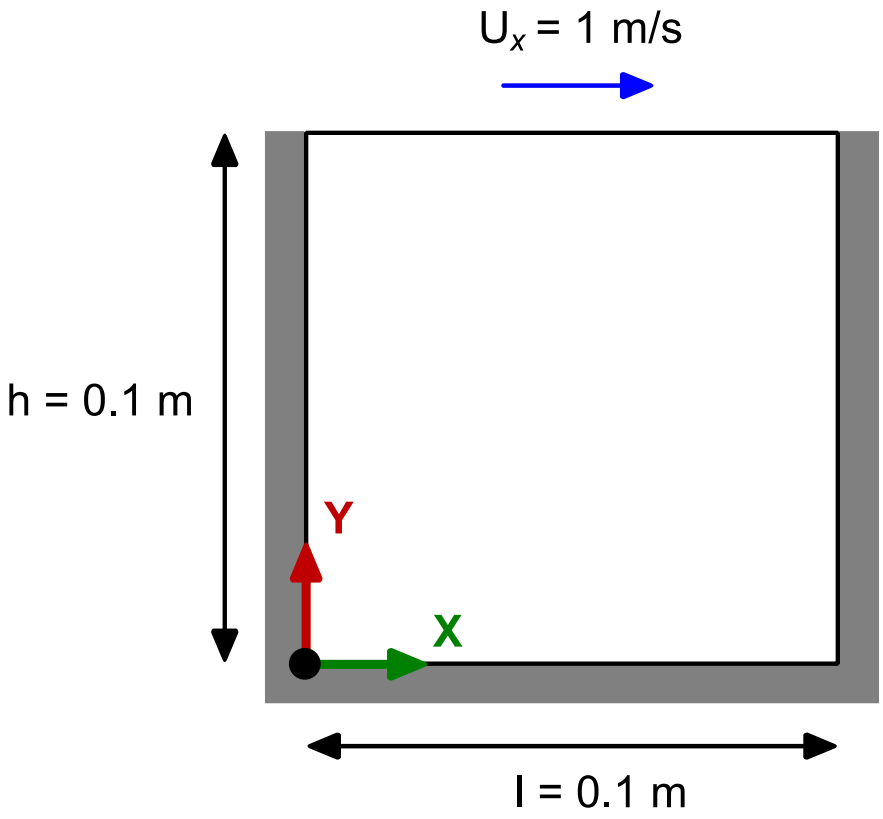
Running my first OpenFOAM® case

Flow in a lid-driven square cavity – Re = 100
Incompressible flow

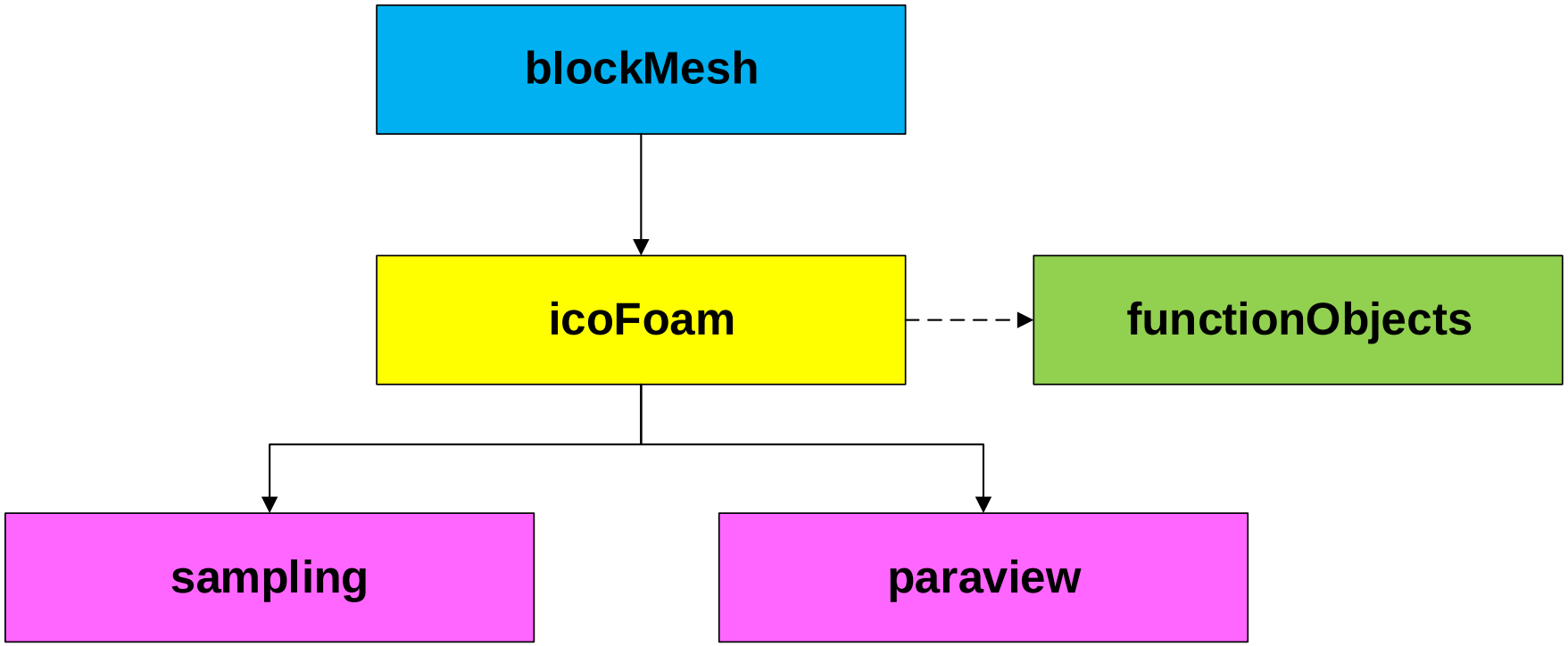


Physical and numerical side of the problem:

- The governing equations of the problem are the incompressible laminar Navier-Stokes equations.
- We are going to work in a 2D domain but the problem can be extended to 3D easily.
- To find the numerical solution we need to discretize the domain (mesh generation), set the boundary and initial conditions, define the flow properties, setup the numerical scheme and solver settings, and set runtime parameters (time step, simulation time, saving frequency and so on).
- For convenience, when dealing with incompressible flows we will use relative pressure.
- All the dictionaries files have been already preset.

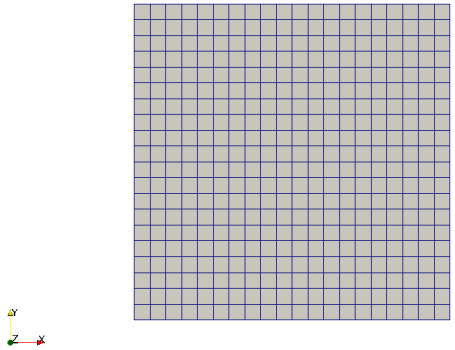
Running my first OpenFOAM® case

Workflow of the case



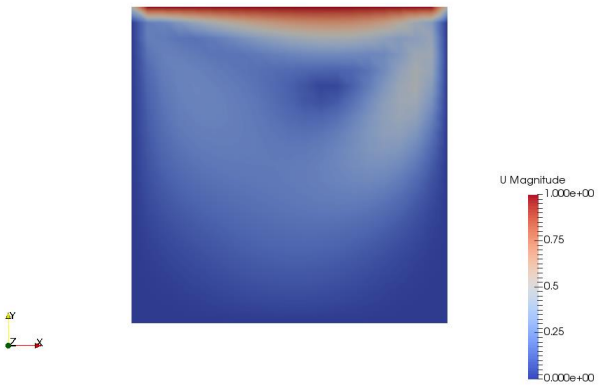
Running my first OpenFOAM® case

At the end of the day you should get something like this



Mesh (very coarse and 2D)

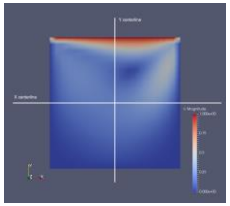
Pressure field (relative pressure)



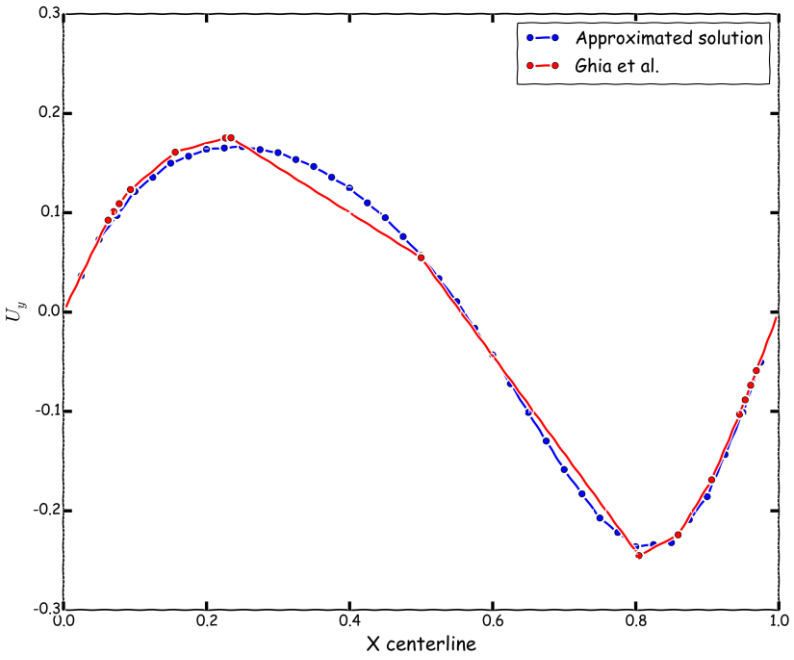
Velocity magnitude field

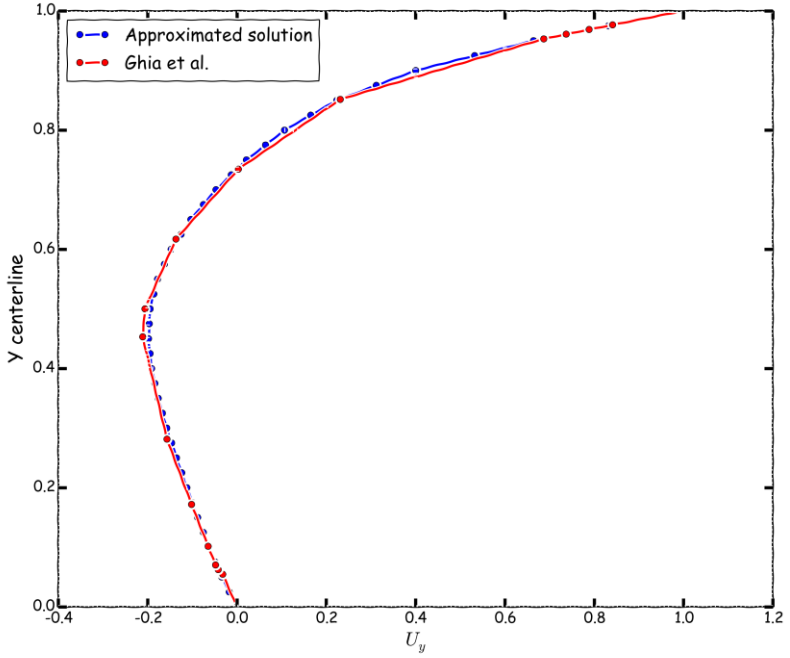
Running my first OpenFOAM® case

At the end of the day you should get something like this



And as CFD is not only about pretty colors, we should also validate the results





High-Re Solutions for incompressible flow using the navier-stokes equations and a multigrid method
U. Ghia, K. N. Ghia, C. T. Shin.
Journal of computational physics, 48, 387-411 (1982)

Running my first OpenFOAM® case

Loading OpenFOAM® environment

- If you are using our virtual machine or using the lab workstations, you will need to source OpenFOAM® (load OpenFOAM® environment).

