Project in the MSc/PhD course: CFD with OpenSource software Water Sprinkler using OpenFOAM 1.7.1

Author: Martin Hammas

Reviewer: Mattias Olander and Jelena Andric

November 2, 2010

Contents

1 Water Sprinkler

Modifications of the geometry of the damBreak case are made to resemble a water sprinkler. This is similar to the Master thesis done by Marianne Sjöstrand. The geometry is done setting up the vertices, blocks etc in `blockMeshDict` in two dimensions. A utility, `AlphaCalc.C` is implemented to analyze the water distribution in the watercollectors. A script then plots the water distribution as a histrogram, where each column corresponds to the amount of water in a single water collector. The script is written in Python. This tutorial is setup using OpenFOAM 1.7.1.

2 The watersprinkler case

2.1 The geometry

The geometry of the case is modified from the original `damBreak` case. The geometry has been modified in different ways, more blocks had to be made due to the twelve water collectors. This is because in `blockMeshDict` where the mesh is defined, the faces have to be merged in every direction, i.e. you cannot have two blocks facing each other (with the same geometric distance) with different number of cells. We start by copying the whole `damBreak` case to a new map by writing the following lines in the terminal window

```
run
mkdir sprinkler
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak sprinkler
cd sprinkler/damBreak
```

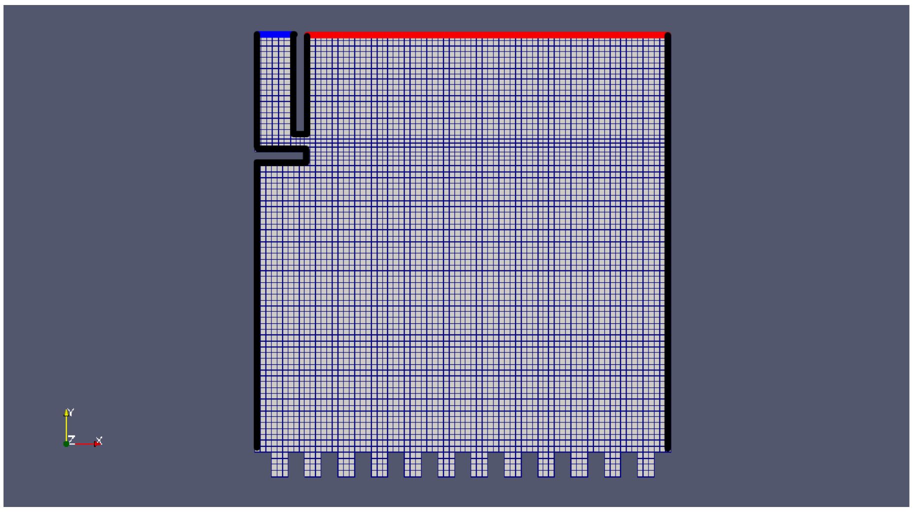
We are now in the case map `sprinkler/damBreak`. Start by replacing the `blockMeshDict`, it is located in the map `/constant/polyMesh` with the dictionary that was downloaded from the webpage
`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2010/`.

```
cp blockMeshDict constant/polyMesh
```

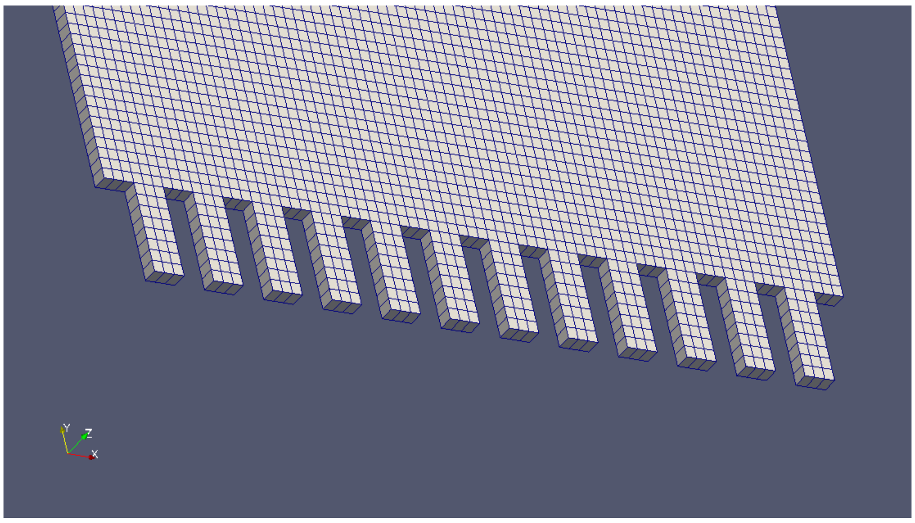
After replacement, write the following in the terminal window

```
blockMesh
checkMesh
```

This builds up the geometry from the file `blockMeshDict` using a mesh generator called `blockMesh`. The mesh now looks like



A zoomed figure showing the twelve water collectors



The blue patch in the upper left corner represents the inlet. In this case it is water with phase value 1. Everywhere else phase value 0. The red upper right patch is defined as "atmosphere". This is because since water is entering the system, something has to come out, in this case, fluid with phase 0. The black patches represent the patch wall.

