elevation $z_b(x)$ and the water surface $z_w(x)$ at $Q = 8.003\ \frac{m^3}{s}$.

The result is shown in Figure 29.

For the calculation of $z_w(x)$ it is better to sum the air-cells. The air-cells in the top part of the mesh have all the same size $dx * dy * dz$, whilst the water-cells in the bottom part of the mesh do vary in size due to `snappyHexMesh` as explained in chapter 3.1.2.

Once the $z_w(x)$ has been calculated the flow depth simply results in:

$$h(x) = z_w(x) - z_b(x) \tag{6.2}$$

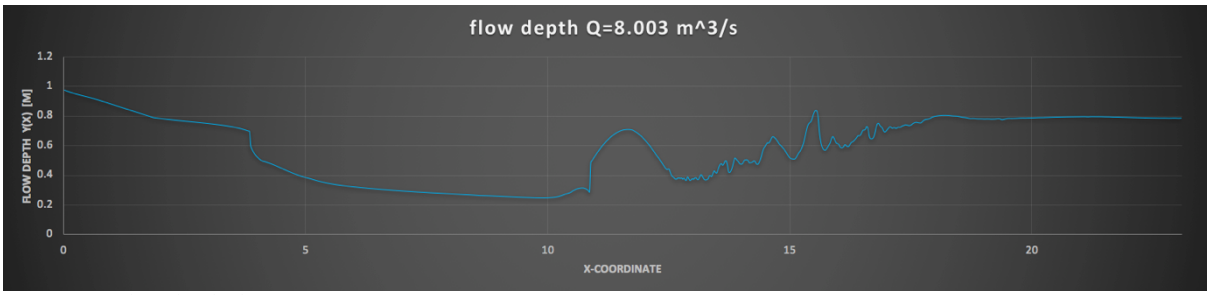

Figure 30: Flow depth chart.

The slope height is linearly interpolated from the coordinates given in chapter 3.1.1 to match the x-coordinate of each column. The flow depth is shown in the chart in Figure 30.

Since the data in the alpha1 file of a time folder is listed in a fashion inconvenient for this work, the data must be rearranged to columns as explained above. OpenFOAM provides a sampling tool called `sample`. All needed inputs are specified in *system/sampleDict/*. The interpolation scheme used is set to `cell`, which means that the sampling value is equal to the value in the centre of a cell. Therefore, the value is considered as constant within a cell [27]. The field sampled for this case is α. The columns sampled correspond to the category of 1D lines. The results are displayed in the folder *sets/* [28]. Every single column must be written down in *system/sampleDict/* under the subdirectory `sets`, including its start and end. The start of each

column is defined to be in the middle of each bottom cell and the end to be in the middle of each top cell of the mesh. The amount of values extracted per column is set in `nPoints` to 80. Figure 31 shows the notation to define the columns.

```
sets
(
line1
{
type uniform;
axis distance;
start (
.0125
1.05
0
);
end (
.0125
1.05
2
);
nPoints 80;
}

line2
{
type uniform;
axis distance;
start (
.0375
1.05
0
```

Figure 31: Columns specified in from sampleDict.

The results in *set/11.5/* for the time $t = 11.5s$ holds one file for every column. Every of those files contain 80 α-values. One value for every cell inside the column. The values are sorted from bottom to top.

To compute the values of the flow depth and other parameters in the following chapter a *.sh script is written with the programming language Octave. All scripts can be found in the attachment.

Now that the data is extracted and sorted the flow depth can be calculated as explained above.

4.1.2 Comparison to Physical Model

The physical model provides data for the inlet flow depth, the lowest point of the wave, the peak of the wave, the outlet flow depth and the discharge. To match the different discharges the inlet velocity for each simulation run needs to be calculated as follows:

$$u_{inlet} = \frac{Q}{h_{inlet} * b} \tag{6.3}$$

Q	4.879	6.958	8.003	9.501	10.131
$\mathbf{y_{inlet}}$	0.844	0.9664	1.0168	1.096	1.1212
$\mathbf{u_{inlet}}$	**0.7260**	**0.9000**	**0.9839**	**1.0836**	**1.1295**

(Table 4: Calculation for inlet velocity.)