- To source OpenFOAM®, type in the terminal:

    - `$> of4x`

- To use PyFoam you will need to source it. Type in the terminal:

    - `$> anaconda2` or `$> anaconda3`

- Remember, every time you open a new terminal window you need to source OpenFOAM® and PyFoam.

- By default, when installing OpenFOAM® and PyFoam you do not need to do this. This is our choice as we have many things installed and we want to avoid conflicts between applications.

Running my first OpenFOAM® case

What are we going to do?

- We will use the lid-driven square cavity tutorial as a general example to show you how to set up and run solvers and utilities in OpenFOAM®.
- In this tutorial we are going to generate the mesh using `blockMesh`.
- After generating the mesh, we will look for topological errors and assess the mesh quality. For this we use the utility `checkMesh`. Later on, we are going to talk about what is a good mesh.
- Then, we will find the numerical solution using `icoFoam`, which is a transient solver for incompressible, laminar flow of Newtonian fluids. By the way, we hope you did not forget where to look for this information.
- And we will finish with some cool visualization and post-processing using `paraFoam`.
- While we run this case, we are going to see a lot of information on the screen (standard output stream or stdout), but it will not be saved. This information is mainly related to convergence of the simulation, we will talk about this later on.

Running my first OpenFOAM® case

Running the case blindfold

- Let us run this case blindfold. Later we will study in details each file and directory.
- In the terminal window type:

1. `$> cd $PTOFC/101OF/cavity`
   Remember, `$PTOFC` is pointing to the path where you unpacked the tutorials.
2. `$> ls -l`
3. `$> blockMesh`
4. `$> checkMesh`
5. `$> icoFoam`
6. `$> postProcess -func sampleDict -latestTime`
7. `$> gnuplot gnuplot/gnuplot_script`
8. `$> paraFoam`

Running my first OpenFOAM® case

Running the case blindfold

- In step 1 we go to the case directory. Remember, `$PTOFC` is pointing to the path where you unpacked the tutorials.
- In step 2 we just list the directory structure. Does it look familiar to you? In the directory **`0`** you will the initial and boundary conditions, in the **`constant`** directory you will find the mesh information and physical properties, and in the directory **`system`** you will find the dictionaries that controls the numerics, runtime parameters and sampling.
- In step 3 we generate the mesh.
- In step 4 we check the mesh quality. We are going to address how to assess mesh quality later on.
- In step 5 we run the simulation. This will show a lot information on the screen, the standard output stream will not be saved.
- In step 6 we use the utility `postProcess` to do some sampling only of the last saved solution. This utility will read the dictionary file named *`sampleDict`* located in the directory **`system`**.
- In step 7 we use a gnuplot script to plot the sampled values. Feel free to take a look and reuse this script.
- Finally, in step 8 we visualize the solution using `paraFoam`. In the next slides we are going to briefly explore this application.

Running my first OpenFOAM® case

Crash introduction to paraFoam

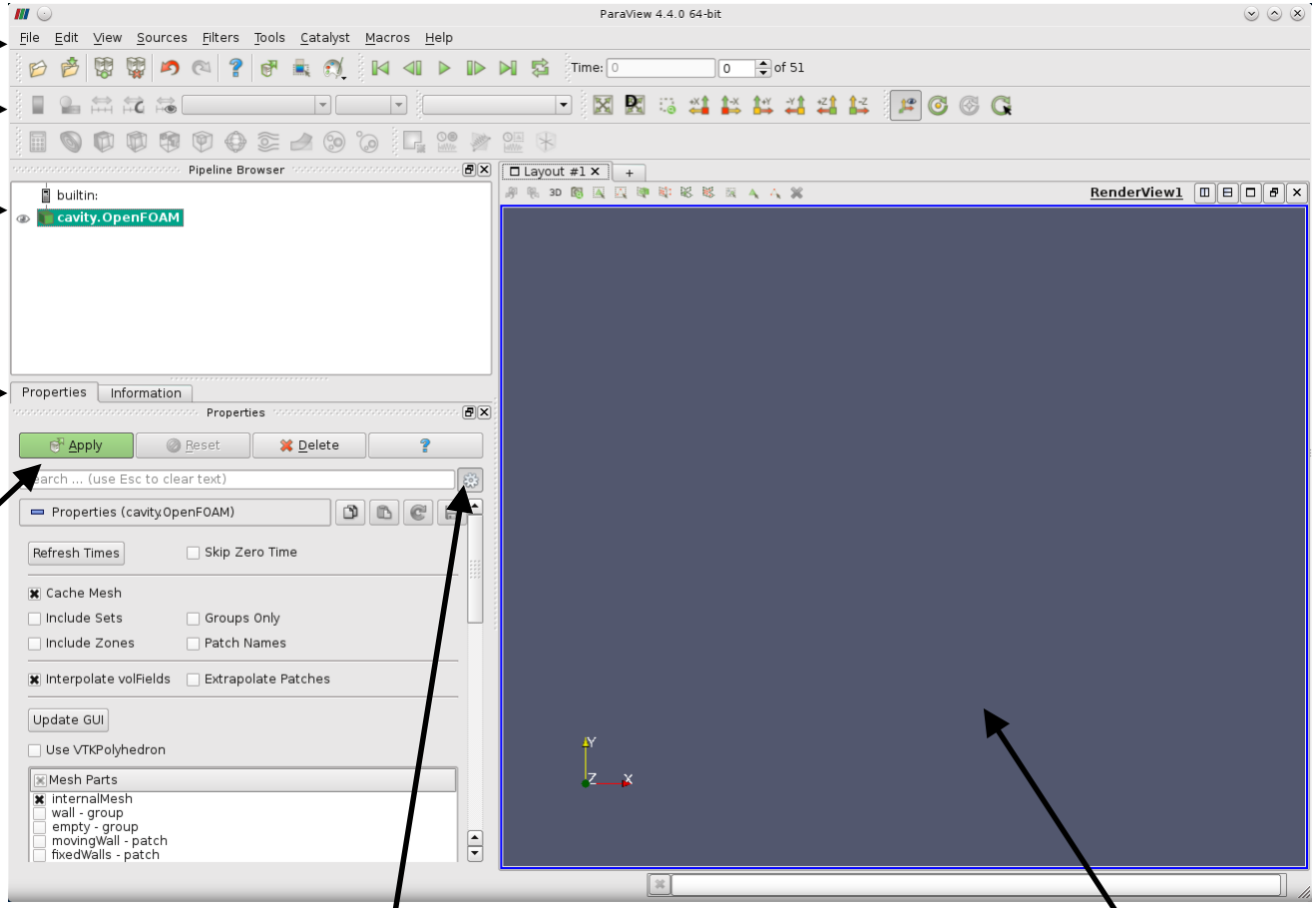
Menu Bar

Toolbars

Pipeline Browser

Properties panel

`Apply button`
Press this button to load the case or to apply a filter



Advanced Toggle

3D View/Canvas

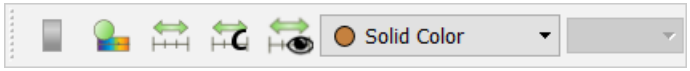
Running my first OpenFOAM® case

Crash introduction to paraFoam – Toolbars

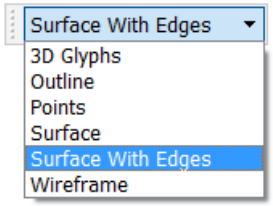
- Main Controls

- VCR Controls (animation controls)

- Current Time Controls



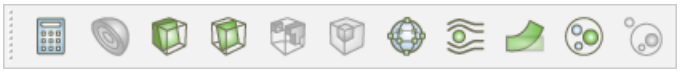
- Active Variable Controls



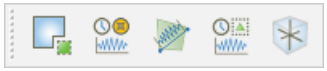
- Representation Toolbar

- Camera Controls (view orientation)

