OpenFOAM - Spillway Tutorial

Spillway tutorial

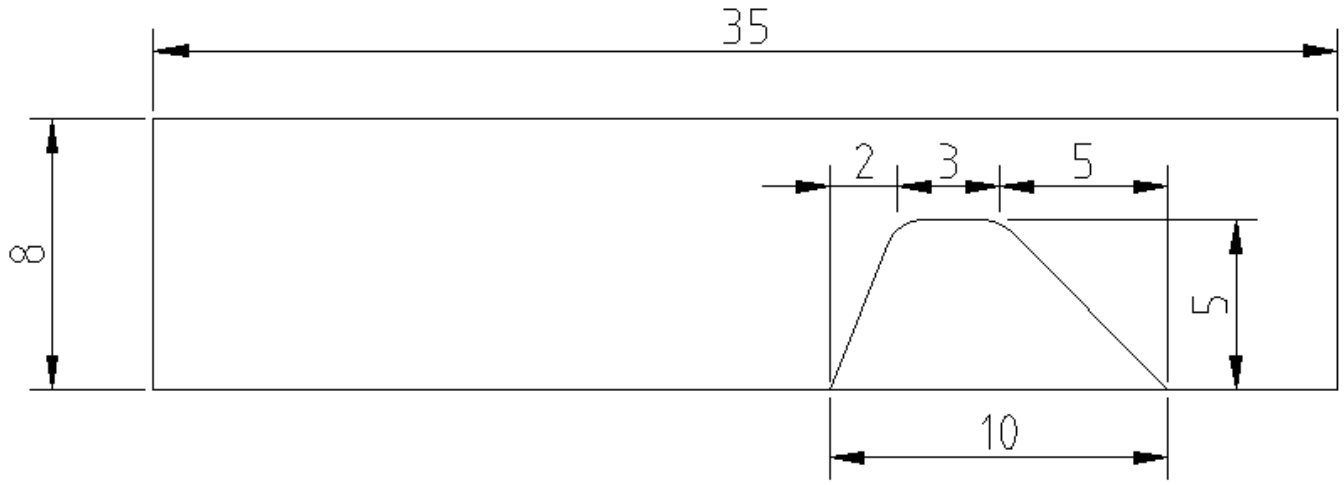
This tutorial is based on cases created by Nils Reidar Bøe Olsen at the Department of Hydraulic and Environmental Engineering. Thank you!

The purpose of this tutorial is to learn to create and analysis slightly more advanced cases in OpenFOAM than the supplied tutorials. This includes the following topics:

- Setup of turbulence model
- Mesh generation with *snappyHexMesh*
- Generation of a 2D mesh with *extrudeMesh*
- Advanced postprocessing in Paraview, including finding and plotting of free surface elevation

> ⚠ **Warning**
>
> Everything presented here are for instructional purposes only, and the results are not validated in any way. Use at your own risk!

The case will be based on the *damBreak* tutorial supplied with OpenFOAM, but we will gather example files from other tutorials as well. In this tutorial we will look at a spillway width dimensions as shown in the image below (all dimensions are in meters):



The water will enter the domain with a fixed velocity along the left face, flow over the dam and out through the right boundary. The corners of the dam is rounded with a radius of 1 meter. As both the water velocity and the area of the inlet is known and fixed, we know the volume flow of water. In this case we will use 3.6 Unknown macro: 'latex' per unit width as the target volume flow. As the solution reaches a steady state (when the discharge over the dam becomes equal to the amount of water entering the reservoir), we can record the height of the free surface.

Before you do this tutorial, you should do the classic *cavity* and *damBreak* tutorials supplied with OpenFOAM. These tutorials are described in the OpenFOAM documentation and in our introduction to OpenFOAM.

To be able to use *snappyHexMesh*, an STL file with the geometry is needed. How to make this is not a topic here, but there are several tools available. The one used in this tutorial is generated with *Autodesk Inventor*, but another good alternative is FreeCAD, witch is available for all common operating systems (Windows, Mac and Linux). If you do not want to generate your own, you can download the one used in this tutorial.

> **View STL files**
>
> If you want to view your STL file to check that it is correct, Paraview can open and display it. Use the "show cube axes" checkbox to check that its dimensions are correct. Please note the location of the origin!

1: Setup of initial case - change of turbulence model

This is clearly a two-phase problem, and that means that we will have to use the *interFoam* solver. The case is also clearly turbulent, so we will have to choose some turbulence model for our problem. To avoid having to write all the input files from scratch, we look in the `$FOAM_TUTORIALS/multiphase/interFoam/ras/` folder. Copy the `damBreak` case to a folder of your choice. If you want you can also rename it to something else, e.g. *spillway*. In the rest of this tutorial all file and folder references will be given relative to this folder if nothing else is specified. It might be wise to try to run the case without any modifications before you proceed. In the following chapters we will go through all files you need to change, and explain the changes made

system/fvSchemes

We will now switch from the Unknown macro: 'latex' turbulence model used in the tutorial to a Unknown macro: 'latex' model. That means that we will have to make some slight adjustments in the differentiation schemes. Replace all occurrences of `epsilon` by `omega` and make sure that the `divSchemes` section looks like:

system/fvSchemes

```
divSchemes
{
    div(rho*phi,U)              Gauss linear;
    div(phi,alpha)              Gauss vanLeer;
    div(phirb,alpha)            Gauss interfaceCompression;
    div(phi,k)                  Gauss upwind;
    div(phi,omega)              Gauss upwind;
    div(phi,R)                  Gauss upwind;
    div(R)                      Gauss linear;
    div(phi,nuTilda)            Gauss upwind;
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}
```

system/fvSolution

In the file `system/fvSolution`, do as in the last file, and replace any occurencies of `epsilon` with `omega`.

constant/RASProperties

The file `constant/RASProperties` contain the information about what turbulence model to use. Switch from `kEpsilon` to `kOmegaSST`:

constant/RASProperties

```
RASModel        kOmegaSST;
turbulence      on;
printCoeffs     on;
```

constant/transportProperties

The `constant/transportProperties` file does not need any changes. Anyways, it is wise to check that the properties of the two phases are correct for air and water respectively. It is also smart to make sure that water is `phase1`, and air is `phase2`. The solver does not care, but we will see later that having water as `phase1` will give some advantages when it comes to the postprocessing.

transportModel

Since both water and air is newtonial fluids, all coefficients refering to other fluid types, such as `CrossPowerLawCoeffs` and `BirdCarreauCoeffs` can be deleted if you like

constant/turbulenceProperties

No changes need to be made to the `constant/turbulenceProperties` file, but again, open it and check that `simulationType` is `RASModel`.