Hence, the values for the discharge Q and the inlet flow depth y_{inlet} of the simulation are identical to the values measured in the laboratory.

The data of the simulations and the physical model are being compared in the diagrams in Figure 32a, 32b and 32c. The lowest point of the wave doesn't differ much between the CFD results and the results from the physical model. Both trend lines have a similar gradient. The measurements from both experiments mostly match the trend line or are very close to it.

Unlike to measurements of the lowest point, the data for the peak of the wave have great differences. This happens because the steady wave reacts very sensitive towards the smallest changes of water level [7]. The peak is the most unstable part of the entire wave. Therefore, it is very natural to observe fluctuations. However, the results of the CFD simulation still follow a linear rising trend line. The measurements do not deviate far. The physical model on the contrary, is more sensitive. It becomes so imprecise that the flow depth trend is declining. It needs to be considered that this chart only contains five measurements. Ruth Morandi has measured about 60 times. All her data put together does

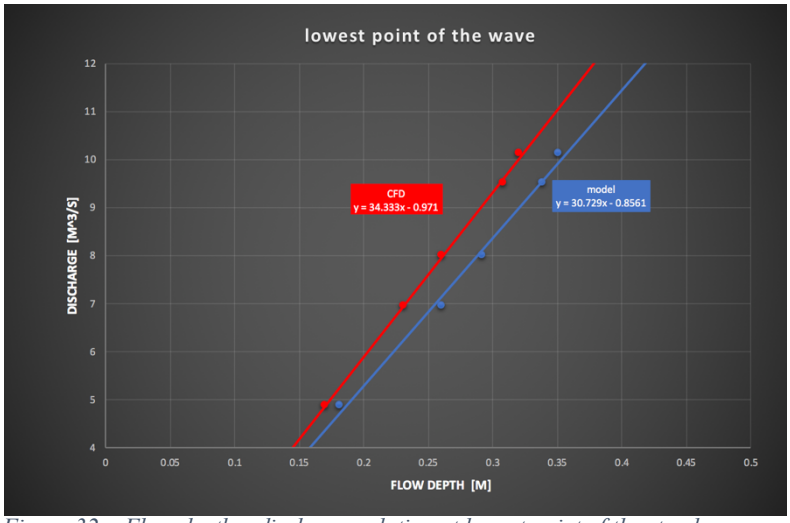

Figure 32a: Flow depth – discharge relation at lowest point of the steady wave. Blue: Physical model. Red CFD simulation.

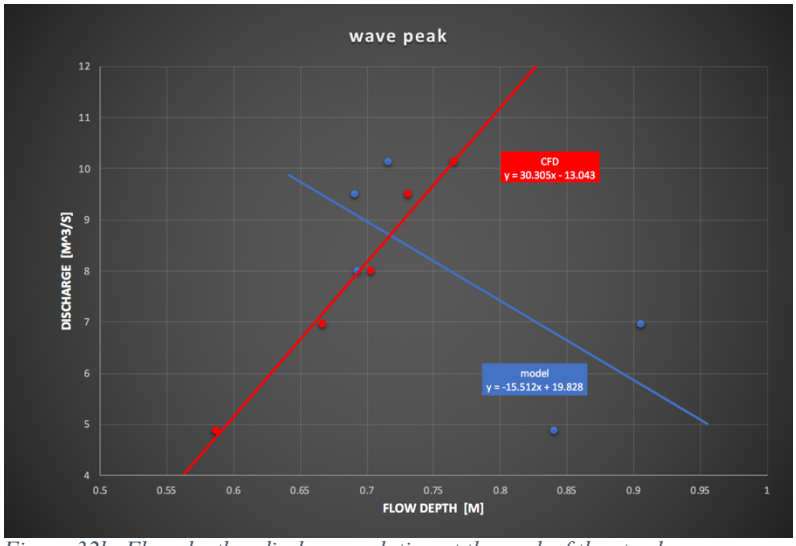

Figure 32b: Flow depth – discharge relation at the peak of the steady wave.. Blue: Physical model. Red CFD simulation.

