OpenFOAM Exercise 2

0. Initial advice

It is best to have done the first OpenFOAM exercise before attacking this one. For those who still want to do this exercise without having done the first, it is recommended to at least read through the first to get a few elementary understandings and tools in place.

The weir from the laboratory exercise is going to be simulated in this exercise, which requires a .stl file from the previous exercise. For those who have not done the previous exercise, a .stl file can be found on blackboard. Others may also want to compare files to ensure everything is ok. In this case, it is better to compare .txt-files than .stl-files, since they are easier to read. Viewing the stl-file in paraView with axes visible (as described in homework 9) should also be a good check of your stl-file. If you saved the stl-file given on blackboard with a different name than the one that you made yourself, you can compare them in paraView.

A general advice regarding running simulation cases in OpenFOAM is that you never create a case from scratch. It is considered better to copy a tutorial or another case that is similar to the case in question, and only make the changes necessary to the existing files. For this case there is a template on blackboard that can be copied and modified. Necessary changes will be described in the following.

To ease the work load on this exercise a certain amount of details is already filled out in this template. The idea is that you shouldn't have to repeat tasks similar to those performed in homework 9. And also to show you the possibility to dereference variables by `$myVariable` and to calculate variables by implementing expressions using inline commands `#calc “myEquation”`. Nevertheless, feel free to remove the contents that are already implemented if you want to learn more or better!

Purpose of the exercise

- Create a mesh using an STL-file and snappyHexMesh
  - The geometry of the STL-file should model the weir from the laboratory assignment
  - Define the domain for a background mesh that fits your STL-file
  - Run snappyHexMesh to create the desired geometry
  - Run the case using interFoam
- Compare results from CFD with experimental results obtained in the lab
  - Using the same post-processing utility in paraFoam as in homework 9

Commands for this case:
`paraFoam` - opens the case in paraView
`foamCleanPolyMesh` - deletes the mesh
`foamCleanTutorials` - deletes settings/calculations that OpenFOAM has done
`blockMesh` - generates background mesh
`snappyHexMesh -overwrite` - inserts weir mesh into background mesh
`extrudeMesh` - creates 2D case from 3D case (nicer cells towards the boundaries)
`checkMesh` – check the quality of the mesh
`interFoam` - computes water flow using only one core
`decomposePar + mpiexec -n x interFoam -parallel + reconstructPar`
-computes water flow in parallel, x is the number of cores.

1. Pre-processing

Download the case template from blackboard and place it in your run-folder.

Backgrond mesh

Meshing is done for the area in which fluid is flowing. This means a meshed weir is not needed, only a meshed area around the weir. In OpenFOAM, the procedure is to create a background mesh including both the area for the flow and the weir, and then insert within the background mesh the shape that the flow will encounter, in this case, the weir.

The .stl file is already made (in homework 9 or available at blackboard), and it is important to take its coordinates into consideration when creating the background mesh. The weir needs to fit into the background. The weir from the laboratory experiment was 37 cm high and 42 cm long. Your weir.stl file should have about these dimensions. If the top of the weir is at a given coordinate, there has to be enough space above it for the water to have a free surface within the background mesh. These measures should be the starting point for creation of your background mesh.

Enter the case folder

```
run
cd ofassignment2
```

If it does not already exist, create a folder named triSurface within the constant directory

```
mkdir constant/trisurface
```

Place your weir.stl file in this folder

```
cp pathToMyStlFile/weir.stl constant/triSurface
```

Open paraFoam to view your .stl-file

```
paraFoam &
```

Open your .stl file by File->Open->*myStlFile.stl* and press *Apply*. Show the axes by checking of the 'Edit Axes Grid' box. Pay attention to the orientation of the coordinate system. For the given weir, z is the vertical direction. The coordinates for your blockMesh, as well as the direction of gravity, should correspond to this. View your g-file in wordpad to check the direction of gravity.

```
write constant/g
```

The area needed above the weir will also depend on the given water discharge. In the laboratory exercise, different discharges were measured. To be able to compare the numerical solution to the one obtained in the laboratory, you have to use the same rate. If you do not have a discharge from the laboratory exercise, you can use a discharge of 25 L/s.