Initial conditions

Since we have changed the turbulence model, and a new variable *omega* has been introduced, we must create boundary and initial condition for this. Simply rename `0/epsilon` to `0/omega`, open the file and replace all occurrences of `epsilon` with `omega`, including switching from `epsilonWallFunction` to `omegaWallFunction`. The initial condition is not important at this stage, and can for instance be left at the current value or set to 0 (will be adjusted later anyway). You will also need to change the dimension for `omega`, as it is different from `epsilon`. The correct dimension are `[0 0 -1 0 0 0 0]`.

Exersise 1

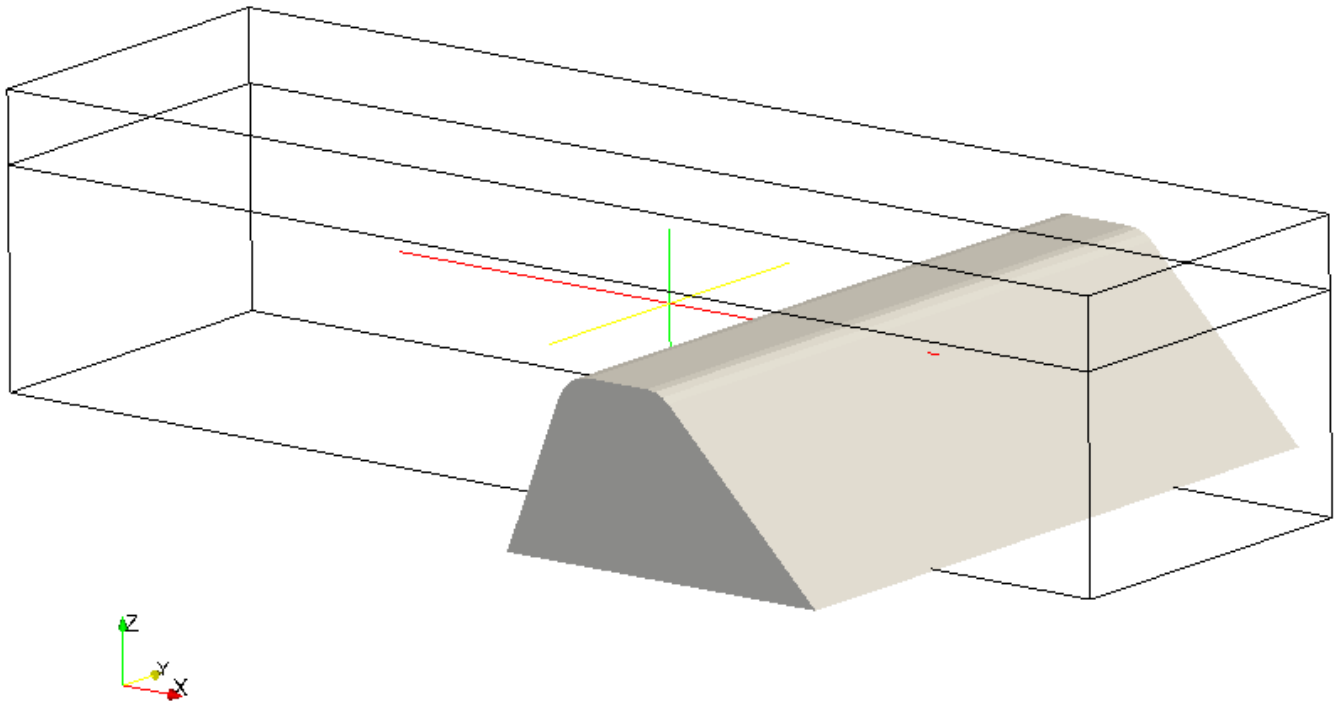
We have now changed the turbulence model from Unknown macro: 'latex' to a Unknown macro: 'latex' model. Follow the procedure in the original *damBreak* tutorial, and try to run this case. If you get any error messages, it is strongly recommended that you correct them before you proceed.

2: Creation of background mesh and boundary conditions

Before we can make the final mesh with *snappyHexMesh*, we will need a simple and coarse background mesh. This will be used as a basis for the meshing in *snappyHexMesh*. The background mesh more or less defines the bounding box and boundaries of our problem. We will also set the boundary conditions in this phase, but we will not include the spillway yet (this is *snappyHexMesh*'s task).

blockMeshDict

The background mesh will consist of two boxes, one upper and one lower. We choose the lower box to be as high as we think the water level behind the spillway will be. This is because we will use the leftmost face of the lower box as an inlet for the water. If the height is chosen properly, the solution will become better and more stable. Otherwise oscillations or standing waves in the free surface might occur.



In the following text we have guessed that the water level behind the spillway will be approximately one meter above the top, and we will create the lower box 6 meters high. The height of the upper box then becomes 2 meters.

Since the geometry in this case is 2D, and we want to "cut out" for the spillway, and not include any ends, it is smart to let the spanwise dimension of the spillway (as defined in your STL file) exceed the spanwise dimension of the mesh, i.e. the maximum and minimum y-coordinate of the spillway should exceed the maximum and minimum y-coordinate of the mesh on both sides (see illustration above). If you have used the attached STL file, you can use the following `blockMeshDict` directly, otherwise you should check your STL file and make the necessary adjustments.

> **Remember to check the reference frame**
>
> Remember to check the location of the origin and directions of the axis in your STL-file. In this case it is assumed that the origin is in the lower left corner of the spillway, and that the z-axis is pointing upwards. If you have made your own STL file, this might be different, and you should make the necessary changes

The `constant/polyMesh/blockMeshDict` file should contain something like this if you are using the supplied STL file:

constant/polyMesh/blockMeshDict

```
convertToMeters 1;

vertices
(
    (-20 -0.05 0 )
    ( 15 -0.05 0 )
    ( 15 -0.05 6 )
    ( 15 -0.05 8 )
    (-20 -0.05 8 )
    (-20 -0.05 6 )
    (-20 -0.15 0 )
    ( 15 -0.15 0 )
```

```
    ( 15 -0.15 6 )
    ( 15 -0.15 8 )
    (-20 -0.15 8 )
    (-20 -0.15 6 )
);

blocks
(
    hex (0 1 2 5 6 7 8 11) (350 60 1) simpleGrading (1 1 1)
    hex (5 2 3 4 11 8 9 10) (350 20 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    inletAir
    {
        type patch;
        faces
        (
            (4 5 11 10)
        );
    }

    inletWater
    {
        type patch;
        faces
        (
            (5 0 6 11)
        );
    }

    outlet
    {
        type patch;
        faces
        (
            (1 2 8 7)
            (2 3 9 8)
        );
    }

    atmosphere
    {
        type patch;
        faces
        (
            (3 4 10 9)
        );
    }

    bottomWall
    {
        type wall;
        faces
        (
            (0 1 7 6)
        );
    }

    front
    {
        type empty;
        faces
        (
            (1 0 5 2)
            (2 5 4 3)
```

```
        );
    }

    back
    {
        type empty;
        faces
        (
            (6 7 8 11)
            (11 8 9 10)
        );
    }

);

mergePatchPairs
(
);
```

If you are unsure about the meaning of anything in this file, please consult he OpenFOAM documentation.