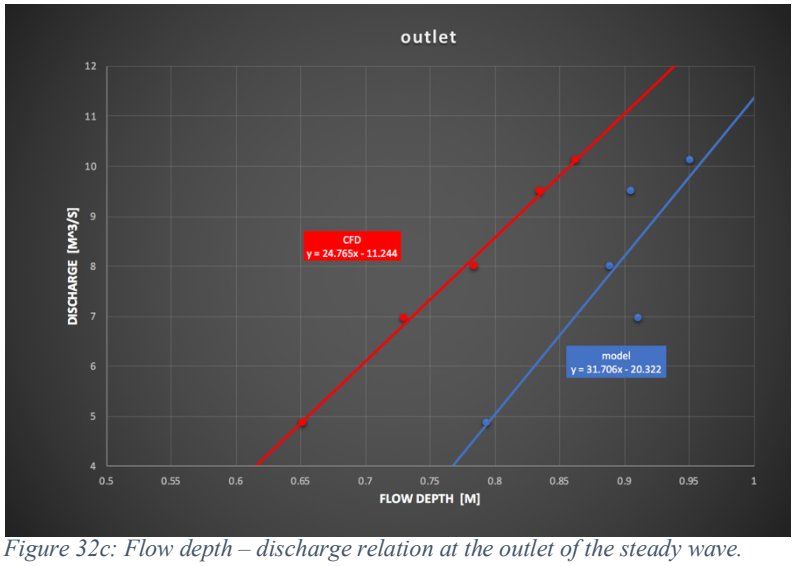

Figure 32c: Flow depth – discharge relation at the outlet of the steady wave. Blue: Physical model. Red CFD simulation.

form a rising trend function [7]. Nevertheless the flow depths on the higher discharges share similarities.

The outlet results are more accurate again. Both trends are linear rising and the gradients are similar, too. The flow depth, however, between the CFD and the experiment differs considerably. Again, they become more similar with rising discharge. In any case the outlet flow depth succumbs the turbulences caused by the hydraulic jump, which can cause significant fluctuations. Therefore, it is natural for these measurements to be inaccurate to a certain degree.

4.1.3 Total Energy Line

The Total Energy Line for the Floßlände is calculated with the Bernoulli equation for open-channel-flow:

$$H(x) = z_b(x) + h(x) + \frac{u^2(x)}{2g} \tag{6.4}$$

The velocity u needs to be depth averaged. In 1D calculations the velocity in x-direction is obtained with

$$u_x = \frac{Q}{h(x)*b} \tag{6.5}$$

But the mesh used is a 2D mesh as the y-dimension has been neglected. That means that the vertical velocity needs to be taken in account. To do so the velocity data needs to be extracted in a similar fashion as the flow depth has been extracted in chapter 4.1.1: again, every column contains 80 sampling points. To eliminate the air velocity all cell velocities within a column are multiplicated with alpha1. Then the 2D magnitude of every cell velocity is calculated. Lastly the velocities are depth averaged.

$$U(x) = \frac{\sum_{i=1}^{n} \alpha_i * \sqrt{{u_{xi}}^2(x) + {u_{zi}}^2(x)}}{h(x)} * dz \tag{6.6}$$

The result of the can be observed in the Figure 33.

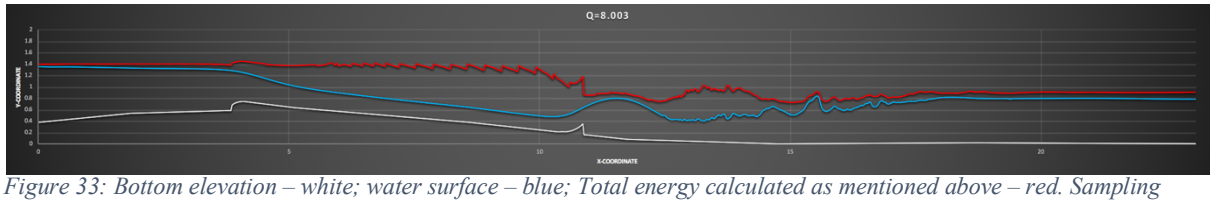

Figure 33: Bottom elevation – white; water surface – blue; Total energy calculated as mentioned above – red. Sampling density of 80 Points Per Column.