2.2 Boundary conditions

The boundary conditions are set up in the `0/U` file. the boundary condition, `rampedFixedValue` has been chosen for the inlet. This is done because the main thing with this tutorial is to calculate the amount of water in the water collectors. This can not be done if the water rushes in through the entire simulation time. It is in other words chosen to simulate some kind of stationary flow. The inlet velocity quickly switches from $U_y = |0.5|$ to $U_y = 0$ on a steep ramp. The `U` file should look like

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField uniform (0 0 0);

boundaryField
{
        inlet
        {
                type            rampedFixedValue;
                refValueLow     uniform (0 -0.5 0);
                refValueHigh    uniform (0 0 0);
                startRamp       0.1;
                endRamp         0.15;
                value           uniform (0 0 0);
        }
        column
        {
                type  fixedValue;
                value  uniform (0 0 0);
        }
        leftWall
        {
                type  fixedValue;
                value  uniform (0 0 0);
        }
        rightWall
        {
                type  fixedValue;
                value  uniform (0 0 0);
        }
        lowerWall
        {
                type  fixedValue;
                value  uniform (0 0 0);
        }
        atmosphere
        {
                type  pressureInletOutletVelocity;
                value  uniform (0 0 0);
        }
        defaultFaces
        {
                type  empty;
        }
}
```

The column is the wall to the right of the inlet. A tutorial on how to download and compile `rampedFixedValue` with changes are discribed in `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2010/programmingTutorial.pdf` under the section ”Implementing a new boundary condition”. This tutorial works fine using the already existing boundary conditions, `fixedValue` or `timeVaryingUniformFixedValue`.

The `0/alpha1` has to be changed. Two new boundaries are prescribed, `inlet` and `column`. The inlet has the uniform value of 1. Value 1 indicates water in this case, and 0 elsewhere is air. The column patch is just a wall, and therefore it should be of type `zeroGradient`. The initial file should be

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }

    column
    {
        type            zeroGradient;
    }

    leftWall
    {
        type            zeroGradient;
    }

    rightWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }

    defaultFaces
    {
        type            empty;
    }
}

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

Finally some modifications should be done in `0/p_rgh`, which is the dictionary for the dynamic pressure.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{

    inlet
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    column
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    leftWall
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    rightWall
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    lowerWall
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        value           uniform 0;
    }

    defaultFaces
    {
        type            empty;
    }
}

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

Change the `endtime` in `/system/controlDict` to 2. Before running the case, the dynamic library that refers to the boundary condition `rampedFixedValue`, should be added on the last line in the file `system/controlDict`

```
libs ("libMyBCs.so");
```

Write the following in the terminal window

```
interFoam
```

In order to run the case.

2.3 The utility AlphaCalc.C

This utility can be downloaded from the webpage along with the other files

`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2010/`.

AlphaCalc needs to be added in the case directory. When standing in `tutorials/sprinkler/damBreak`, write

```
cp -r AlphaCalc tutorials/sprinkler/damBreak/AlphaCalc
cd AlphaCalc
```

The directory should consist of the following files

```
AlphaCalc.C Make
```

Inside the `Make` directory there should be two files

```
files options
```

Make sure that the `files` contains the following

```
AlphaCalc.C

EXE = $(FOAM_USER_APPBIN)/AlphaCalc
```

And the `options` file should contain

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = \
    -lfiniteVolume
```

If we take a look in the file `AlphaCalc.C`. The most interresting part are the loops that checks if the y-coordinate of each cell center is negative. If so, the x-coordinate is in a specific interval (that corresponds to a specific collector). Then it takes the previously loaded scalarfield alpha1 and multiplies it with the vectorfield V, where V are the volumes for each cell. It sums up this product for each cell that belongs to a specific collector. The different products represents the amount of water in each collector. Only the alpha values from the last timestep are loaded.

```
scalar waterLevel = 0;

forAll(centres,nIter)
{
        if(centres[nIter][1] < 0)
        {
                if(centres[nIter][0] < 0.4*0.146) //  first collector
                {
                        waterLevel = waterLevel + alpha1[nIter]*volumes[nIter];
                }
        }
}

Info << "waterLevel: " << waterLevel << endl;
```