Boundary conditions

The boundary conditions are set in the `0`-folder. Since for instance `0/alpha1` are overwritten by the *setFields* utility, it is often wise to create a folder `0.org` with the original and unmodified boundary and initial condition files. Therefore copy the entire `0`-folder to a new folder called `0.org`. The variables that need boundary conditions in the simulation are:

- alpha1
- k
- omega
- p_rgh
- U

Make sure that all these files are present. If any more files are present (for example `nuTilda`, `nut` or `alpha1.org`) they can be deleted. To help us judging the boundary conditions for the turbulent variables *k* and *omega*, you can use the CFD-online turbulence calculator. With a target volume flow of 3.6 Unknown macro: 'latex' per unit width over a height of 6 meter, that will give a inlet (freestream) velocity of 0.6 m/s. You can use your own judgments for the turbulence intensity levels and length scales (whether this spillway is in a river or in a great lake will affect the choices you make). In this example we have used 2% turbulence intensity with a characteristic length scale of 0.1 meter.

The boundary conditions that are to be implemented then becomes:

	alpha1	k	omega	p_rgh	U
inletAir	*fixedValue* of 0	*fixedValue* of 2.16e-4	*fixedValue* of 0.1470	*fixedValue* of 0	*fixedValue* of (0 0 0)
inletWater	*fixedValue* of 1	*fixedValue* of 2.16e-4	*fixedValue* of 0.1470	*fixedValue* of 0	*fixedValue* of (0.6 0 0)
outlet	*zeroGradient*	*zeroGradient*	*zeroGradient*	*buoyantPressure* of 0	*zeroGradient*
atmosphere	*inletOutlet*	*inletOutlet* of 2.16e-4	*inletOutlet* of 0.1470	*totalPressure*	*pressureInletOutletVelocity of 0*
bottomWall	*zeroGradient*	*kqRWallFunction*	*omegaWallFunction*	*buoyantPressure* of 0	*fixedValue* of (0 0 0)
front, back, defaultFaces	*empty*	*empty*	*empty*	*empty*	*empty*

The boundary conditions can of course be discussed. Feel free to alter the boundary conditions if you like. If you wonder why this boundary conditions have been chosen, it is simply because it is the same boundary conditions that is already implemented in the case we copied in the beginning, and only some patch names and the values need to be changed.

Groups of patches

You can group patches together if two or more require the same boundary conditions. If you want to use the same conditions on both *front*, *back* and *defaultFaces*, you can replace the patch name with `"(front|back|defaultFaces)"`.

Initial conditions

The initial conditions can be given as constant velocity of (0.6 0 0) m/s, with the same turbulent properties as the freestream properties of water. The pressure and phase fraction `alpha1` can both be set to zero at the moment.

g

This tutorial is in the XZ-plane with the Z-axis pointing upwards, and we must check that the gravity is applied correctly, i.e. acceleration in the negative Z-direction. Open the `constant/g` file, and switch the direction of gravity to:

constant/g

```
value           (0 0 -9.81);
```

Exersise 2

We have now made a new mesh, boundary and initial conditions. Before we try to run this case it might be wise to clean up the results from any previous runs. This is done with the `foamClearPolyMesh` and `foamCleanTutorials` commands. In addition to this we will have to move our initial and boundary condition files from the `0.org`-folder to the `0`-folder:

Terminal window

```
foamClearPolyMesh
foamCleanTutorials
cp 0.org/* 0/
blockMesh
interFoam
```

and see if it works. Correct any errors you have made, and inspect the results in Paraview. As the mesh consist of relatively few cells of large size, it should not be necessary to decompose the case and run it in parallel yet.

3: Mesh creation with *snappyHexMesh*

It is now time to do the (perhaps) most difficult task: to create a mesh with *snappyHexMesh*. Brefore we start on this, you are strongly encouraged to read the OpenFOAM documentation on *snappyHexMesh*, as that tool is not very intuitive at first.

snappyHexMesh and memory

You should be aware that *snappyHexMesh* is a pure 3D tool, and will create a 3D mesh. It will refine your base mesh several times, and each refinement level will divide each cell subject to refinement in 8 new cells (each original cell becomes a 2x2x2 cell). This consumes a lot of memory, and you might potentially crash your computer. Therefore, before running *snappyHexMesh*, close all non-essential programs, close most of the tabs in your browser (all except this tutorial) and save all important work.

Finding a snappyHexMeshDict file

As nobody wants to write the input files for *snappyHexMesh* from scratch, we will copy it from one of the examples supplied with OpenFOAM. Look in the `$FOAM_TUTORIALS/incompressible/pimpleDyMFoam/wingMotion/wingMotion_snappyHexMesh/system/` folder and copy the `snappyHexMeshDict` file into your `system` directory. Also copy the `$FOAM_TUTORIALS/incompressible/pimpleDyMFoam/wingMotion/wingMotion2D_simpleFoam/system/extrudeMeshDict` file to your `system` directory (we will need it later). This can be done with the following commands in the terminal:

Terminal window

```
cp $FOAM_TUTORIALS/incompressible/pimpleDyMFoam/wingMotion/wingMotion_snappyHexMesh/system/snappyHexMeshDict
system/
cp $FOAM_TUTORIALS/incompressible/pimpleDyMFoam/wingMotion/wingMotion_snappyHexMesh/system/extrudeMeshDict
system/
```

Creating a mesh around the spillway

The first thing we want to do is to "cut out" a hole in our base mesh, fit this to the spillway and then make some refined layers around the spillway such that the effects of the walls are correctly resolved. We will here go through the `system/snappyHexMeshDict` file to make the neccesary changes. Before we start, you should place your STL file in a folder `constant/triSurface`. We will assume that the file is named `dam.stl`.

The first thing to do in the `system/snappyHexMeshDict` file is to specify the STL file to use. This is done in the *geometry* section. Remove the already defined `wing_5degrees.obj` section and insert a new for our spillway:

system/snappyHexMeshDict

```
    dam.stl
    {
        type triSurfaceMesh;
        name dam;
    }
```

The *refinementBox* section below can be commented out, as we will modify this later. The comments are standard C-syntax, i.e. you can add a double backslash at the beginning of each line to be commented out, or use the `/* something */` block comment style.

The next setting to be changed is the `minRefinementcells` option. For this (rather small and simple) case, it can be set to 0, i.e. we will not tolerate any bad cells on the refinement surface. `nCellsBetweenLevels` can also be reduced to for example 3 (this is the setting used in this tutorial). Leaving it as it is will create a thicker refinement layer around the spillway and a mesh with overall more cells.

In the *refinementSurfaces* section the surface name must be changed from `wing_5degrees.obj` to `dam`. The refinement levels can for example be reduced from "5 5" to "4 5" here. Everything inside the following *refinementRegions* must also be commented out (we will come back to this later).