One would agree that the result does not meet the requirements to be assumed as realistic. To find the error in this calculation a second attempt of calculating the total energy is done with a sampling density of 320 sampling points per column.

The result in Figure 34 proofs that the irregularities in the section of supercritical flow are due to the sampling density, as they disappear with a higher sampling resolution.

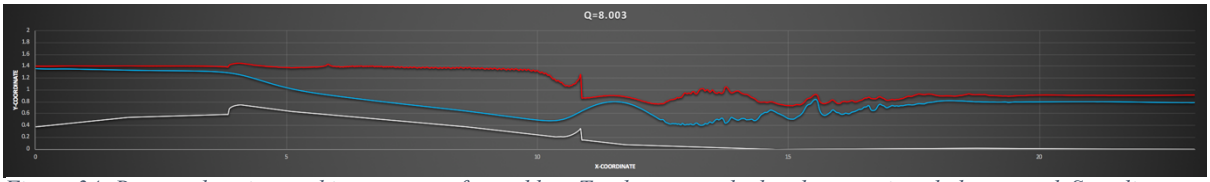

Figure 34: Bottom elevation – white; water surface – blue; Total energy calculated as mentioned above – red. Sampling density of 320 Points Per Column.

When extracting the gauge pressure values at the interface, where they are supposed to be equal to zero, it is noticeable that this is not always the case. Along the length of the channel the pressure does have some fluctuations in positive direction. This can be observed un Figure 35. Some locations of the fluctuations match the locations of the disrealities of the Total Energy Line. Therefore, it can be assumed that this simulation fails to calculate all relevant parameter needed to entirely analyse the channel flow. The reason for the failure could be the mesh resolution, the numerics or maybe `interFoam` itself, but it yet remains unknown.

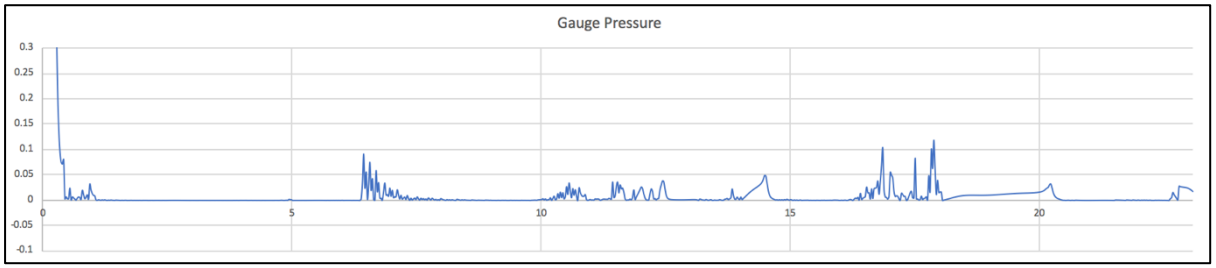

Figure 35: Gauge pressure plotted against the length of the channel.

Based on the failed calculation an assumption of the Total Energy Line can be done. It is displayed in Figure 36. The values are averaged and the friction losses neglected. The losses at the sill and the stationary lip are calculated based on the averaged energy levels before and after the subject of issue. Both happen to be

$$\Delta e = 0.05 * u \tag{6.7}$$

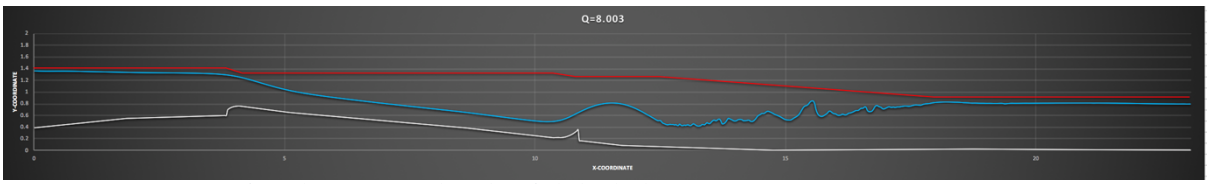

Figure 36: Assumption of Total Energy Line based on failed calculations.