This is the first loop out of twelve. There are twelve loops because there are twelve different water collectors in the system. The first collector has x-coordinates in the range [0.2 0.4]*0.146, and of course, all y-coordinates of the cell centers are negative. This sum then corresponds to the amount of water in collector number 1. To compile this C++ utility, write the following when standing in the directory `sprinkler`

```
wmake AlphaCalc
```

It then compiles the utility, so it can be used. Run the utility when standing in the case directory, and save the print as a log file with

```
AlphaCalc >& log
```

2.4 foamLog

To get all the twelve sums from `AlphaCalc.C` in a file that later on can be used write the following in the terminal window, while standing in the directory `sprinkler`

```
sudo gedit /<InstallationDirectoryOfOpenFOAM>/bin/foamLog.db
```

In this file, add the following line in the beginning

```
waterLevel/waterLevel: /waterLevel:
```

Comment all the others with `#`. Now, write the following while standing in the `sprinkler/damBreak` directory

```
foamLog log
```

This extracts the sums from the log-file created when we ran the `AlphaCalc.C` utility. To get rid of the "waterLevel:" in the file `logs/waterLevel_0` write

```
sed -e s/"waterLevel"/""/g logs/waterLevel_0 > waterLevel
```

It then creates a file, `sprinkler/damBreak/waterLevel` with 12 rows that corresponds to the sums of each water collector.

2.5 Python

For the following script we need to install PyFoam, with a few packages. To get Python, write the following text in the terminal window

```
cd $HOME/OpenFOAM
mkdir linuxSrc
cd linuxSrc
svn co https://openfoam-extend.svn.sourceforge.net/svnroot/ \
openfoam-extend/trunk/Breeder/other/scripting/PyFoam/

python setup.py install --prefix=$HOME/OpenFOAM
```

Afterwards, add the following lines in the end of your `bashrc` file, located in `etc/apps/paraview3/bashrc`

```
alias PF=export FOAM_INST_DIR=$HOME/OpenFOAM; \
export PYTHONPATH=$FOAM_INST_DIR/PyFoam/lib/python-2.6/site-packages:$PYTHONPATH; \
export PATH=$FOAM_INST_DIR/PyFoam/bin:$PATH
```

Use the alias `PF` to load Python. Two packages are needed for this tutorial, namely Matplotlib and Numpy. Matplotlib can be found on `http://matplotlib.sourceforge.net` and Numpy at `http://numpy.scipy.org`. If you are running Ubuntu you can easily download and compile it using Synaptic Package Manager, just search for the packages `python-matplotlib` and `python-numpy`.

2.6 Histogram

To get some kind of distribution, or visualisation from the sums calculated from the `AlphaCalc.C` utility, we make a script in Python that plots the distribution as a histogram. It looks like

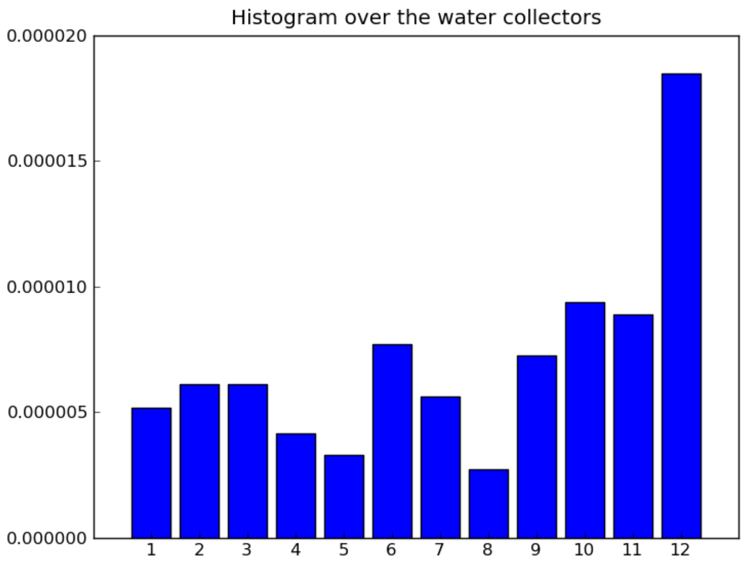
```
#!/usr/bin/env python
import numpy.numarray as na
from pylab import *
import matplotlib.mlab as mlab

X = mlab.load('waterLevel')

labels = ["1","2","3","4","5","6","7","8","9","10","11","12"]

xlocations = na.array(range(len(X)))+0.8
width = 0.8
bar(xlocations, X, width=width)
xticks(xlocations+ width/2, labels)
xlim(0, xlocations[-1]+width*2)
title("Histogram over the water collectors")
gca().get_xaxis().tick_bottom()
gca().get_yaxis().tick_left()

show()
```

First it imports the packages, Numpy and Matplotlib. It then loads the different sums corresponding to the waterlevel of each collector as an array. Since there are twelve different collectors, the length of "waterLevel" should be twelve. They are also labeled using labels, from 1 to 12.