`locationInMesh` is a coordinate that must be within your mesh region. It simply points out what part of the mesh that should be kept after *snappyHexMesh* has read the STL file and done the edge snapping. (-2 -0.1 2.5) will be fine in this example.

The last ting we want to change in the `snappyHexMeshDict` file is that we want to create some layers on our walls, both the *bottomWall* and the surface of the spillway. This layer addition process is controlled by the settings in the *addLayersControl* section. First change the `"wing.*" by (bottomWall|dam).`. This means that both the surface *bottomWall* and the surface *dam* will get layers added. You can reduce the number of layers to 2 if you like. The other settings can be adjusted to be:

system/snappyHexMeshDict

```
expansionRatio 1.0;
finalLayerThickness 0.3;
minThickness 0.1;
```

None of these settings are essential, and you might want to play around with them ans look at the differences afterward.

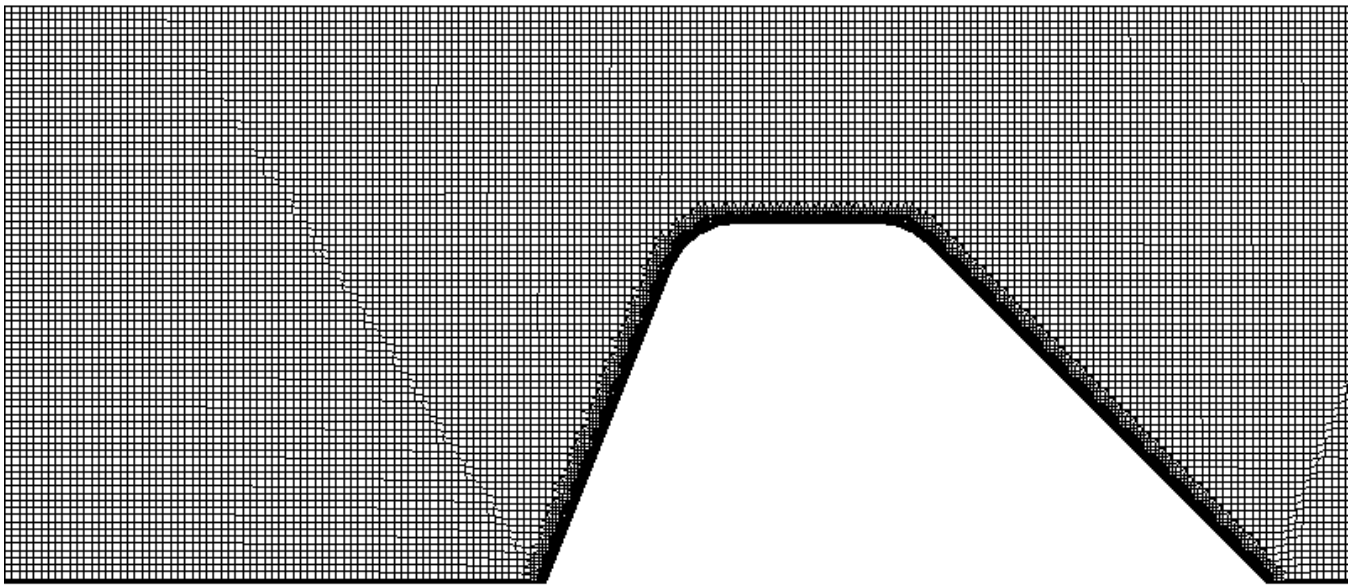
Exersise 3a

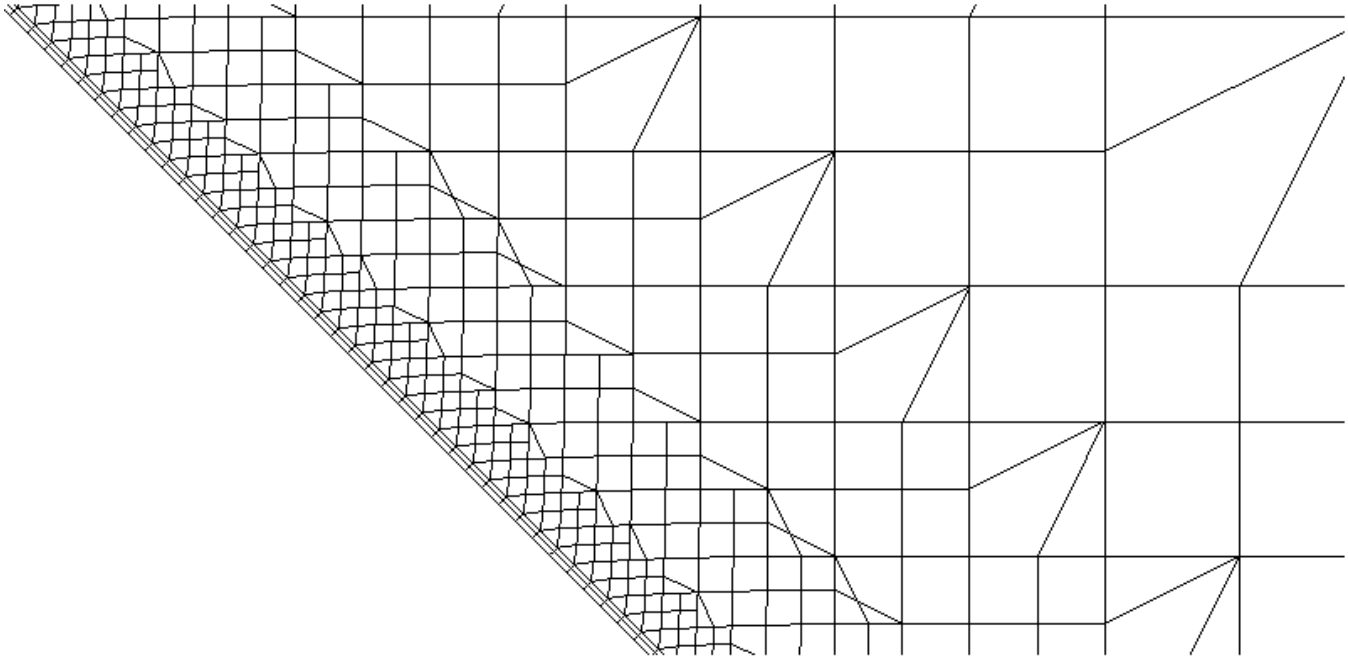
Now create the mesh:

Terminal window

```
blockMesh
snappyHexMesh -overwrite
```

Inspect the result in Paraview. Observe that the mesh is now fully 3D around the dam. The resulting mesh should look something like this:





Try to play with different refinement levels and find out how it works!