- Center Axes Controls



- Common Filters



- Data Analysis Toolbar

Running my first OpenFOAM® case

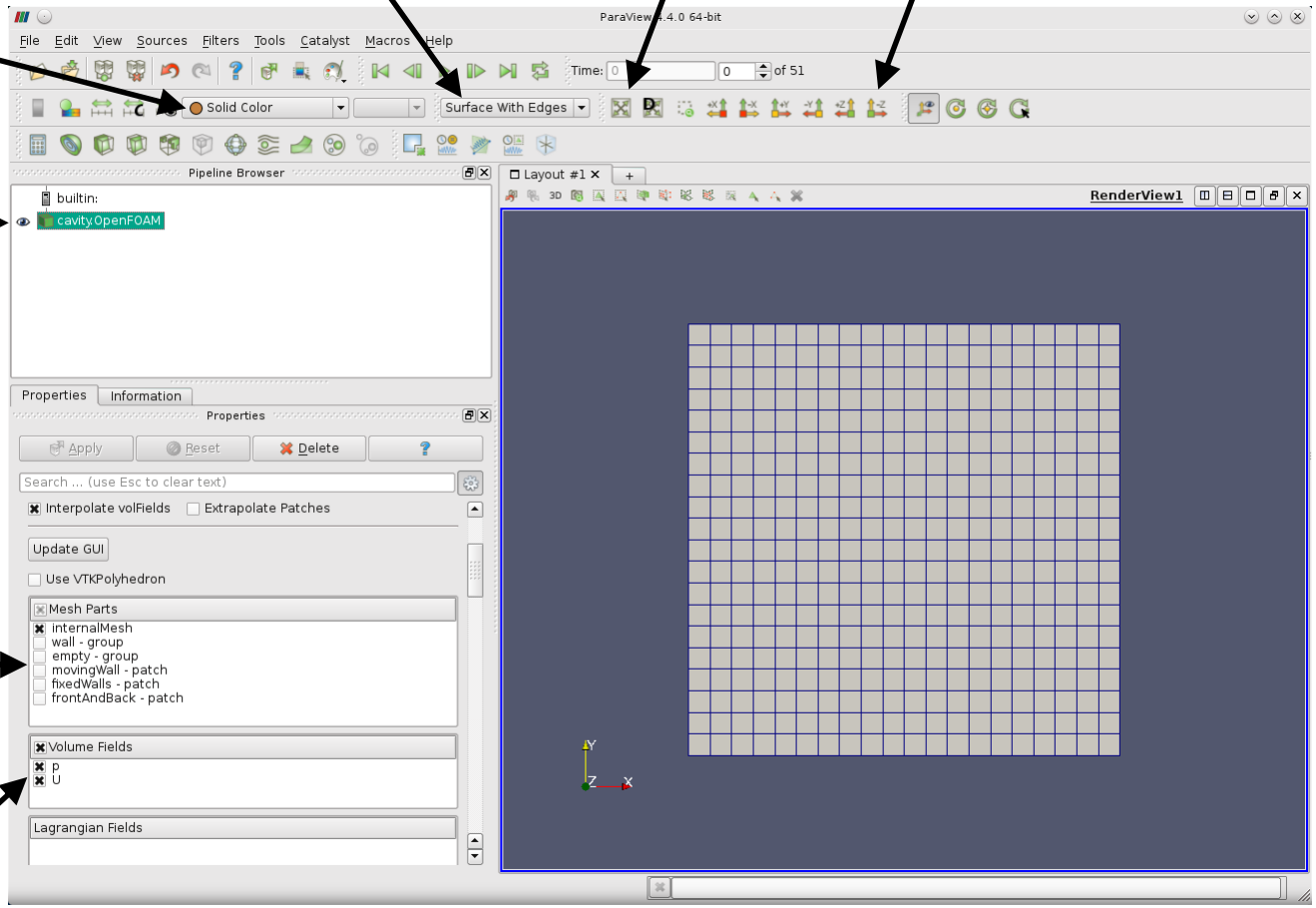
Crash introduction to paraFoam – Mesh visualization

Select `Surface With Edges` in the Representation Toolbar

Fit to screen

Select the -Z view

Select `Solid Color` in the Active Variable Controls



Click on the eyeball in the Pipeline Browser to hide/unhide the object

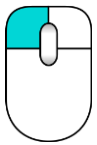
Select mesh parts to visualize. By default it will automatically select `internalMesh`

Select the volume fields to visualize. By default it will select **U** and **p**

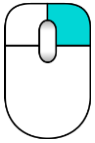
Running my first OpenFOAM® case

Crash introduction to paraFoam – 3D View and mouse interaction

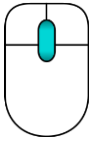
Mouse interaction in the 3D view



Rotate



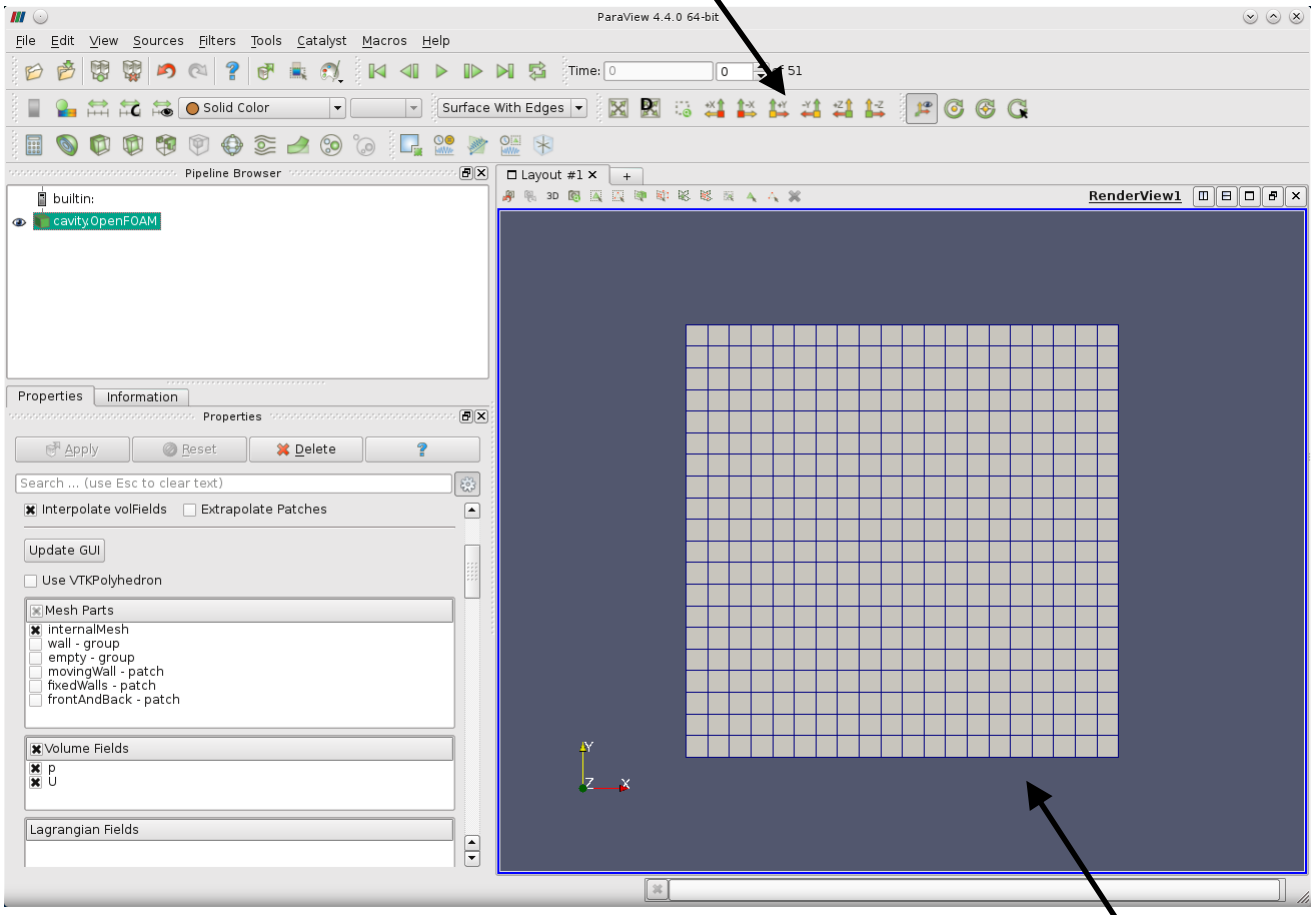
Zoom



Pan

Zoom

Select view orientation in the Camera Controls



3D View/Canvas

Running my first OpenFOAM® case

Crash introduction to paraFoam – Fields visualization

Select `Last Frame` in the VCR Controls

Current Time Controls

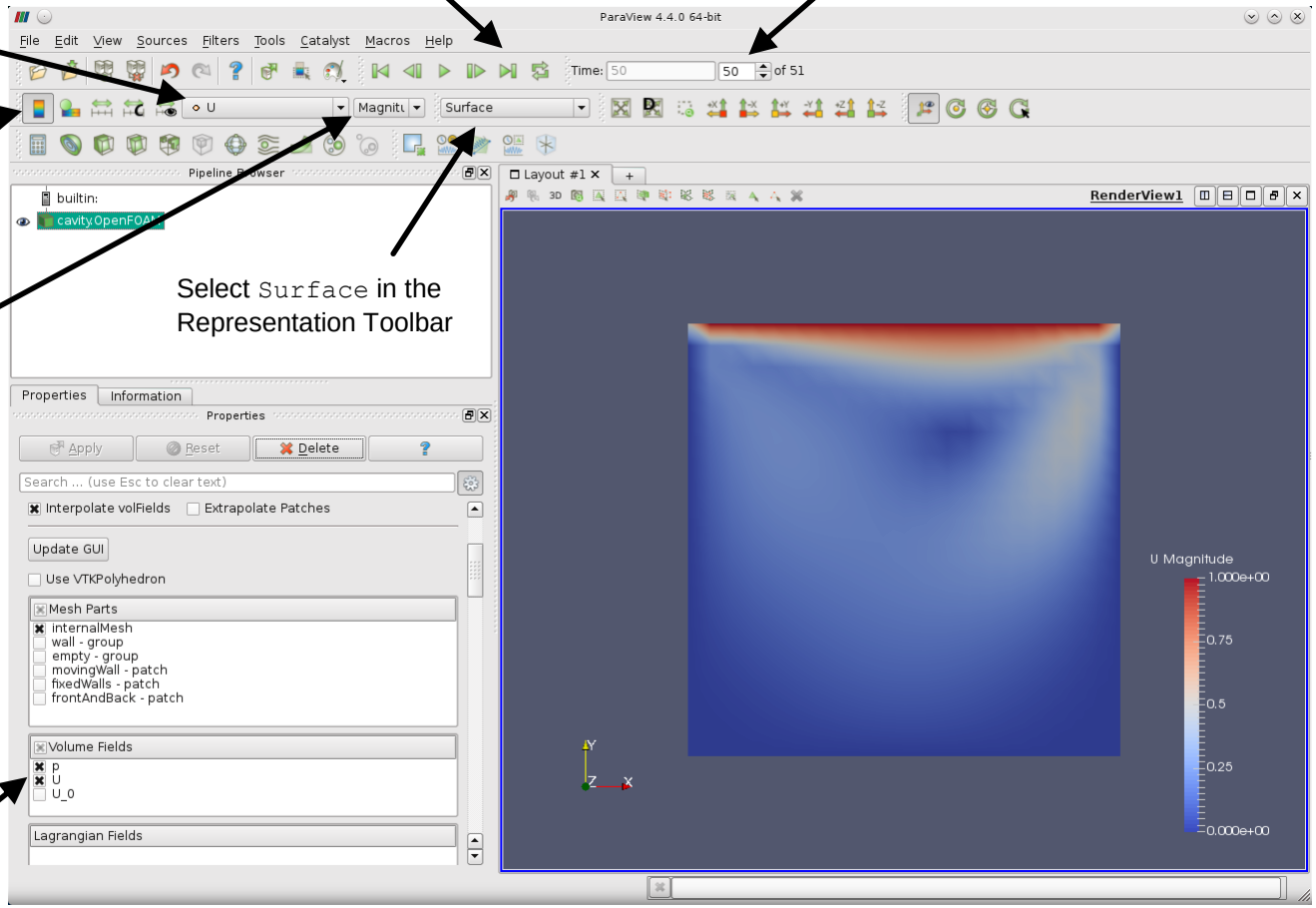
Select **U** in Active Variable Controls