4.2 2D – results

In Figure 37 and 38 the result of the 2D simulation can be observed. The results become clearer in when looking at the chart in Figure 39. The chart shows the bottom elevation and the water surface line plotted against the length of the channel. `shallowFoam`s output data only provides the flow depth h. The water surface location can easily be calculated as

$$z_w = z_b + h \tag{7.10}$$

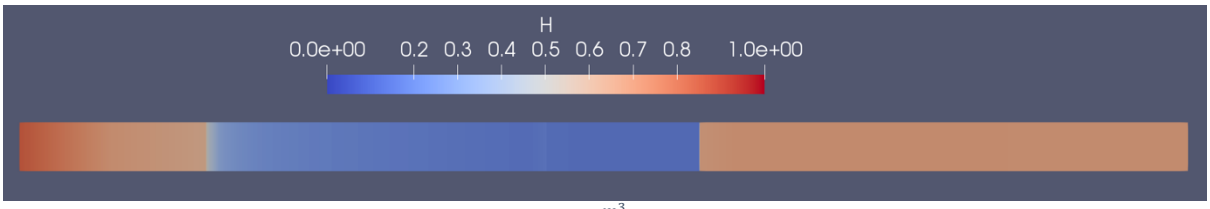

Figure 37: Flow depth h result of the 2D simulation at $Q = 5\ \frac{m^3}{s}$.

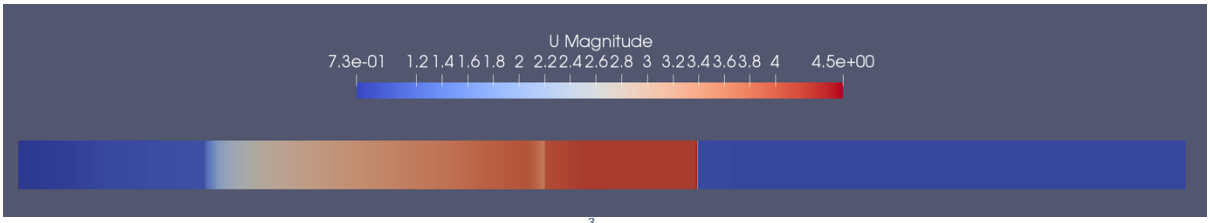

Figure 38: Velocity u result of the 2D simulation at $Q = 5\ \frac{m^3}{s}$.

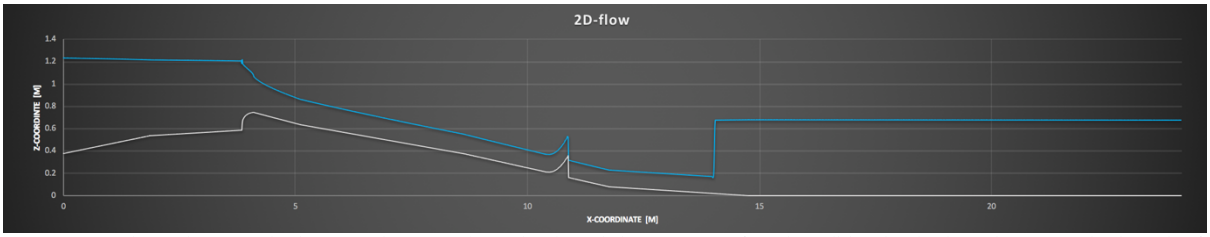

Figure 39: Chart showing the bottom elevation (grey line) and the water surface line (blue line) plotted against the length of the channel.

The flow is undoubtedly unnatural and cannot be assumed to be realistic. The instant drop of the water level by the sill, the missing wave just after the stationary lip and the instant change from supercritical to subcritical flow are indicators that the simulation failed.