Creation of refinement regions (optional)

If you want to better resolve the free surface you might want to create a finer mesh around where we previously have guessed that our free surface should be (1 meter above spillway). Open your `system/snappyHexMeshDict` file again and uncomment the *refinementBox* section in the start that we previously commented out. It can be renamed to anything you like, for example *surface*. Enter coordinates such that it encapsulates the region you want refined. The coordinates of your box might exceed the mesh dimensions if you like. The result should be something like:

system/snappyHexMeshDict

```
    surface
    {
        type searchableBox;
        min ( -15  -1  4.5 );
        max (   4   0  7   );
    }
```

The coordinates must be two opposite corners of the box, and all the coordinates in the minimum-corner must be less than the coordinate values in the maximum corner (i.e. the coordinates of a vector from *min* to *max* must all be positive).

Now we need to scroll down to the *refinementRegions* section of the file. Uncomment the *refinementBox* section that was previously commented out. If you changed the name of *refinementBox* previously, you should do that here as well. For this case it is probably sufficient with one level of refinenemt, so change make sure the section looks like:

system/snappyHexMeshDict

```
 refinementRegions
    {
        surface
        {
            mode inside;
            levels ((1E15 1));
        }
    }
```

The first number in *levels* is ignored for mode *inside*.

Exercise 3b

Recreate the mesh with the new settings:

Terminal window

```
foamClearPolyMesh
blockMesh
snappyHexMesh -overwrite
```

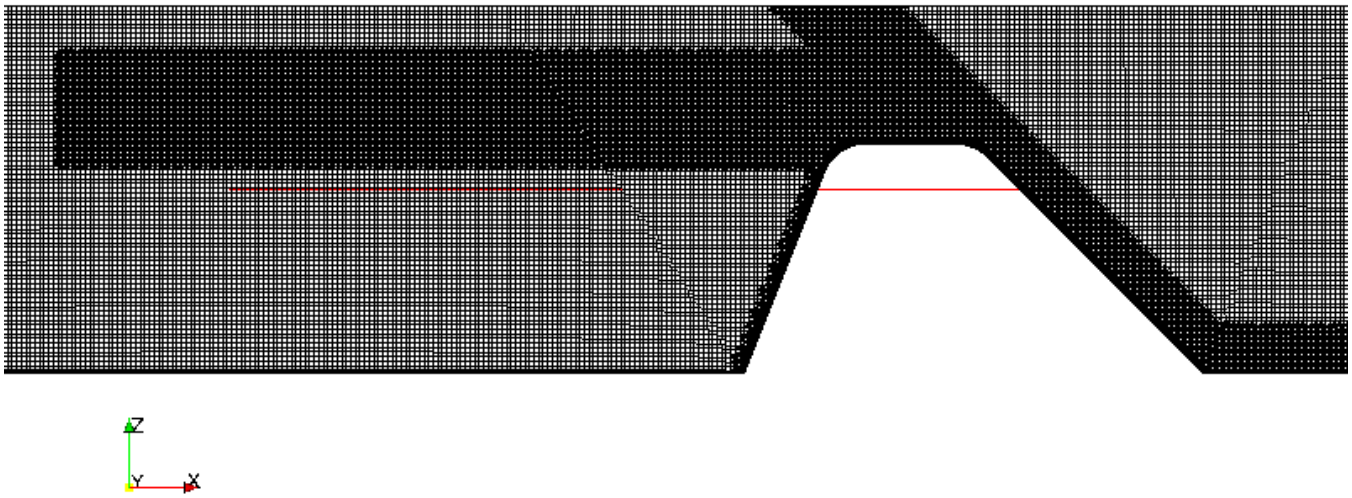
Observe that there now is a refined region around where the free surface is going to be. This will give increased resolution and a sharper interphase.

Exersise 3c

Add two new refinement regions. One should be on the downstream side, and one should be along the bottom wall after the spillway. Both refinement regions are meant to better capture the flow behind.

Refinement regions

The region along the bottom wall can be created the same way as we created the region around the free surface (with a `searchableBox`). The region along the downstream side of the spillway is on the other hand created easiest by using `type searchablePlane`. You can read more about this in snappyWiki. If you use the `searchablePlane` type, remember to set the distance in the `levels` parameter to some other value than 1e15 meter (for example 1 meter), or the entire mesh will be refined.



4: Creation of a 2D mesh with extrudeMesh

As previously mentioned, the mesh from *snappyHexMesh* is a 3D mesh. Depending on the choices made when defining refinement surfaces and regions, the number of cells can vary much, but you probably have a mesh with somewhere between 200 000 and 400 000 cells. This is unnecessary and a waste of compute resources and time in this simple case. What we want to do now is to take the *front* patch of our mesh and extrude it 1 cell in the positive y-direction. This will create a new mesh that is pure 2D. This is done by the tool called *extrudeMesh*. Open the previously copied `system/extrudeMeshDict` file, and set the `sourceCase` to `"."`.

Exercise 4a

Check the number of cells in the mesh (with the `checkMesh` command) before and after running the `extrudeMesh` command. Also look at some of the statistics that *checkMesh* gives you about mesh quality. If the mesh fails one or more checks, you should try to find out the cause of the error and fix it.

Setting boundary conditions on the spillway

As we now have introduced a new patch in our domain, we have to set boundary conditions on this as well. Go through all the files in the *0.org* folder, and make a new boundary with the exact same boundary conditions as *bottomWall*. The name of your new boundary might vary, depending on your STL file. If you created the file with *FreeCAD*, it is probably named `dam_Mesh`, and if you used Autodesk Inventor, it is probably called `dam_ascii`.

> **Use of wildcards**
>
> OpenFOAM allows for the use of grouping and wildcards in patch names. If you want to apply the same boundary conditions on both *bottomWall* and the spillway, you can group them together and use a trailing wildcard. Use `"(bottomWall|dam).*"` as the patch name in this case.

setFieldsDict

To reduce solution time, we will initialize the case with water behind the spillway up to the edge (z=5 meter). In the `system/setFieldsDict` file, change the box coordinates so that it encapsules the part behind the dam:

system/setFieldsDict

```
boxToCell
    {
        box (-20 -1 0) (3 1 5);
        fieldValues
        (
            volScalarFieldValue alpha1 1
        );
    }
```

Exercise 4b

We have all our boundary condition files in a folder `0.org`, but we must put them in the folder `0`. Therefore, delete all the files already present in the `0` folder (*snappyHexMesh* creates some files there that becomes invalid after mesh extrusion. Those files must be removed.) and copy the contents of `0.org` into `0`. Then you can use the *setFields* utility to initialize the volume fraction:

Terminal window

```
rm 0/*
cp 0.org/* 0/
setFields
```

Visualize the result in Paraview, and verify that everything is correct:

5: Running the interFoam solver

Before we run the solver, a few more changes need to be made to the files controlling the simulation:

controlDict

The `system/controlDict` file is the main control file of the simulation. Open the file, find the keywords and adjust if necessary so that the lines are:

system/controlDict

```
endTime          30;   // To simulate until steady state conditions
adjustTimeStep  yes;   // With this parameter on, OpenFOAM itself will adjust the timestep according to maxCo
and maxAlphaCo
maxCo           0.5;   // Give the value of maximum Courant number
maxAlphaCo      0.5;   // Same as above
```