Turn on/off color bar

Select `Magnitude` in the drop down menu

Select `Surface` in the Representation Toolbar

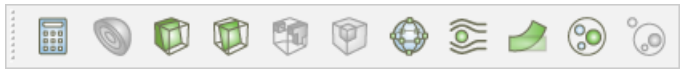
Select volume fields to visualize.
By default it will select **U** and **p.**



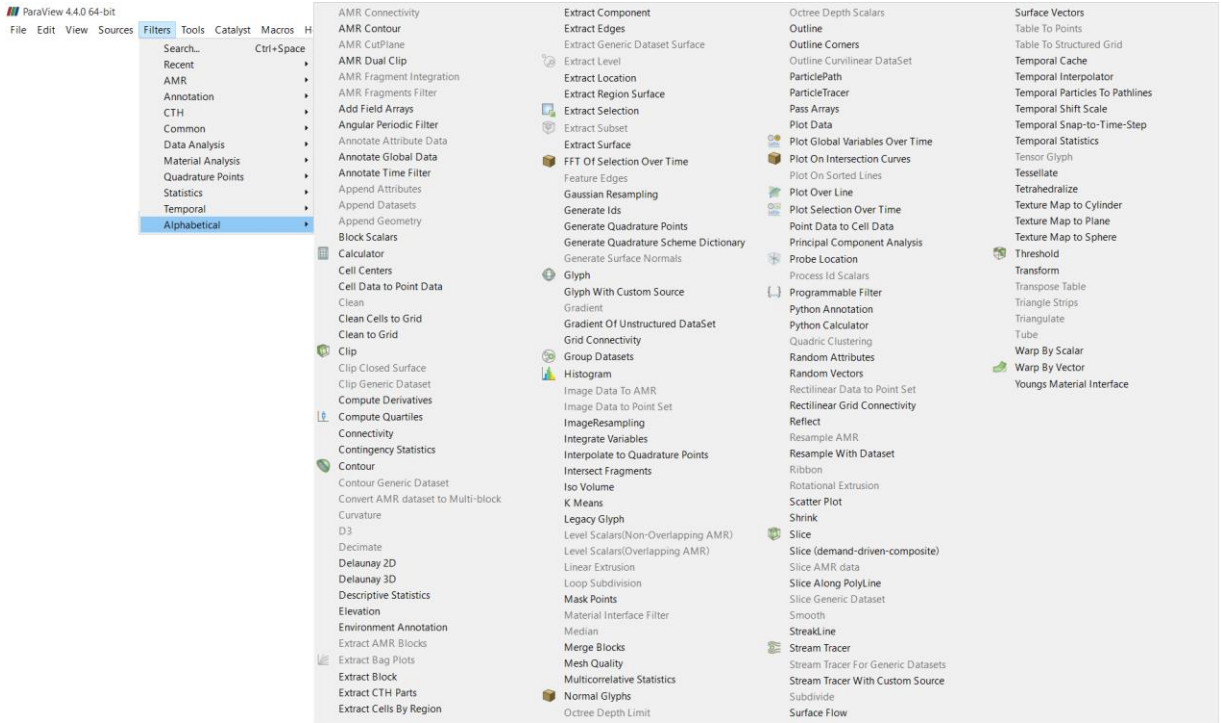
Running my first OpenFOAM® case

Crash introduction to paraFoam – Filters

- Filters are functions that generate, extract or derive features from the input data.
- They are attached to the input data.
- You can access the most commonly used filters from the `Common Filters` toolbar



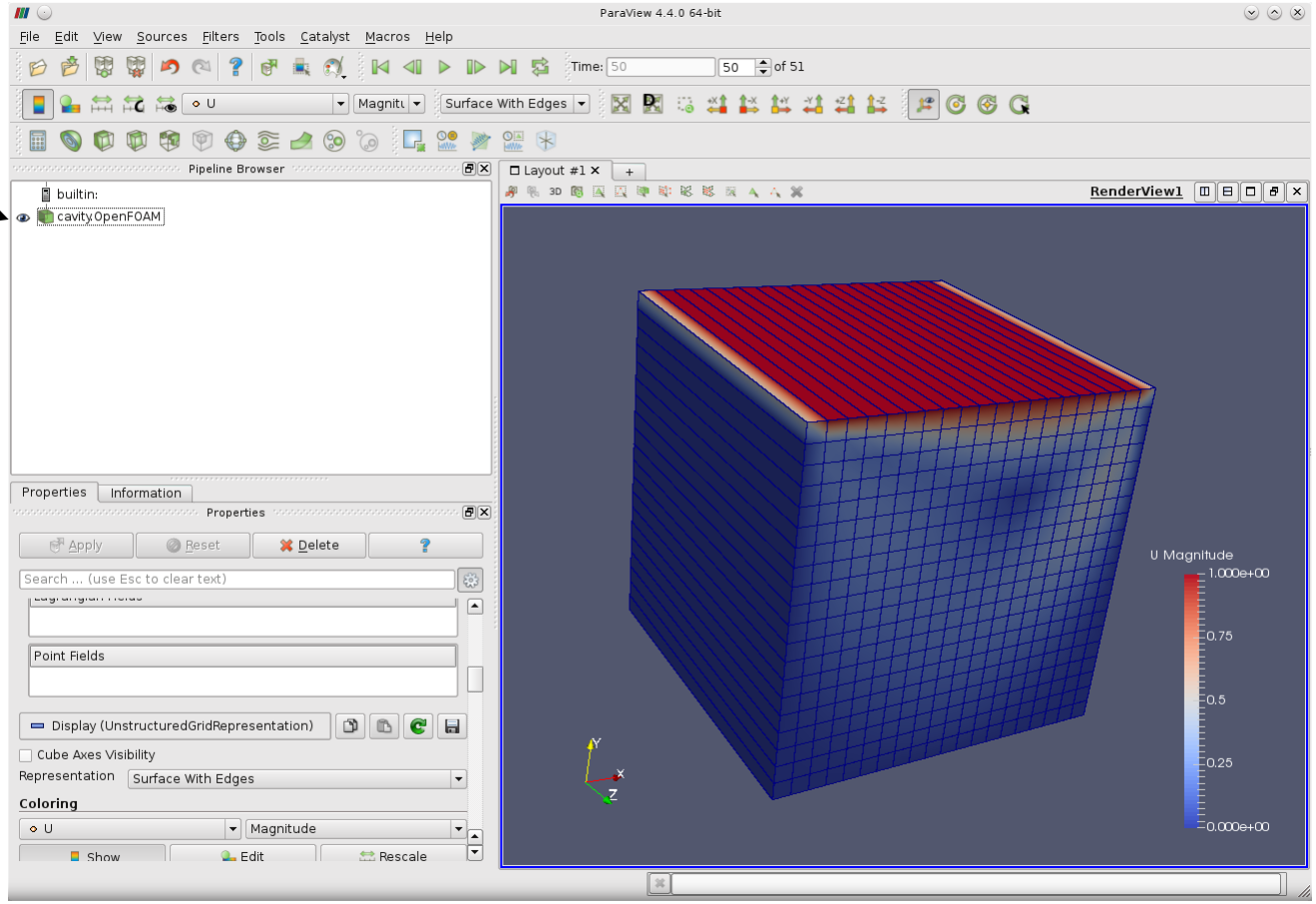
- You can access all the filters from the menu `Filter`.



Running my first OpenFOAM® case

Crash introduction to paraFoam – Filters

Filters are attached to the input data



- Even if the case is 2D, it will be visualized as if it were a 3D case.
- Notice that there is only one cell in the **Z** direction.
- Let us use the slice filter. This filter will create a cut plane.
- Let us create a slice normal to the **Z** direction.