The two main simplifications on which shallow water equations are based on (the neglection of the vertical dimension, which results in hydrostatic pressure distribution and the depth-averaging of the horizontal velocities) result in significant limitations. Since the vertical component of the velocity is missing, shallow water equations can appear to have trouble simulating flows that include vertical velocities such as waves [25]. As shown in Figure 40 the streamlines do have a gradient in many areas of the flow. The areas with the strongest gradients are the sill area and the wave area. Therefore, it is comprehensibly that the solver produces unnatural results in these areas. The instant change from supercritical to subcritical flow is

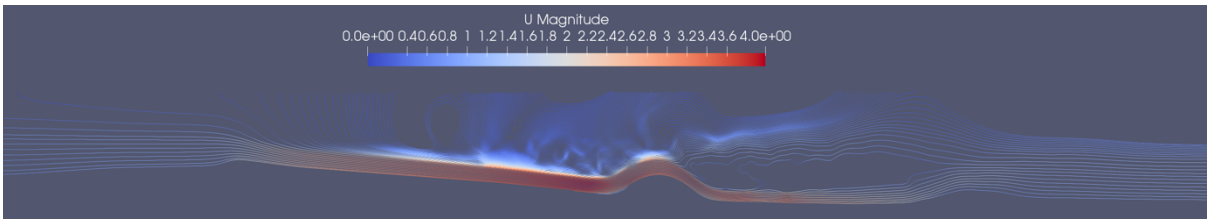

Figure 40: Streamline visualization of the 3D simulation.

assumed to be due to the depth averaging of the horizontal velocities.
However, `shallowFoam` does perform well in the inlet section, in the outlet section and in the section of supercritical flow. When comparing the flow depth of the 3D with the 2D simulation (Figure 41) one can observe the flow depth in the supercritical section to be identical, although they were expected to vary due to the high magnitude of the vertical velocity resulting of the very fast velocities of supercritical flow. Meanwhile the flow depth in the inlet and outlet section do vary a little bit, although they were expected to be more similar than the results in the supercritical section.

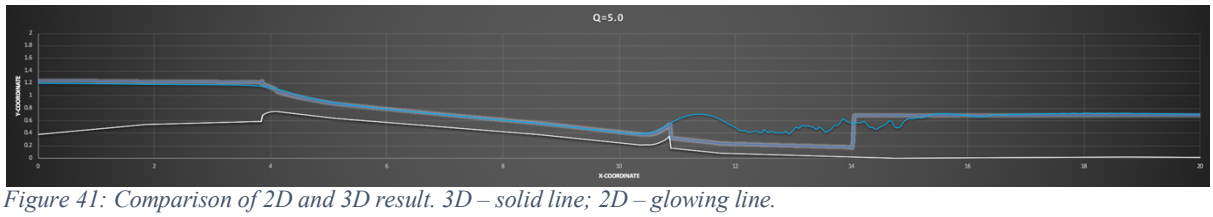

Figure 41: Comparison of 2D and 3D result. 3D – solid line; 2D – glowing line.

5. Conclusion

The tools that are provided by the OpenFOAM library to use on `interFoam` allow the 3D solver to create a simulation that replicates a real wave well. Nevertheless this simulation includes a change from subcritical to supercritical flow which creates a challenge for the developer to find the right settings especially regarding the numerical settings. However, it is very concerning that the challenge of calculating the Total Energy was not completed successful. The question of why the equation couldn't be resolved to produce physical results is still an outstanding issue. It is concerning because it remains unknown whether it is `interFoam`s performance or the mesh's properties that cause the problem.

The performance of `shalloFoam` is definitely incapable of producing any physical results on simulating a steady wave. However, it is possible to make use of the 2D solvers efficiency. When simulating a steady wave such as the one analysed in this work it is necessary to include the flow before and the after the wave itself. Perhaps these flows are shallow and non-vertical depending on the circumstances of the wave to be simulated. Then it would be possible to represent these sections in a 2D manner.

It would be highly suggested to make use of the solver `shallowInterFoam` developed by Florian Mintgen for his doctoral dissertation. He developed a method to couple a 2D mesh to a 3D mesh. It allows the user to run the 2D solver on the 2D mesh and the 3D solver on the 3D mesh. This method saves the user copious amounts of computation time [25].

Bibliography

[1] Igsm.info. (2019). *Floßlände – IGSM*. [online] Available at: https://www.igsm.info/flosslaende/ [Accessed 8 Apr. 2019].