Read through the rest of this file and make sure that you do understand what all the parameters mean. Note that we now have enabled automatic time-step scaling based on the Courant number. OpenFOAM will at each time iteration scale the timestep to satisfy Unknown macro: 'latex'.

decomposeParDict

The `system/decomposeParDict` file controls the decomposition in several processes. Since we have one main direction, we will only decompose along the x-axis. Make sure that you decompose the domain in the same number of parts as you have processor cores available on your computer. You can choose decomposition method `simple` or `hierarchical` as you like, they will behave the same in this case. In this example we will use four processors and the simple method. The `system/decomposeParDict` file should then contain:

system/decomposeParDict

```
numberOfSubdomains 4;
method             simple;

simpleCoeffs
{
    n               ( 4 1 1 );
    delta           0.001;
}
```

Start and wait

You are now ready to start the analysis. The first you do is to decompose the case , and then you start the solver in parallel:

Terminal window

```
decomposePar
mpirun -np 4 interFoam -parallel
```

The analysis will probably take several hours, so you can for example start it in the evening, and it should be ready the next morning. If you have decomposed in more or less domains than 4, this must be changed in the `mpirun` command. It might be wise to start the solver, let it run until the simulation time is between 1 and 2 seconds, and then stop it with *Ctrl+C*. Open the case in Paraview as described below and verify that thing works. Then you can start the simulation again with `mpirun -np 4 interFoam -parallel`.

6: Postprocessing in Paraview

When the analysis is finished it is time to find the answer to the original question: *given a volume flow of 3.6* Unknown macro: 'latex' *, what is the water level behind the spillway*? This can be found using the post-processing tools in Paraview.

When it comes to running Paraview, you have two choices:

1. Recompose the case with *reconstructPar* as taught in the original *damBreak* tutorial, and then open Paraview with *paraFoam*.
2. Create an empty file called `spillway.foam` (or something else ending in `.foam`), open this (empty) file in Paraview (no reconstruction needed). The last option is definitely the fastest, and will be used here.

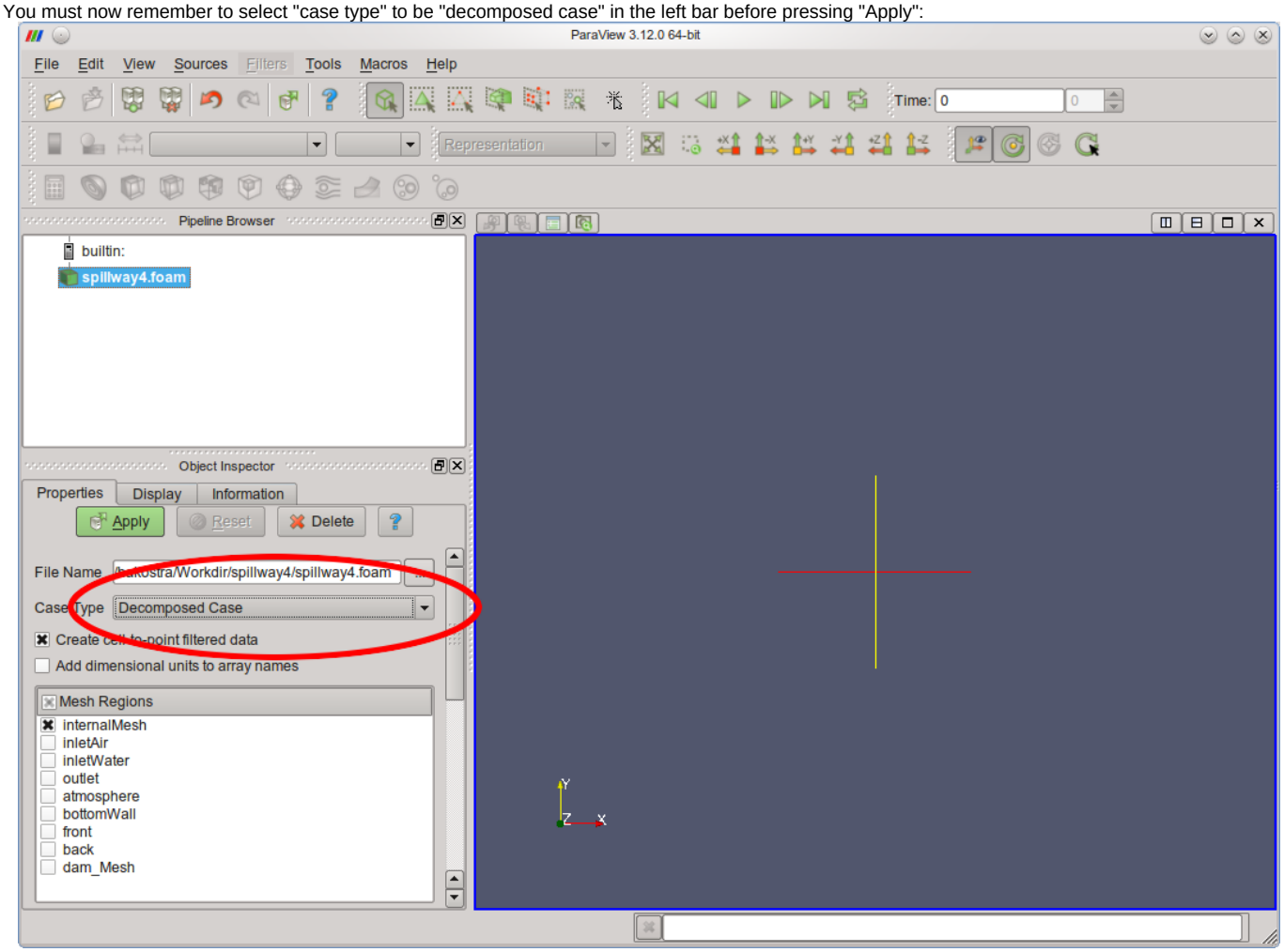
Open the decomposed case in Paraview

To create an empty file, we can use the `touch` command. Then we open the file in Paraview:

Terminal window

```
touch spillway.foam
paraview spillway.foam
```

You must now remember to select "case type" to be "decomposed case" in the left bar before pressing "Apply":



Exercise 6a

Look at the results by playing the animation. Does the result seem to be reasonable? How well was our initial guess about the water level behind the dam?