Running my first OpenFOAM® case

Crash introduction to paraFoam – Slice filter

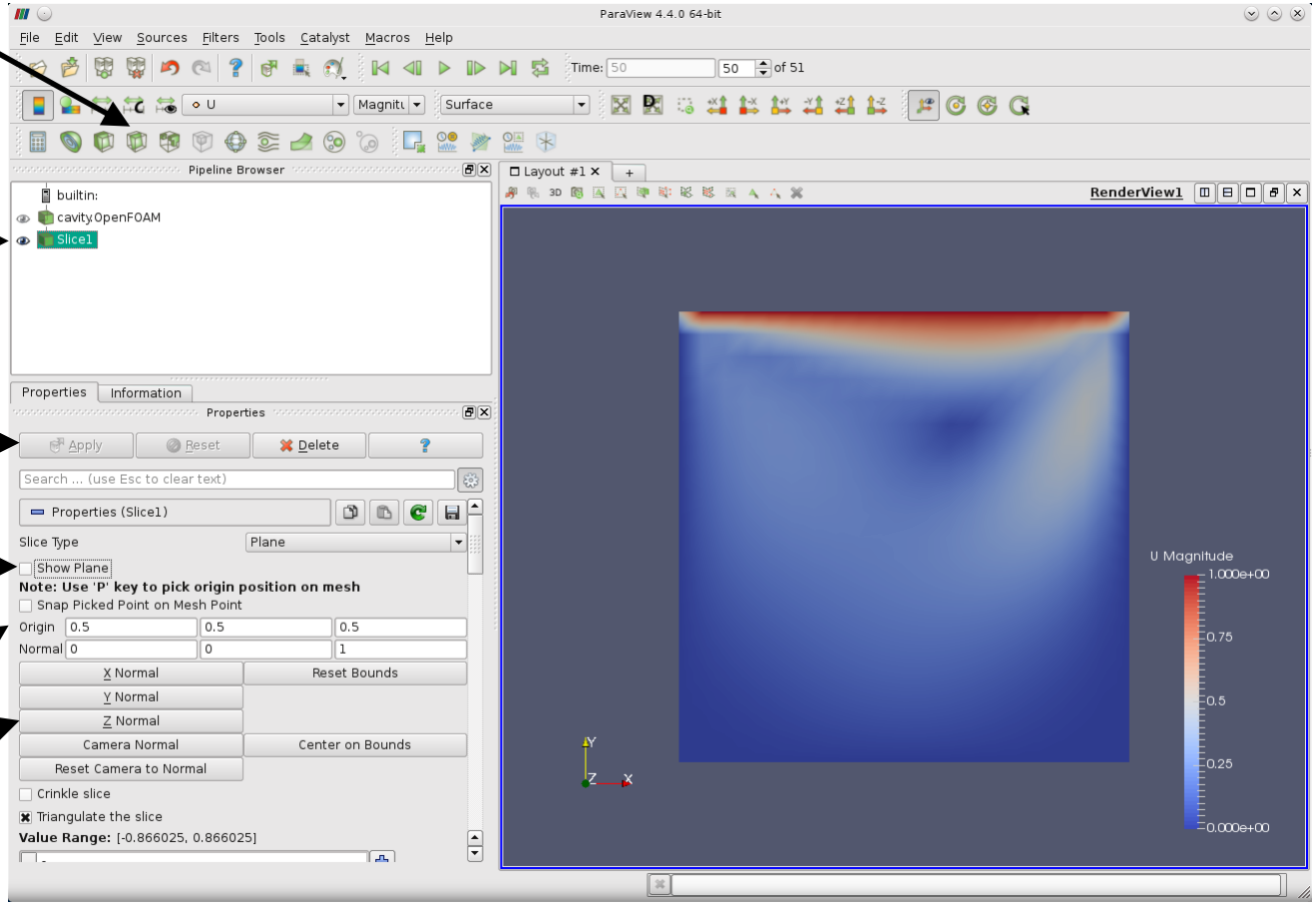
1. Select the `Slice` filter

If you want to erase a filter, right click on it and select `Delete`

4. Press `Apply`

3. Optional - Turn off the option `Show Plane`

2. Select the direction `Z Normal`. Additionally you can choose the origin of the plane (by default is the mid section)



Running my first OpenFOAM® case

Crash introduction to paraFoam – Glyph filter

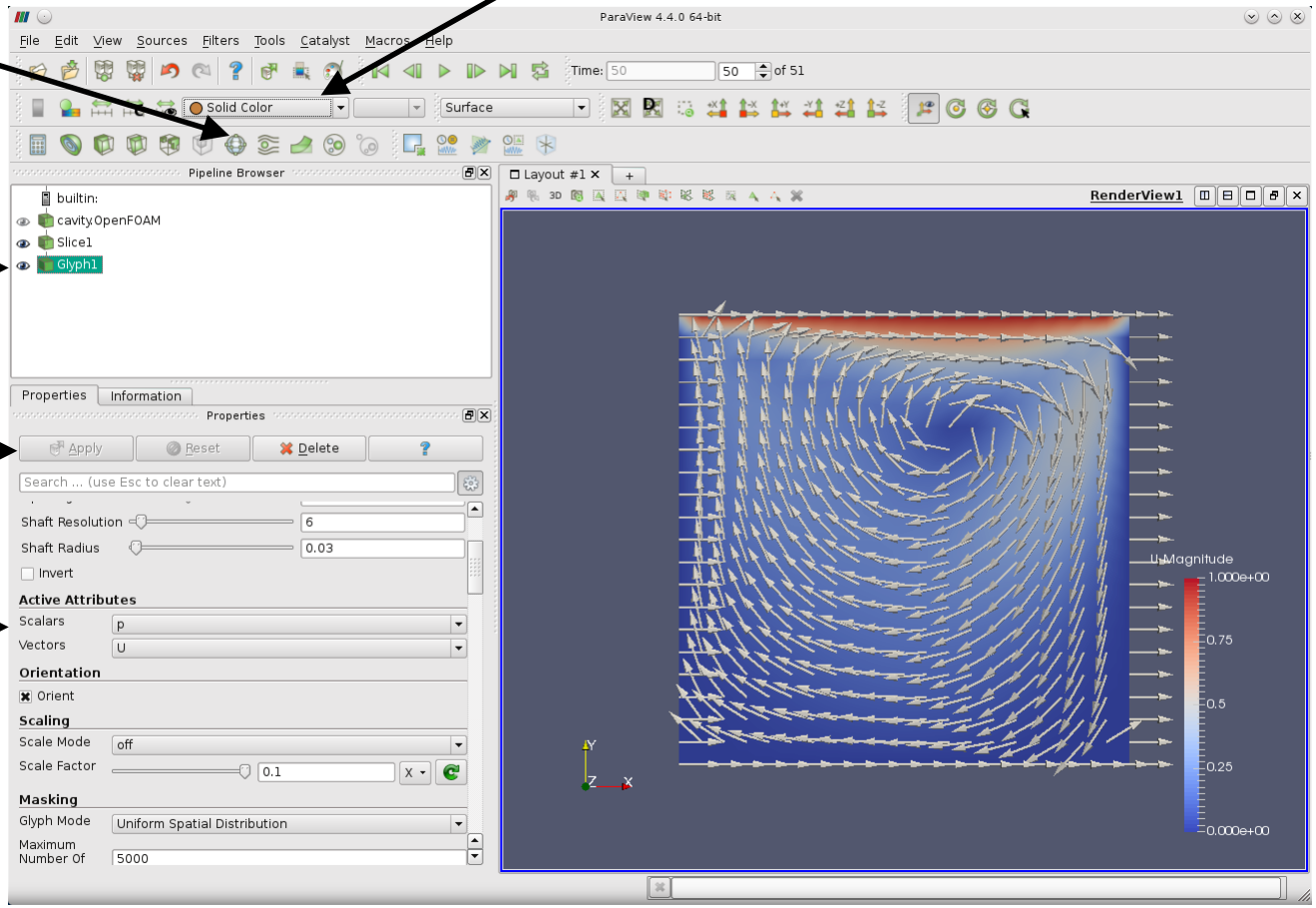
1. Select the `Glyph` filter. This filter will be applied on the `Slice1` filter

4. Color the colors using Solid Color

Notice that the filter `Glyph` was applied on the `Slice1` filter.

3. Press `Apply`

2. Filter options



Notice that the vectors are plotted in the cell vertices. To plot the vectors at the cell centers, use the filter `cell centers` and replot the vectors.

Running my first OpenFOAM® case

Crash introduction to paraFoam – Plot Over Line filter

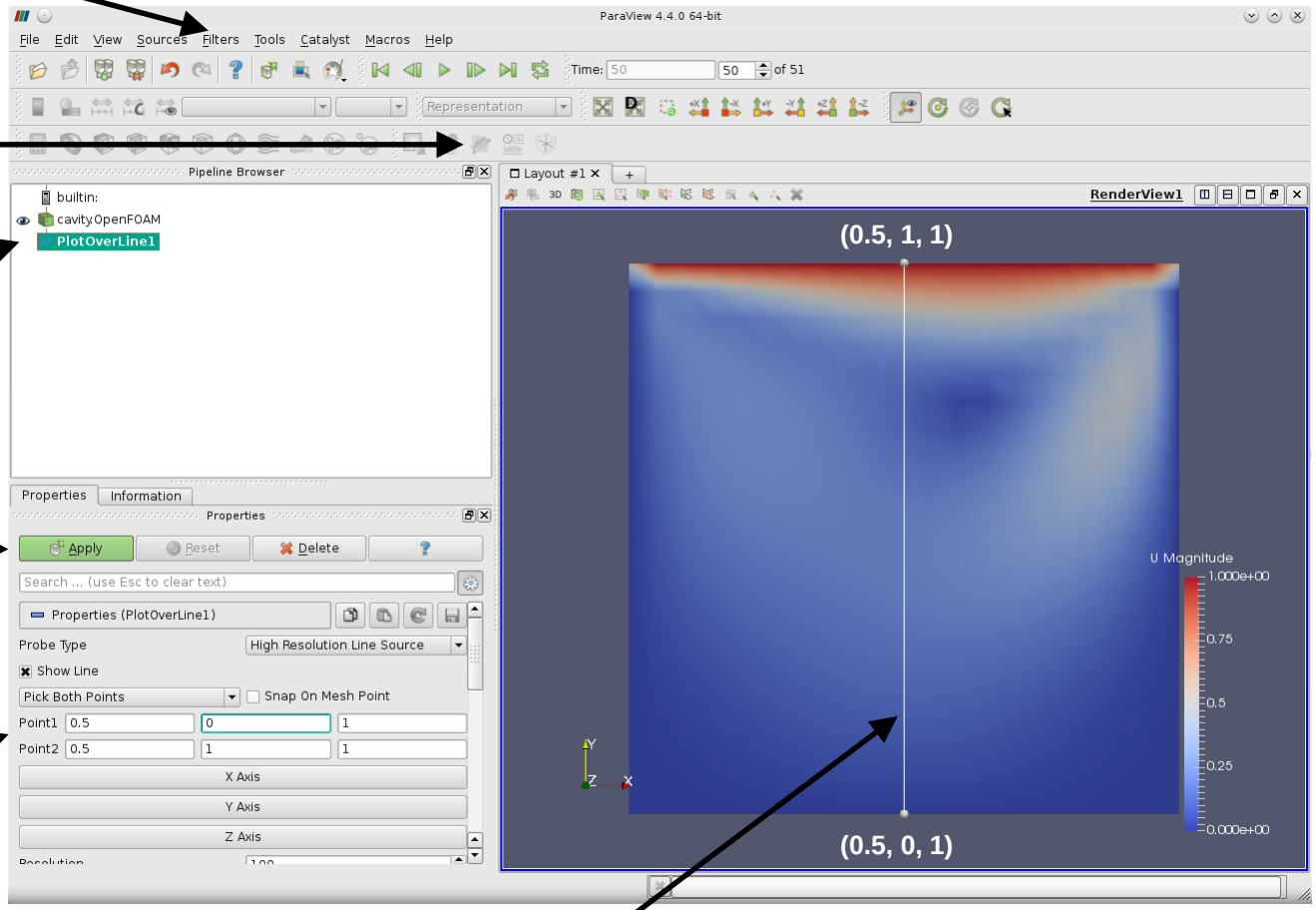
1.a. Select the `Plot Over Line` filter.