[2] Eisbachwelle.de. (2019). *eisbachwelle.de - EISBACH MÜNCHEN RIVER SURFING – alles über Surfer & Fluss Surfen an der Eisbach Welle München Deutschland – river wave surfing Munich Germany*. [online] Available at: http://www.eisbachwelle.de/2013/flosslaende-anfaenger-welle-surfen-surfbar-zeiten/ [Accessed 8 Apr. 2019].

[3] Deventer, D. (2011). *Riversurfing von München bis zum Amazonas. Munich*. 1st ed. Munich: Terra Magica, p.37.

[4] Wortwellenreiter.de. (2019). *Isar Wellenreiter | Surfspots München | Wort Wellenreiter*. [online] Available at: http://wortwellenreiter.de [Accessed 8 Apr. 2019].

[3] Deventer, D. (2011). *Riversurfing von München bis zum Amazonas. Munich*. 1st ed. Munich: Terra Magica, p.53.

[6] Planungsbüro für Wasserbau. (2015). *Versuch zur Verbesserung der Surfwelle am 1.7.2015 Zwischenbericht*. Ländkanal, Lände, Maria-Einsiedel-Bach Maßnahmen 2015. Munich: Landeshauptstadt München Baureferat HA Ingenieurbau - J 32, pp. Annex 1, Annex 2.

[7] Morandi, R. (2018). *Maßstäbliche Untersuchung zur stationären Welle an der Floßlände*. Bachelor. Technical University of Munich.

[8] Openfoamwiki.net. (2019). *BlockMesh - OpenFOAMWiki*. [online] Available at: https://openfoamwiki.net/index.php/BlockMesh [Accessed 8 Apr. 2019].

[9] Openfoamwiki.net. (2019). *SnappyHexMesh - OpenFOAMWiki*. [online] Available at: https://openfoamwiki.net/index.php/SnappyHexMesh [Accessed 8 Apr. 2019].

[10] CFD Direct. (2019). *OpenFOAM v6 User Guide: 5.4 Meshing with snappyHexMesh*. [online] Available at: https://cfd.direct/openfoam/user-guide/v6-snappyHexMesh/ [Accessed 8 Apr. 2019].

[11] Deshpande, S. S., Anumolu, L., & Trujillo, M. F. (2012, November 9). *Evaluating the performance of the two-phase flow solver interFoam* (Tech.). Madison, USA: Department of Mechanical Engineering, University of Wisconsin.

[12] Openfoamwiki.net. (2019). *InterFoam - OpenFOAMWiki*. [online] Available at: https://openfoamwiki.net/index.php/InterFoam [Accessed 8 Apr. 2019].

[13] Physik.cosmos-indirekt.de. (2019). *Volume-of-Fluid-Methode – Physik-Schule*. [online] Available at: https://physik.cosmos-indirekt.de/Physik-Schule/Volume-of-Fluid-Methode [Accessed 8 Apr. 2019].

[14] Openfoamwiki.net. (2019). *Contrib/swak4Foam*. [online] Available at: https://openfoamwiki.net/index.php/Contrib/swak4Foam [Accessed 8 Apr. 2019].

[15] OpenCFD. (2019). OpenFOAM® - Official home of The Open Source Computational Fluid Dynamics (CFD) Toolbox. [online] Available at: https://www.openfoam.com/ [Accessed 8 Apr. 2019].

[16] Openfoamwiki.net. (2015). Contrib/funkySetFields. [online] Available at: https://openfoamwiki.net/index.php/Contrib_funkySetFields#Expression_syntax [Accessed 8 Apr. 2019].

[17] OpenCFD. (2018). Standard boundary conditions. [online] Available at: https://www.openfoam.com/documentation/user-guide/standard-boundaryconditions.php. [Accessed 8 Apr. 2019].

[18] Greenshields, C. J., & CFD Direct Ltd. (2015, December 13). *OpenFOAM The Open Source CFD Toolbox User Guide* 3.0.1 ed. OpenFOAM Foundation, pp.18-23.