Finding the free surface elevation at any point

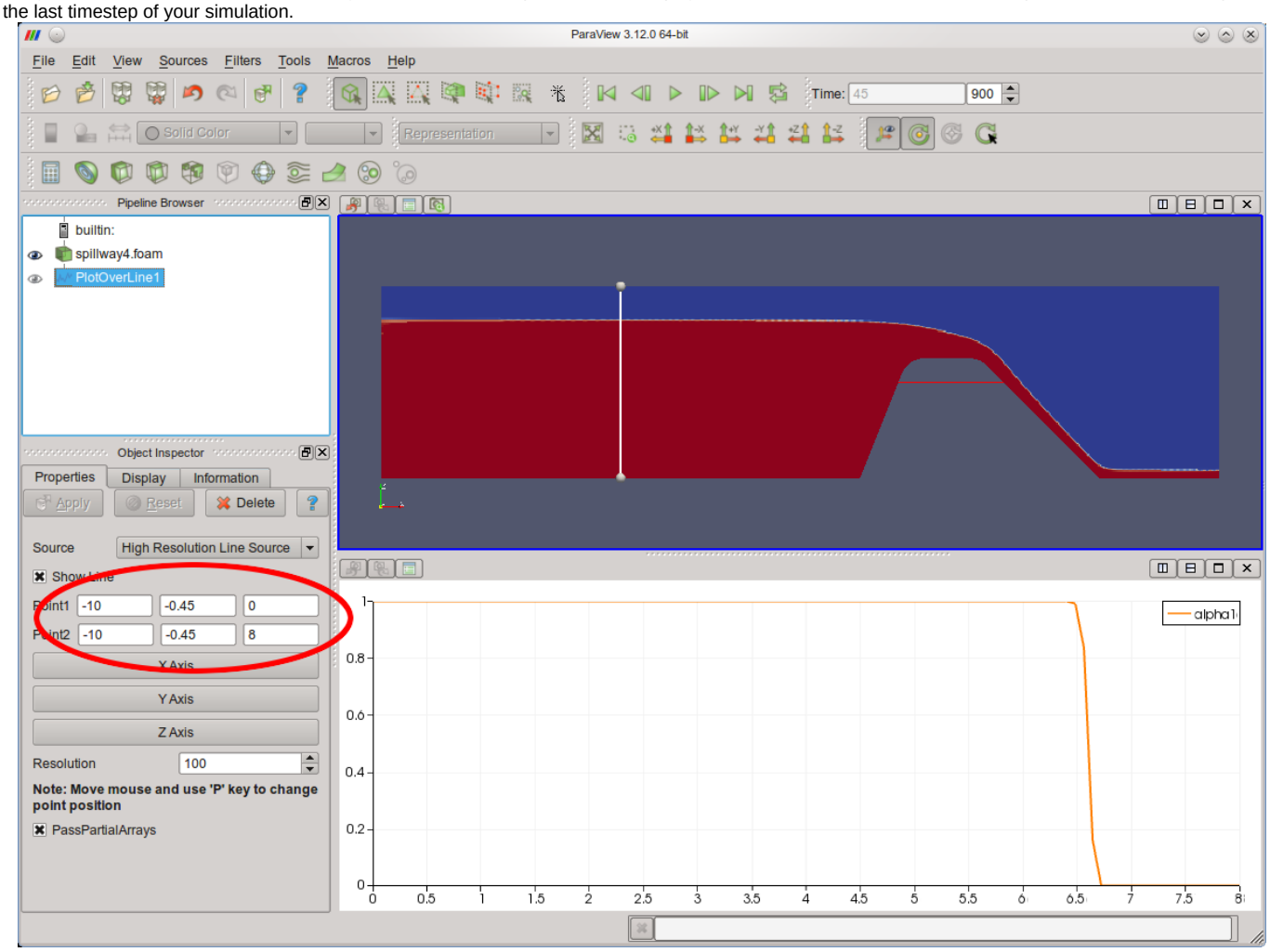
To find the free surface elevation at a point can at first seem to be somewhat difficult, as Paraview does not have any such "surface elevation tool". Another difficulty is that OpenFOAM itself does not solve for the surface elevation, it solves for the volume fraction Unknown macro: 'latex'. That means that we will at least have one cell, probably more, where the volume fraction is somewhere in the region Unknown macro: 'latex'. There are (at least) two common ways to find the surface elevation from the volume fraction:

1. Sample the volume fraction along a vertical line parallel to the z-axis crossing the free surface, and use interpolation to find where Unknown macro: 'latex'.
2. Integrate the volume fraction from the bottom of your domain to the atmosphere along a line parallel to the z-axis:

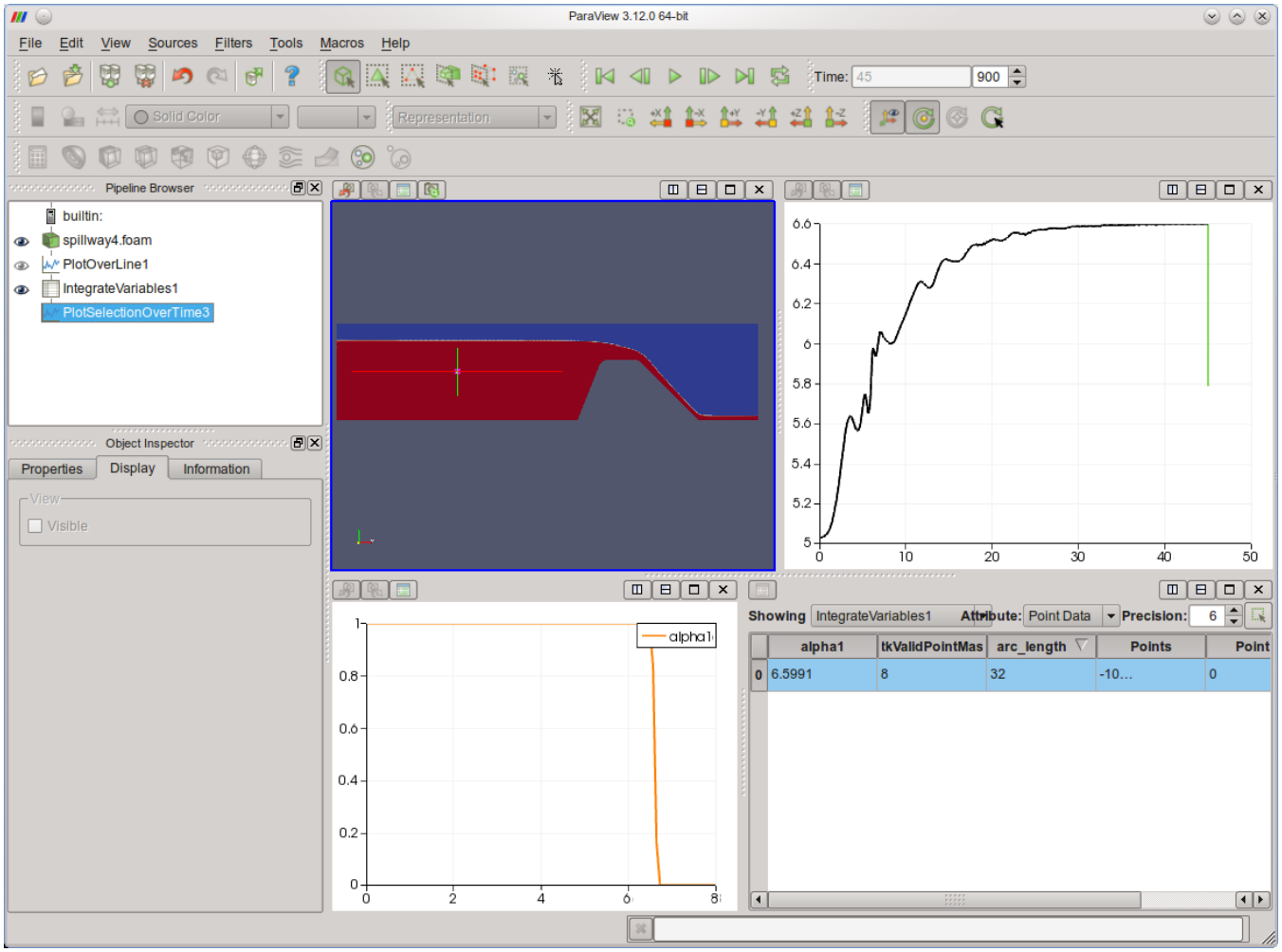
Unknown macro: 'latex'