1.b. Alternative, you can select `Plot Over Line` filter from the Data Analysis Toolbar

Notice that we are using the filter in a clean `Pipeline`

3. Press `Apply`

2. Enter the coordinates of the line



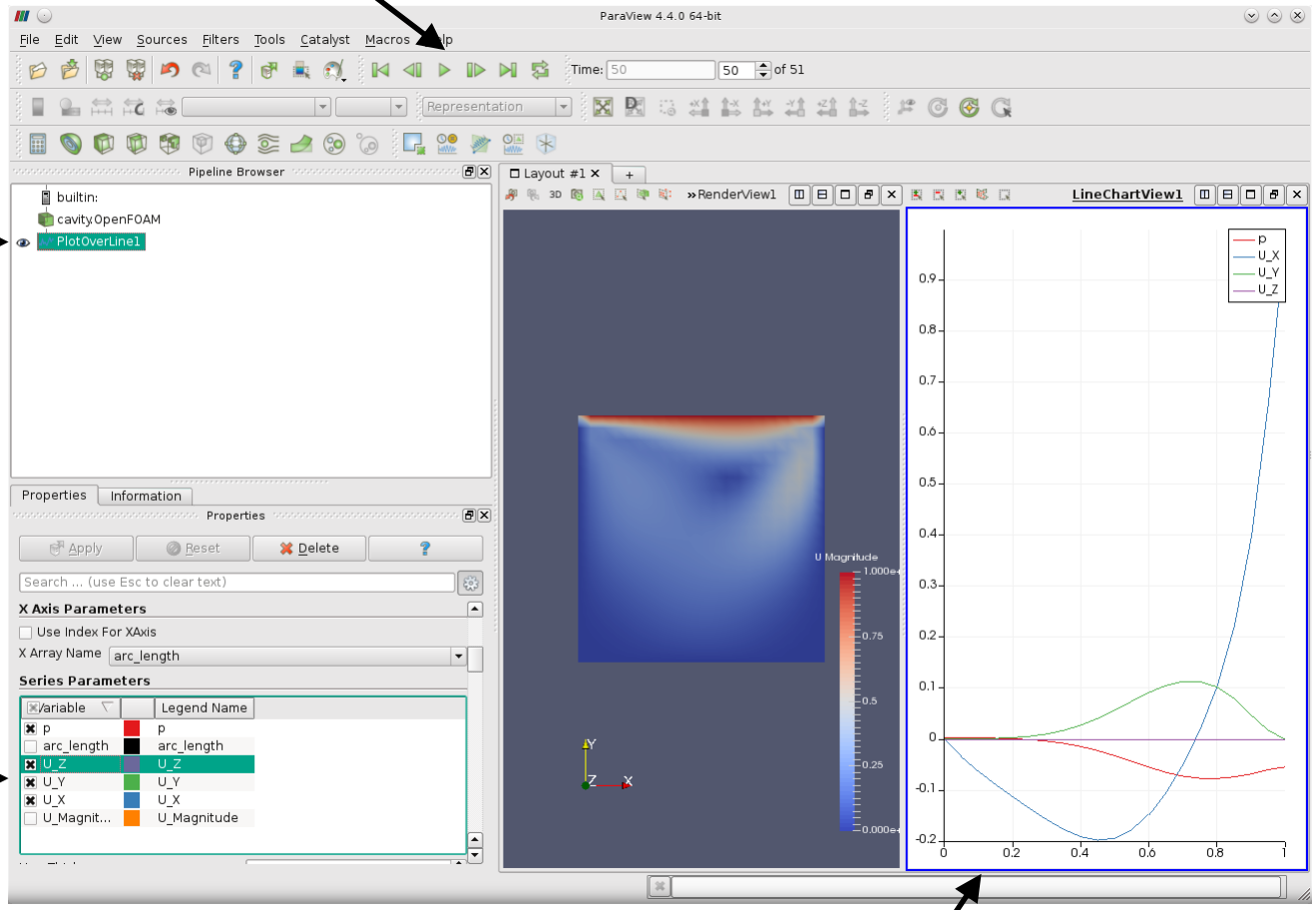
Running my first OpenFOAM® case

Crash introduction to paraFoam – Filters

4. Optional – Use the VCR Control to change the frame.
The line chart view will be updated automatically

3. Optional - To save the sampled data in CSV format, click on the filter. Then click on the `File` menu and select the option `Save Data`

2. Select the variables to plot in the line chart view



1. Click on the line chart view (the blue frame indicates that it is the active view)

Running my first OpenFOAM® case

Running the case blindfold with log files

- In the previous case, we ran the simulation but we did not save the standard output stream (stdout) in a `log` file. We saw the information on-the-fly.
- Our advice is to always save the standard output stream (stdout) in a `log` file.
- It is of interest to always save the `log` as if something goes wrong and you would like to do troubleshooting, you will need this information.
- Also, if you are interested in plotting the residuals you will need the `log` file.
- By the way, if at any point you ask us what went wrong with your simulation, we will ask you for this file. We might also ask for the standard error stream (stderr).

Running my first OpenFOAM® case

Running the case blindfold with log files

- To save a *log* file of the simulation, we proceed as follows:

```
1. $> foamCleanTutorials
2. $> foamCleanPolyMesh
3. $> blockMesh
4. $> checkMesh
5. $> icoFoam > log.icoFoam
6. $> gedit log.icoFoam &
7. $> foamLog log.icoFoam
8. $> gnuplot
```

These steps are optional

Running my first OpenFOAM® case

Running the case blindfold with log files

- In steps 1 and 2 we erase the mesh and all the folders, except for **`0`**, **`constant`** and **`system`**. These scripts come with your OpenFOAM® installation.
- In step 3, we generate the mesh using the meshing tool `blockMesh`.
- In step 4 we check the mesh quality.
- In step 5 we run the simulation. Hereafter, we redirect the standard output to an ascii file with the name *`log.icoFoam`* (it can be any name). However, you will not see the information on the fly. If you do not add the `> log.icoFoam` modifier you will see your standard output on the fly but it will not be saved.
- In step 6, we use gedit to open the file *`log.icoFoam`* (we run gedit in background). Remember, you can use any editor. Also, depending on the size of the log file, opening the file can be very time consuming.
- In step 7, we use the script `foamLog` (distributed with your OpenFOAM® installation), to extract the information inside the file *`log.icoFoam`*. This information is saved in an editable/plottable format in the directory **`logs`**.
- Finally, in step 8 we use gnuplot to plot the information extracted from the *`log.icoFoam`* file.

Running my first OpenFOAM® case

Running the case blindfold with log files

- To plot the information extracted with `foamLog` using gnuplot we can proceed as follows (remember, at this point we are using the gnuplot prompt):

1. `gnuplot> set logscale y`
   Set log scale in the y axis

2. `gnuplot> plot 'logs/p_0' using 1:2 with lines`
   Plot the file p_0 located in the directory logs, use columns 1 and 2 in the file p_0, use lines to output the plot.

3. `gnuplot> plot 'logs/p_0' using 1:2 with lines, 'logs/pFinalRes_0' using 1:2 with lines`
   Here we are plotting to different files. You can concatenate files using comma (,)

4. `gnuplot> reset`
   To reset the scales

5. `gnuplot> plot 'logs/CourantMax_0' u 1:2 w l`
   To plot file CourantMax_0. The letter u is equivalent to using. The letters w l are equivalent to with lines

6. `gnuplot> set logscale y`

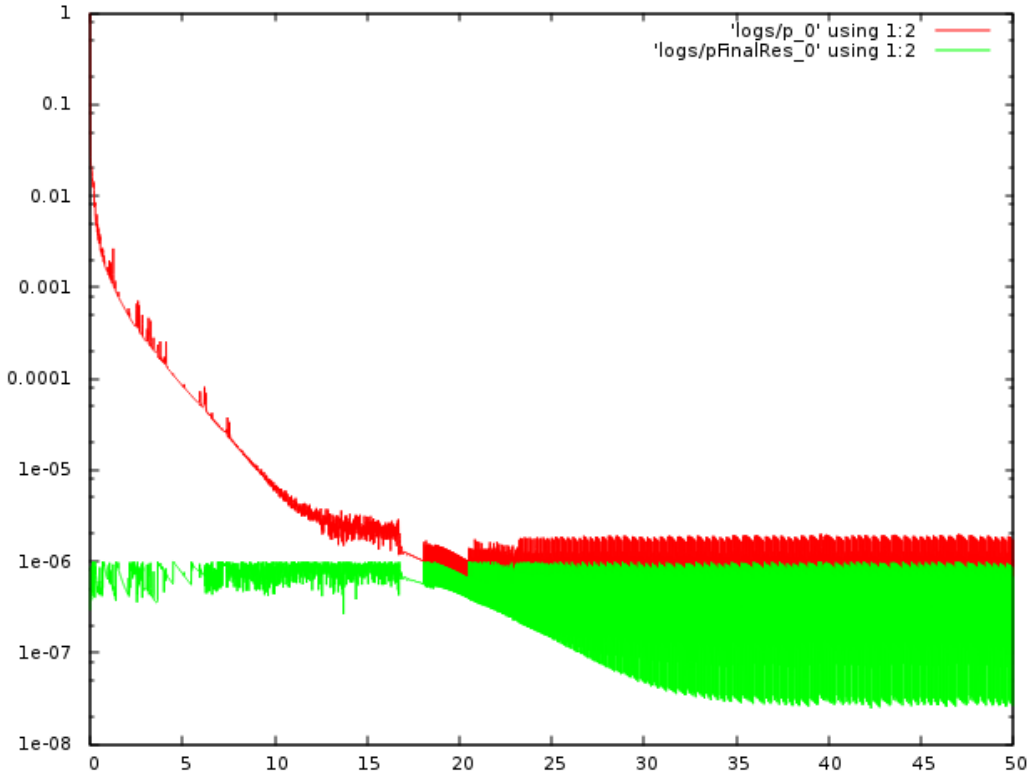
7. `gnuplot> plot [30:50][] 'logs/Ux_0' u 1:2 w l title 'Ux','logs/Uy_0' u 1:2 w l title 'Uy'`
   Set the x range from 30 to 50 and plot tow files and set legend titles

8. `gnuplot> exit`
   To exit gnuplot

Running my first OpenFOAM® case

Running the case blindfold with log files

- The output of step 3 is the following:



- The fact that the initial residuals (red line) are dropping to the same value of the final residuals (monotonic convergence), is a clear indication of a steady behavior.

Running my first OpenFOAM® case

Running the case blindfold with log files

- But what if we want to save the standard output stream (stdout), and monitor the information on the fly?
- To do this and if you are using BASH shell, you can proceed as follows:
  - `$> icoFoam > log.icoFoam  | tail -f log.icoFoam`
- This will redirect your standard output to an ascii file with the name *`log.icoFoam`*. Then using the pipeline operator (|) it will use the `tail` command to show you the information on the fly.
- When the simulation is over, you will notice that the terminal window is blocked. To unblock the terminal window press `ctrl-c`.
- You can also save the standard output in the file *`log.icoFoam`*, open a new terminal window and then use `tail` to output the information on the fly. To do this you can proceed as follow:
  - `$> icoFoam > log.icoFoam`
- Now, in a new terminal window (or tab) and in the same directory where you are running the application, type in the terminal,
  - `$> tail -f log.icoFoam`
- This will use `tail` to show you the information on the fly. Have in mind that you need to be in the case directory.