[19] Funature, A. (2013, September 23). About buoyant Pressure in OpenFoam. [online] Available at: http://blog.funature.net/en/2013/09/24/about-buoyantpressure-in-openfoam/. [Accessed 8 Apr. 2019].

[20] CFD Direct. (2018, July 10). OpenFOAM v6 User Guide: 4.4 Numerical schemes.. [online] Available at: https://cfd.direct/openfoam/user-guide/v6-fvschemes/. [Accessed 8 Apr. 2019].

[21] Larsen, B. E., Fuhrman, D. R., & Roenby, J. (2018, April 3). *Performance of interFoam on the simulation of progressive waves*. Lyngby, Denmark: Technical University of Denmark, Department of Mechanical Engineering, Section of Fluid Mechanics, Coastal and Maritime Engineering.

[22] CFD Direct. (2018, July 10). OpenFOAM v6 User Guide: 4.5 Solution and algorithm control. [online] Available at: https://cfd.direct/openfoam/user-guide/v6-fvsolution/. [Accessed 8 Apr. 2019].

[23] Wasserman, S. (2016, November 22). Choosing the Right Turbulence Model for Your CFD Simulation. [online] Available at: https://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/13743/Choosing-the-Right-Turbulence-Model-for-Your-CFD-Simulation.aspx. [Accessed 8 Apr. 2019].

[24] NAFEMS. (2019). The Importance of Mesh Convergence. [online] Available at: https://www.nafems.org/join/directory/knowledge-base/the-importance-of-mesh-convergence-part-1/ [Accessed 9 Apr. 2019].

[25] Mintgen, G. F. (2017). *Coupling of Shallow and Non-Shallow Flow Solvers - An Open Source Framework*. Doctoral dissertation. Technical University of Munich.

[26] Holzmann, T. (2018, January 24). Gauss Upwind Scheme (1st Order). [online] Available at: http://voluntary.holzmann-cfd.de/numerical-schemes/standard-schemes/gauss-upwind. [Accessed 8 Apr. 2019].

[27] Openfoamwiki.net. (2012, September 12). Sample. [online] Available at: https://openfoamwiki.net/index.php/Sample. [Accessed 8 Apr. 2019].

[28] OpenCFD. (n.d.). Sampling data. [online] Available at: https://www.openfoam.com/documentation/user-guide/userse21.php. [Accessed 8 Apr. 2019].

[29] Mintgen, F. (2018, July 30). Mintgen/shallowFoam. [online] Available at: https://github.com/mintgen/shallowFoam. [Accessed 8 Apr. 2019].

[30] Dr. Pfefferer, J. (2018). *Numerik 2*. Lecture notes. Technical University of Munich.

[31] Jasak, H. (2015). Finite Volume Discretization in OpenFOAM. [online] Available at: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2015/HrvojeJasak/DiscretisationBestPractice.pdf. [Accessed 25 Apr. 2019].

[32] Tryggvason, G. (2017). Computational Fluid Dynamics. [online] Available at: https://www3.nd.edu/~gtryggva/CFD-Course2017/Lecture-9-2017.pdf. [Accessed 25 Apr. 2019].

[33] Fluid Mechanics 101. (2018, November 25). [CFD] What is the difference between Upwind, Linear Upwind and Central Differencing?. [online] Available at: https://www.youtube.com/watch?v=JVE0fNkc540&frags=pl%2Cwn. [Accessed 26 Apr. 2019].

[34] CFD online. (2015, May 15). PISO algorithm – Pressure Implicit with Split Operator. [online] Available at: https://www.cfd-online.com/Wiki/PISO_algorithm_-_Pressure_Implicit_with_Split_Operator. [Accessed 26 Apr. 2019].

[35] Lynch, D. (2018, January 9). CFD: PIMPLE algorithm. [online] Available at: https://www.simscale.com/forum/t/cfd-pimple-algorithm/81418. [Accessed 26 Apr. 2019].

[36] CFD Direct. (2018, July 10). OpenFOAM v6 User Guide: 2.3 Breaking of a dam. [online] Available at: https://cfd.direct/openfoam/user-guide/v6-dambreak/. [Accessed 29 Apr. 2019].