The latter method will be used here, as it is easy implementable in Paraview using standard tools.

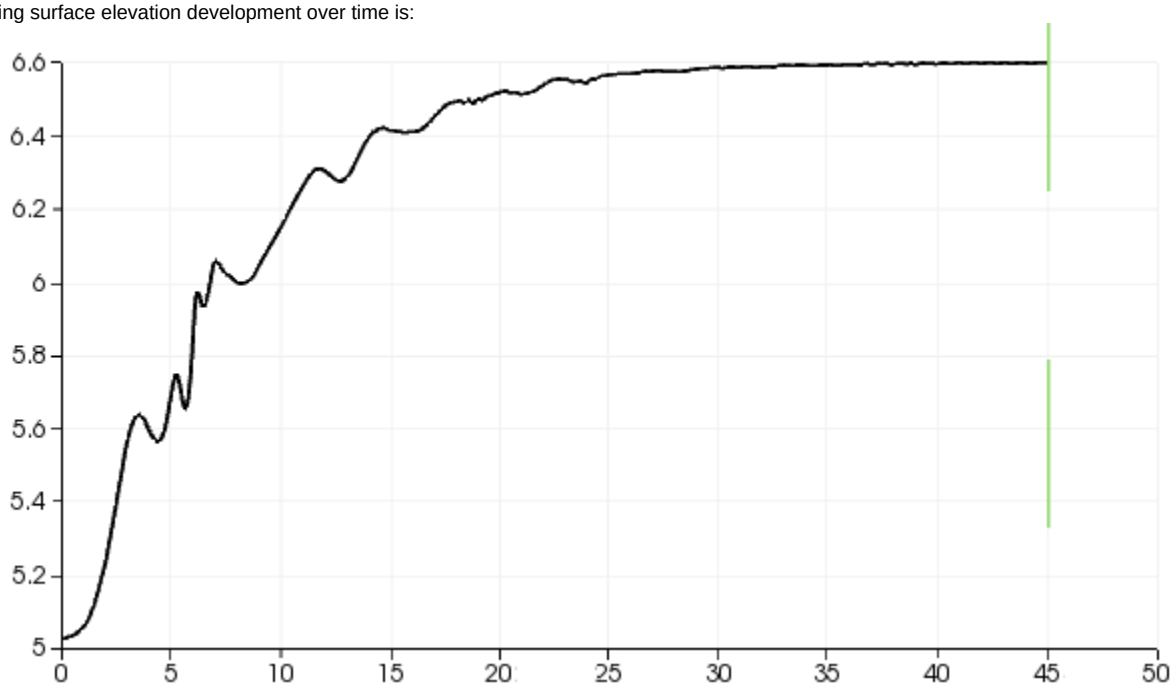
The first thing we need to do is to sample the volume fraction along a line. This is done by using the *Plot Over Line* utility, found under *Filters -> Data Analysis*. Create a line from the bottom of your domain to the top. Under the "Display" tab, deselect all other variables than *alpha1*. Remember to step to the last timestep of your simulation.



Now we want to integrate the volume fraction along this line. This is done by the *Integrate Variables* tool, also found under *Filters -> Data Analysis*. Press "Apply", and you see the integrated variables. Then we want to plot this integrated variable over time, so that we can see how the free surface elevation changes during the simulation. This is done with the *Plot Selection Over Time* tool, and now you can guess yourself where to find it. Select the row from the table with the integrated variables, and choose "Copy active selection". Then press "Apply" and wait. This might take some time, as it will sample the volume fraction and integrate it at every timestep stored. Go and grab a cup of coffee while you wait. After the process has finished, you can deselect those variables not needed in the plot. The resulting Paraview workspace now looks something like this:



The resulting surface elevation development over time is:



In this example the simulation is proceeded to 45 seconds, and as we can see, the surface elevation is more or less converged.

Bug in Paraview

At the time of writing there is a bug in Paraview when using "Plot Selection Over Time" with the parallel OpenFOAM reader (i.e. when Paraview reads the decomposed case directly as described here). The result is thousands of error messages and no plot of surface elevation over time. A possible workaround is to reconstruct the parallel case with the command `reconstructPar` and open Paraview with `paraFoam`.

Exercise 6b

Create a nice and illustrative image of the results, with the the spillway, streamlines and an isosurface at alpha1=0.5 (what does that mean?) (final timestep only).

Exercise 6c

Restart the simulation and run another 15 seconds (if you stopped at 30 seconds). Re-run the postprocessing and check the difference in final surface elevation. Has the solution converged?

Exercise 6d

Use the "calculator" utility in Paraview, and calculate the volume flow of *water* out of the domain at any time. The procedure is roughly the same as when we plotted the free surface elevation. Is 30 seconds of simulation time enough to reach steady state?

7: Futher work

There are a lot of interesting exercises that can be done based on this tutorial. Here are a few:

1. Adjust the height of the inflow region (and the inflow velocity) to match the free surface elevation recorded previously. Did this change the results?
2. When doing CFD simulations (and other simulations as well) it is common to make the mesh as fine as the computing resources allow within the boundaries of your desired solution time. Now try to make the background mesh as coarse as possible, make refinements where needed, and see how far down you can get the solution time!
3. Is the "inlet length" (the length from the inlet to the dam) long enough? Will a longer inlet change the surface elevation?
4. Read about swak4Foam and follow the basic example given to measure surface elevation at runtime and make a new and better inlet.

8: Download

You can download all necessary case files in spillway.tar.gz. Scripts to run and clean the case is also attached.