Copy and paste this, save it as `histogram.py` in the `sprinkler/damBreak` directory. To run the script, write

```
python histogram.py
```

This command plots a histogram of the water distribution in the twelve different collectors. Number 1 is the collector closest to the inlet. This is, as said before, the amount of water in each collector calculated from the last timestep in the simulation. This is how the histogram should look like



2.7 Allrun script

In order to run all this including the solver `interFoam` we can make a script, call it `script`. Make it the following way

```
gedit script
```

The file should consist of the following lines

```
interFoam;
AlphaCalc >& log;
foamLog log;
sed -e s/"waterLevel"/""/g logs/waterLevel_0 > waterLevel;
python histogram.py;
```

Afterwards we have to ”tell” linux that this is a script. In order to do that

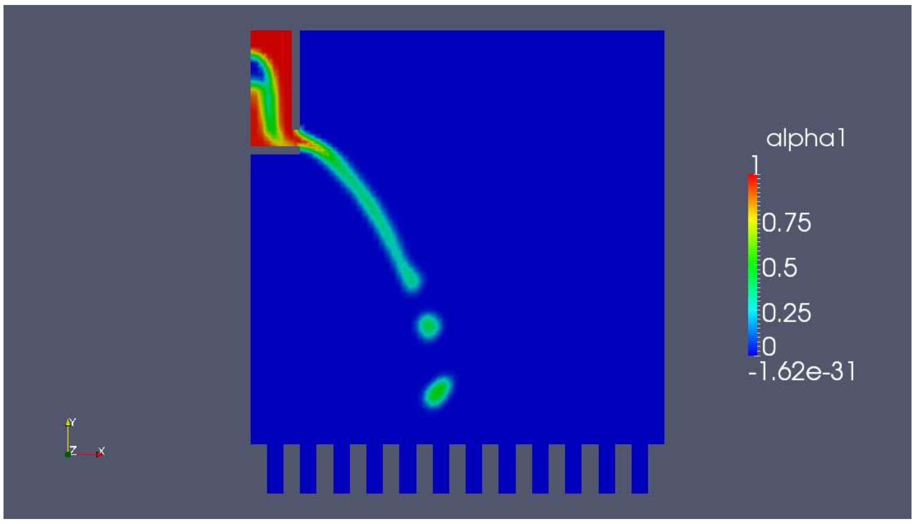
```
chmod 777 script
```

It should now be ”green” and executable using the command

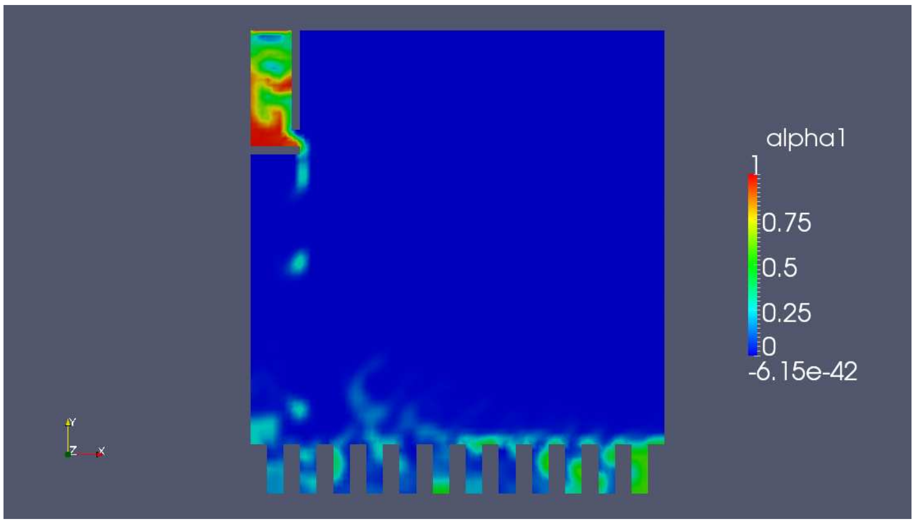
```
./script
```

2.8 Visualisaton of the case

Here is a picture of the case, at timestep `0.6`. This is a quite early timestep, and the velocity of the inlet isn’t that high. The water hasn’t arrived to the water collectors yet.



The picture below shows the simulation at the latest timestep, that is 2. Now it can be seen that the collectors contain some water.



The inlet is defined following in the `0/U` file

```
inlet
{
        type            rampedFixedValue;
        refValueLow     uniform (0 -0.5 0);
        refValueHigh    uniform (0 0 0);
        startRamp       0.1;
        endRamp         1;
        value           uniform (0 0 0);
}
```