A starting point for the inlet height can be taken from your laboratory results, or estimated from the following equation:

$$Q = C * L * H^{3/2}$$

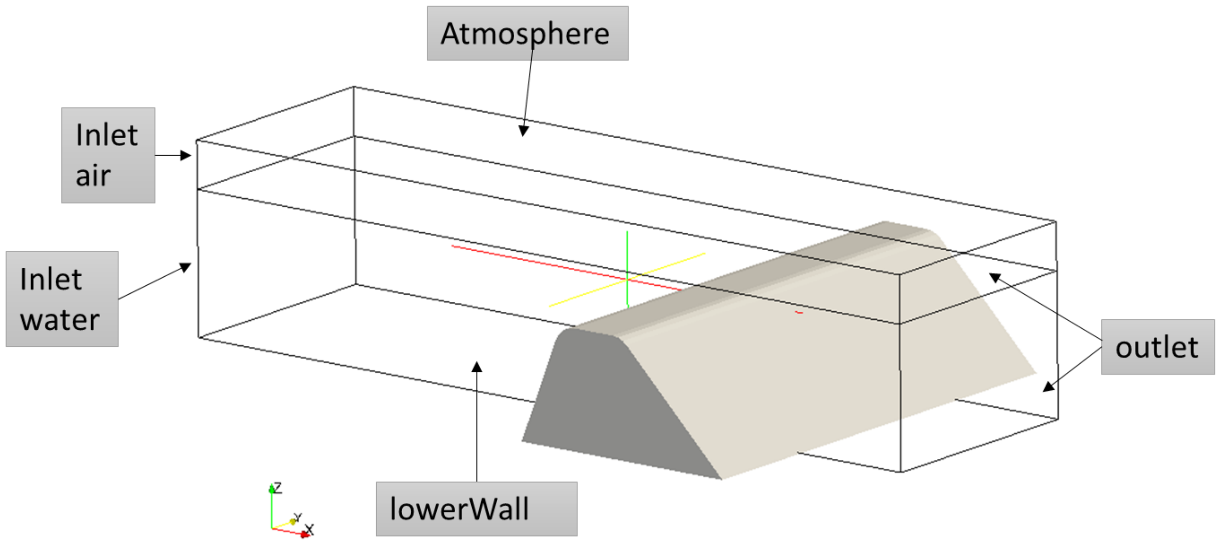
Remember that the *H* in this equation is the height above the top of your weir.

Open the U-file placed in the 0 directory.

```
write 0/U
```

This is the file where the boundary- and initial conditions are to given. Insert the volumetric flow rate given in m^3/s. The spots where you have to fill in the velocity are marked with some comments (/*some explanation*/).

Coordinates for the background mesh are given in blockMeshDict. As in homework 9, we use hexahedral boxes for the grid. Different from homework 9 we are now only going to make a simple rectangular grid. In this case it is practical to use two hexahedral blocks, one upon the other, see Figur 1.



Figur 1: This *figure shows how the two blocks and weir look. It also illustrates the placement of the different patches (boundary regions).*

Since you already spent some time on defining the blockMesh in homework 9, you don't have to spend time on this here. That means that the vertices, blocks, and patches are already defined in the template, based on some user defined input variables. Nevertheless, you have to supply the *blockMeshDict* file with these input variables.

The variables you have to supply are the grid dimensions, which means the simulated length of the channel, the width of the channel, the height of your domain, as well as the height of your water inlet. You also have to give in the cell size of your grid (more about this later).

The spots for the different input variables are described in lines that are commented out in the file. Open the *blockMeshDict* file and give in your measures

```
write system/blockMeshDict
```

Towards the top of the file will look something like this

```
// Defining variables - values given in assignment text
h_in ;  // heigth inlet [m]
b ;     // width of channel [m]

//choose grid dimensions
xstart ; // start of domain in x-direction [m]
xend ;    // end of domain in x-direction [m]
h_top ; // top heigth of domain [m]
```

Type your measures right in front of the semicolons, which means no space between the value and the semicolon.

The width of the channel is given (0.6 m). Make sure that the stl-file exceeds the block mesh in both ends. If not, extend the width of your stl-file.

The size of a block obviously depends on the coordinates of the vertices, and cell size depends on the number of cells in each direction. As we want to do a 2 dimensional analysis, we only use one layer off cells in the y-direction. As a starting point, you can use cell size of 2.5 cm. This is a parameter (one of several) that can be reduced if you experience convergence problems.