Running my first OpenFOAM® case

Running the case blindfold with log files

- You can also save the standard output in a *log.icoFoam*, send the job to background and then use `tail` to output the information on the fly. To do this you can proceed as follows,
  - `$> icoFoam > log.icoFoam &`
- Now you can type in the terminal window,
  - `$> tail -f log.icoFoam`

  This will use `tail` to show you the information on the fly. Notice we are still working in the same terminal window or tab.
- If you want to stop the command `tail`, press `ctrl-c`.
- You can also use the Linux command `tee`,
  - `$> icoFoam | tee log.icoFoam`

  This will redirect your standard output to an ascii file with the name *log.icoFoam*, and it will show at the same time the information that is saved in the file.
- If for any reason you do not want to see the standard output stream and you are not interested in saving the *log* file, you can proceed as follows,
  - `$> icoFoam > /dev/null`

Running my first OpenFOAM® case

Running the case blindfold with log files

- You can also save the standard output stream (stdout) and the standard error stream (stderr), as follows
  - `$> icoFoam > log.icoFoam 2>&1 | tail -f log.icoFoam`
- This will redirect the standard output and standard error streams to an ascii file with the name *`log.icoFoam`*. Then using the pipeline operator (|) it will use `tail` to show you the information on the fly.
- Finally, when you are running in a cluster using a job scheduler, you are always interested in saving the *`log`* files in order to monitor the solution. Remember to always redirect the solver standard output and error streams to a *`log`* file.
- To monitor your solution, just login to the cluster, go to the working directory (the directory where you launched the solver) and type
  - `$> tail -f name_of_the_log_file`
- You can login and logout with no problem, everything is being managed by the job scheduler.
- Besides the *`log`* file you are saving, the job scheduler will save all the standard output stream (stdout) and standard error stream (stderr) in a default file. You can also access these files to monitor the solution.

Running my first OpenFOAM® case

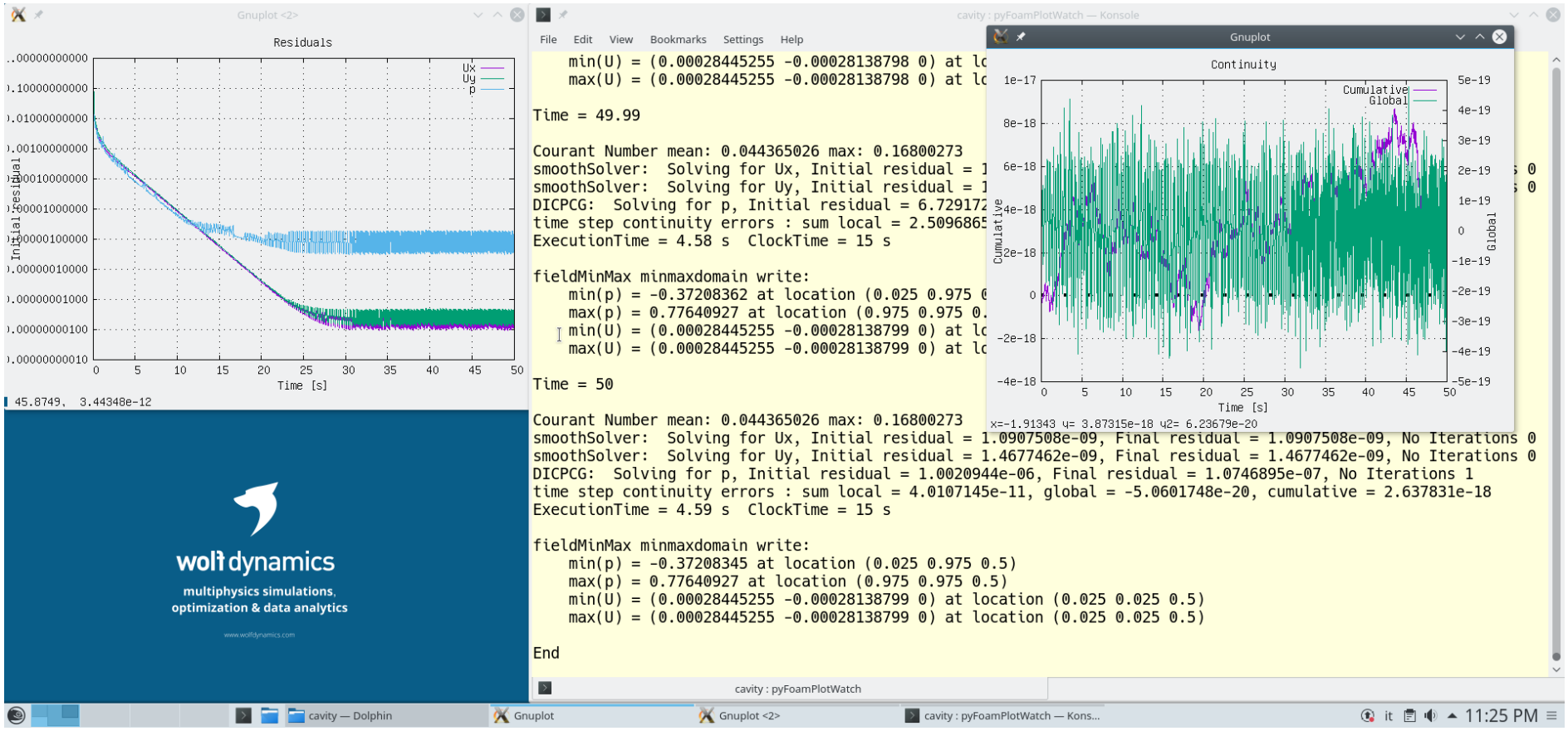
Running the case blindfold with log files and plotting the residuals

- It is also possible to plot the *log* information on the fly.
- The easiest way to do this is by using PyFoam (you will need to install it):
  - `$> pyFoamPlotRunner.py [options] <foamApplication>`
- If you are using our virtual machine or using the lab workstations, you will need to source PyFoam.
- To source PyFoam, type in the terminal:
  - `$> anaconda2` or `$> anaconda3`
- To run this case with `pyFoamPlotRunner.py`, in the terminal type:
  - `$> pyFoamPlotRunner.py icoFoam`
- If you need help or want to know all the options available,
  - `$> pyFoamPlotRunner.py --help`
- If you do not feel comfortable using `pyFoamPlotRunner.py` to run the solver, it is also possible to plot the information saved in the *log* file using PyFoam. To do so you will need to use the utility `pyFoamPlotWatcher.py`. For example, in the terminal type:
  - `$> icoFoam > log.icoFoam &`
  - `$> pyFoamPlotWatcher.py log.icoFoam`
- This will plot the information saved in *log.icoFoam*.
- You can also use `pyFoamPlotWatcher.py` to plot the information saved in an old *log* file.

Running my first OpenFOAM® case

Running the case blindfold with log files and plotting the residuals

- This is a screenshot on my computer. In this case, `pyFoamPlotRunner` is plotting the initial residuals and continuity errors on the fly.



Running my first OpenFOAM® case

Stopping the simulation

- Your simulation will automatically stop at the time value you set using the keyword **endTime** in the *controlDict* dictionary.

endTime 50;

- If for any reason you want to stop your simulation before reaching the value set by the keyword **endTime**, you can change this value to a number lower than the current simulation time (you can use 0 for instance). This will stop your simulation, but it will not save your last time-step or iteration, so be careful.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  4.x                                   |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      controlDict;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  application     icoFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          endTime;
24
25  endTime         50;
```

Running my first OpenFOAM® case

Stopping the simulation

- If you want to stop the simulation and save the solution, in the *controlDict* dictionary made the following modification,

stopAt writeNow;

This will stop your simulation and will save the current time-step or iteration.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  4.x                                   |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      controlDict;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  application     icoFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          writeNow;
24
25  endTime         50;
```

Running my first OpenFOAM® case

Stopping the simulation

- The previous modifications can be done on-the-fly, but you will need to set the keyword **runTimeModifiable** to **true** in the *controlDict* dictionary.
- By setting the keyword **runTimeModifiable** to **true**, you will be able to modify most of the dictionaries on-the-fly.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  4.x                                   |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      controlDict;
14  }

44
45  runTimeModifiable true;
46
```

Running my first OpenFOAM® case

Stopping the simulation

- You can also kill the process. For instance, if you did not launch the solver in background, go to its terminal window and press `ctrl-c`. This will stop your simulation, but it will not save your last time-step or iteration, so be careful.
- If you launched the solver in background, just identify the process `id` using `top` or `htop` (or any other process manager) and terminate the associated process. Again, this will not save your last time-step or iteration.
- To identify the process `id` of the OpenFOAM® solver or utility, just read screen. At the beginning of the output screen, you will find the process `id` number.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.x-e964d879e2b3
Exec   : icoFoam
Date   : Mar 11 2017
Time   : 23:21:50
Host   : "linux-ifxc"
PID    : 3100
Case   : /home/joegi/my_cases_course/4x/101OF/cavity
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

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

Process `id` number

Running my first OpenFOAM® case

Stopping the simulation

- When working locally, we usually proceed in this way:
  - `$> icoFoam > log.icofoam | tail -f log.icofoam`

This will run the solver `icoFoam` (by the way, this works for any solver or utility), it will save the standard output stream in the file *`log.icofoam`* and will show the solver output on the fly.

- If at any moment we want to stop the simulation, and we are not interested in saving the last time-step, we press `ctrl-c`.

- If we are interested in saving the last time step, we modify the *`controlDict`* dictionary and add the following keyword

  stopAt writeNow;

- Remember, this modification can be done on the fly. However, you will need to set the keyword **runTimeModifiable** to **yes** in the *`controlDict`* dictionary.

Running my first OpenFOAM® case

Cleaning the case folder

- If you want to erase the mesh and the solution in the current case folder, you can type in the terminal,

```
$> foamCleanTutorials
```

If you are running in parallel, this will also erase the **`processorN`** directories. We will talk about running in parallel later.

- If you are looking to only erase the mesh, you can type in the terminal,

```
$> foamCleanPolyMesh
```

- If you are only interested in erasing the saved solutions, in the terminal type,

```
$> foamListTimes -rm
```

- If you are running in parallel and you want to erase the solution saved in the **`processorN`** directories, type in the terminal,