Insert this cell size as ‘dx’ in blockMeshDict

```
// Cell size given in assignment text
dx ; // cell size [m]
```

The number of cells in each direction is then calculated based on size of your domain and the desired cell size in the following lines.

```
// Calculate number of cells in each direction
length #calc "$xend - $xstart"; // length of domain [m]
dhlow $h_in; // heigth of lower block
dhtop #calc "$h_top - $h_in"; // heigth of upper block

nx #calc "std::ceil($length/$dx)";
ny 1;
nz1 #calc "std::ceil($dhlow/$dx)";
nz2 #calc "std::ceil($dhtop/$dx)";
```

In this file we define the input variables and refer to these variables by typing ‘`$myVar`’ in the following, instead of typing the same number several times. This makes it easier if we want to change anything (i.e. the domain size) later. Then we only have to change the variable at one place, instead of at every occasion the variable is used. Nevertheless, the resulting mesh would have been identical if you have used numbers as we did in homework 9.

Pay attention to the calculation possibility within OpenFOAM, you can do mathematical calculations within #calc “*myMathematicalExpression*”. Within the #calc-frame you also have the possibility to use c++ functions, as done when using the ceil function (https://en.wikipedia.org/wiki/Floor_and_ceiling_functions) for calculating the elements in each direction. This function is used here to get integers, which is required when giving the number of elements in each direction.

Then you are pretty much done with your *blockMeshDict* file.

Run the `blockMesh` command and look at the resulting mesh in paraView. View the mesh choosing ‘surface with edges’. Check that your patches, as well as their placements, corresponds to the ones given in Figur 1.

Meshing weir with snappyHexMesh

If everything looks as it should, you are ready to go to the next task. You are now going to adjust your grid using snappyHexMesh. Make sure that your weir.stl file is placed in the constant/triSurface directory

```
ls constant/triSurface
```

View your stl-file together with your background mesh to make sure that they fit each other. If everything looks ok, still standing in the case directory, execute

```
snappyHexMesh –overwrite
```

This will insert the weir into the background mesh. Then execute `extrudeMesh`, which makes the mesh 2D instead of the default 3D. Now execute `checkMesh`. If this command prints "Mesh OK" in the cygwin terminal, take a look in paraView for verification that everything is as it should. If it prints out "Failed x mesh checks", try to find out why, and correct the error. If you run `checkMesh` before you have executed `extrudeMesh`, `checkMesh` will print out "Failed x mesh checks".

Setting initial fields

The water and air are now set to come flowing in from the left, and there is no water any other place in the mesh. This means the upstream side of the weir has to be filled before water can flow over the crest. This steals unnecessary calculation time. Fortunately, it is easy to fill these cells with water before running a calculation, by defining the desired region in a dictionary called setFieldsDict, and then execute setFields. Open the dictionary, and make sure the coordinates given are reasonable, or change them if they are not. It may be helpful to view your backround mesh in paraView, with grid axes visible, to see whether the coordinates in setFieldsDict are reasonable.

Before rexecuting the setFields command, make sure that the 0 directory contains both files alpha.water and alpha.water.org. If not copy the one that are lacking from the other.

```
cp 0/alpha.water.org 0/alpha.water
```

or

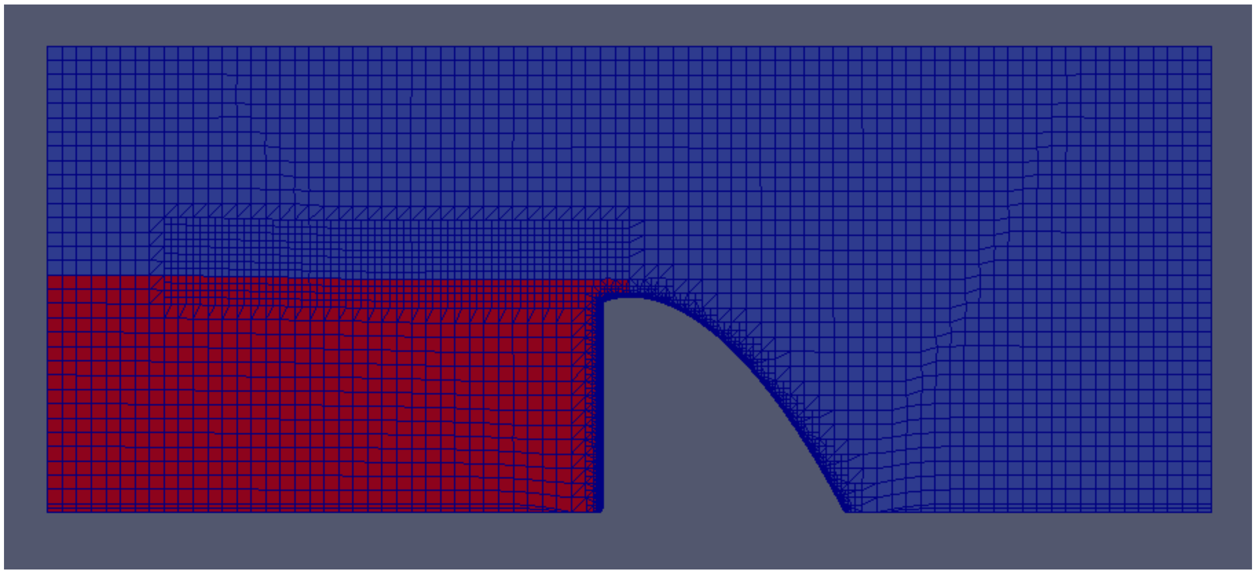
```
cp 0/alpha.water 0/alpha.water.org
```

It is the alpha.water-file that is included in the simulation. The setFields command will do changes to the 0/alpa.water-file, by giving all cells values of 0 or 1. This means that this file becomes mesh dependent. If you do changes to your mesh and then run the setFields command you will get an error message related to the inconsistent number of cells. In these cases you need a clean alpha.water file, which can be copied from the alpha.water.org file, executing the copy command:

```
cp 0/alpha.water.org 0/alpha.water
```

So if you ever find out that you want to run the case with another grid size, you have to copy back the original version of the alpha.water-file (alpha.water.org), before it will be possible to run the file again.

Then execute setFields. View the case in paraView by choosing alpha.water as the viewed variable. This should give you a view like this



It may be the case that the windows version of paraView got a problem by showing this for the initial time step, if so, just inspect the alpha.water-file. If there is some ones, and not only zeros in that file, probably everything is ok. A solution can also be to start the simulation and let it write at least one time step. Then you can view the alpha.water field of the first time step, which should be pretty similar to the initial situation.

2. Running the case

The case is now almost ready for simulation. The simulation will take some time, depending on the power of the computer and number of cores available. A computer with several cores provides the opportunity to split the case into as many parts as there are cores, and will calculate the case in a fraction of the time. See section 0 for commands and their variations according to the number of cores. If the user chooses to use several cores for the calculation, the *decomposeParDict* file, placed in the system folder, has to be adjusted. The file *decomposeParDict* requires the number of cores that the user wants to split the case into. This number is irrelevant if the user prefers to run the case on one core, as OpenFOAM will not read *decomposeParDict* unless the command decomposePar is actually executed.

Since this simulation may take some time, it is recommended to run interFoam as a background process, and save the output to a log-file

```
interFoam > log.interFoam &
```

It might be that your simulation do not converge at your first attempt. Then you can try to adjust the inlet height, or the length in front of the weir, reduce the time step (by adjusting the courant number maxCo, maxAlphaCo), reducing the cell size or changing the discretization scheme (in system/fvSchemes). Change different parameters one at the time. You can also go into the *snappyHexMeshDict*-file and adjust the refined area. For example, you can play around with the parameter `nCellsBetweenLevels` and `nSurfaceLayers` and the size of the surface refinement area defined in the surface and *surfaceAfter* dictionaries. Nevertheless, the values given in the template should be reasonable for running the case.

3. Post-processing

After simulation, open paraView and look at the motion of the water. Experiment with parameters, angles, colors and so on.

Use the method described in homework 9 to find the height of the surface on the upstream side of the weir at the last stadium of the calculation. Can you see from the surface elevation graph whether the simulation has converged? Calculate the discharge coefficient.

Compare it with the results obtained from the laboratory exercise. What is the deviation? Include an evaluation of this, and the surface elevation graph in your answer.