```
$> foamListTimes -rm -processor
```

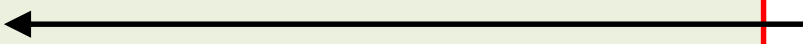
A deeper view to my first OpenFOAM® case setup

- We will take a close look at what we did by looking at the case files.
- The case directory originally contains the following sub-directories: **`0`**, **`constant`**, and **`system`**. After running `icoFoam` it also contains the time step directories **`1`**, **`2`**, **`3`**, **`...`**, **`48`**, **`49`**, **`50`**, the post-processing directory **`postProcessing`**, and the *`log.icoFoam`* file (if you chose to redirect the standard output stream).
  - The time step directories contain the values of all the variables at those time steps (the solution). The **`0`** directory is thus the initial condition and boundary conditions.
  - The **`constant`** directory contains the mesh and dictionaries for thermophysical, turbulence models and advanced physical models.
  - The **`system`** directory contains settings for the run, discretization schemes and solution procedures.
  - The **`postProcessing`** directory contains the information related to the **functionObjects** (we are going to address **functionObjects** later).
- The `icoFoam` solver reads these files and runs the case according to those settings.

A deeper view to my first OpenFOAM® case setup

- Before continuing, we want to point out the following:
  - Each dictionary file in the case directory has a header.
  - Lines 1-7 are commented.
  - You should always keep lines 8 to 14, if not, OpenFOAM® will complain.
  - According to the dictionary you are using, the **class** keyword (line 12) will be different. We are going to talk about this later on.
  - From now on and unless it is strictly necessary, we will not show the header when listing the dictionaries files.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  4.x                                   |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      controlDict;
14  }
```



A deeper view to my first OpenFOAM® case setup

Let us explore the case directory

A deeper view to my first OpenFOAM® case setup

The **`constant`** directory

(and by the way, open each file and go thru its content)

- In this directory you will find the sub-directory **`polyMesh`** and the dictionary file *`transportProperties`*.
- The *`transportProperties`* file is a dictionary for the dimensioned scalar **nu**, or the kinematic viscosity.

```
17    nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;       //Re 100
18    //nu              nu [ 0 2 -1 0 0 0 0 ] 0.001;     //Re 1000
```

- Notice that line 18 is commented.
- The values between square bracket are the units.
- OpenFOAM® is fully dimensional. You need to define the dimensions for each field dictionary and physical properties defined.

- Your dimensions shall be consistent.

A deeper view to my first OpenFOAM® case setup

Dimensions in OpenFOAM® (metric system)

No.	Property	Unit	Symbol
1	Mass	Kilogram	**kg**
2	Length	meters	**m**
3	Time	second	**s**
4	Temperature	Kelvin	**K**
5	Quantity	moles	**mol**
6	Current	ampere	**A**
7	Luminuous intensity	candela	**cd**

[1 (kg), 2 (m), 3 (s), 4 (K), 5 (mol), 6 (A), 7 (cd)]

A deeper view to my first OpenFOAM® case setup

The **constant** directory

(and by the way, open each file and go thru its content)

- Therefore, the dimensioned scalar **nu** or the kinematic viscosity,

```
17    nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;
```

has the following units

```
[ 0 m^2 s^-1 0 0 0 0 ]
```

Which is equivalent to

$$\nu = 0.01 \frac{m^2}{s}$$

A deeper view to my first OpenFOAM® case setup

The `constant` directory

(and by the way, open each file and go thru its content)

- In this case, as we are working with an incompressible flow, we only need to define the kinematic viscosity.

$$\nu = \frac{\mu}{\rho}$$

- Later on, we will ask you to change the Reynolds number, to do so you can change the value of **nu**. Remember,

$$Re = \frac{\rho \times U \times L}{\mu} = \frac{U \times L}{\nu}$$

- You can also change the free stream velocity U or the reference length L.

A deeper view to my first OpenFOAM® case setup

The **`constant`** directory

(and by the way, open each file and go thru its content)

- Depending on the physics involved and models used, you will need to define more variables in the dictionary *`transportProperties`*.
- For instance, for a multiphase case you will need to define the density **rho** and kinematic viscosity **nu** for each single phase. You will also need to define the surface tension σ.
- Also, depending of your physical model, you will find more dictionaries in the constant directory.
- For example, if you need to set gravity, you will need to create the dictionary *`g`*.
- If you work with compressible flows you will need to define the dynamic viscosity **mu**, and many other physical properties in the dictionary *`thermophysicalProperties`*.
- As we are not dealing with compressible flows (for the moment), we are not going into details.

A deeper view to my first OpenFOAM® case setup

The **`constant/polyMesh`** directory

(and by the way, open each file and go thru its content)

- In this case, the **`polyMesh`** directory is initially empty. After generating the mesh, it will contain the mesh in OpenFOAM® format.
- To generate the mesh in this case, we use the utility `blockMesh`. This utility reads the dictionary *`blockMeshDict`* located in the **`system`** folder.
- We will now take a quick look at the *`blockMeshDict`* dictionary in order to understand what we have done. Do not worry, we are going to revisit this dictionary during the meshing session.
- Go to the directory **`system`** and open *`blockMeshDict`* dictionary with your favorite text editor, we will use gedit.

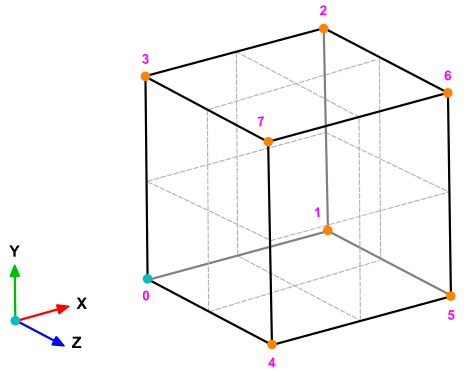
A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary first at all contains a list with a number of vertices:

```
17      convertToMeters 1;
18
19      xmin 0;
20      xmax 1;
21      ymin 0;
22      ymax 1;
23      zmin 0;
24      zmax 1;
25
26      xcells 20;
27      ycells 20;
28      zcells 1;
29
37      vertices
38      (
39          ($xmin   $ymin   $zmin)       //vertex 0
40          ($xmax   $ymin   $zmin)       //vertex 1
41          ($xmax   $ymax   $zmin)       //vertex 2
42          ($xmin   $ymax   $zmin)       //vertex 3
43          ($xmin   $ymin   $zmax)       //vertex 4
44          ($xmax   $ymin   $zmax)       //vertex 5
45          ($xmax   $ymax   $zmax)       //vertex 6
46          ($xmin   $ymax   $zmax)       //vertex 7
47
48      /*
49          (0 0 0)
50          (1 0 0)
51          (1 1 0)
52          (0 1 0)
53          (0 0 0.1)
54          (1 0 0.1)
55          (1 1 0.1)
56          (0 1 0.1)
57      */
58      );
```

- The keyword **convertToMeters** (line 17), is a scaling factor. In this case we do not scale the dimensions.
- In the section vertices (lines 37-58), we define the vertices coordinates of the geometry. In this case, there are eight vertices defining the geometry. OpenFOAM® always uses 3D meshes, even if the simulation is 2D.
- We can directly define the vertex coordinates in the section vertices (commented lines 49-56), or we can use macro syntax.
- Using macro syntax we first define a variable and its value (lines 19-24), and then we can use them by adding the symbol **$** to the variable name (lines 39-46).
- In lines 26-28, we define a set of variables that will be used at a later time.
- Finally, notice that the vertex numbering starts from 0 (as the counters in c++). This numbering applies for blocks as well.



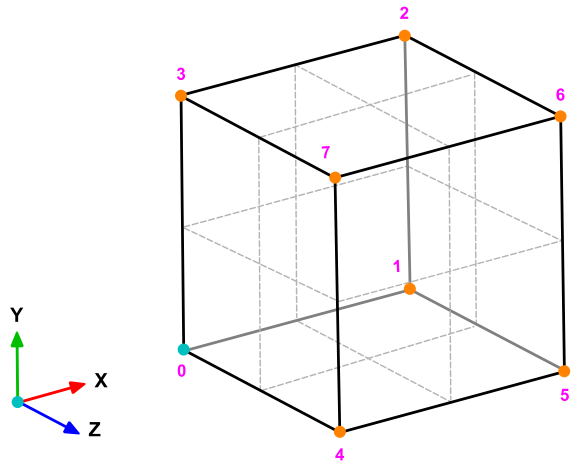
A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary then defines a block from the vertices:

```
60      blocks
61      (
62          hex (0 1 2 3 4 5 6 7) ($xcells $ycells $zcells) simpleGrading (1 1 1)
63      );
```

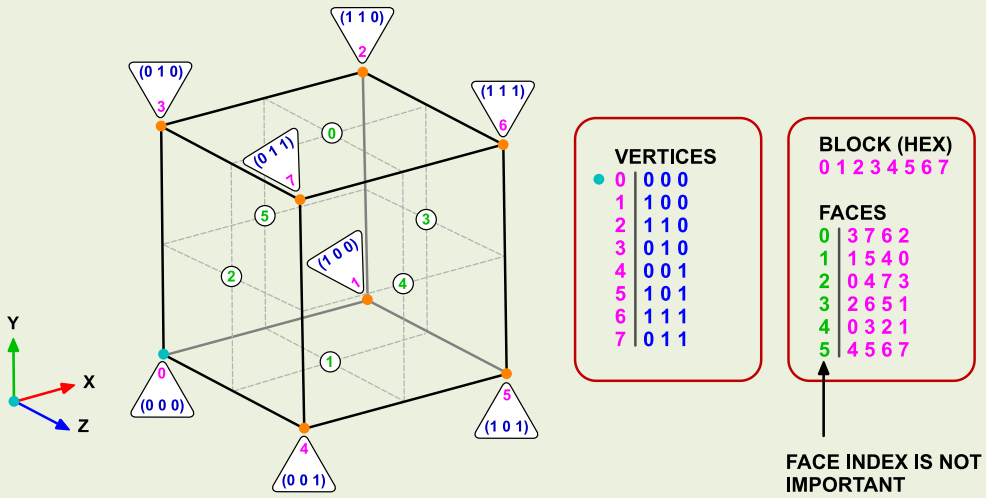
- In lines 60-63, we define the block topology, **hex** means that it is a structured hexahedral block. In this case, we are generating a rectangular mesh.
- **(0 1 2 3 4 5 6 7)** are the vertices used to define the block topology (and yes, the order is important). Each hex block is defined by eight vertices, in sequential order. Where the first vertex in the list represents the origin of the coordinate system.
- **($xcells $ycells $zcells)** is the number of mesh cells in each direction (**X Y Z**). Notice that we are using macro syntax, which is equivalent to **(20 20 1)**.
- **simpleGrading (1 1 1)** is the expansion ratio or mesh stretching in each direction (**X Y Z**), in this case the mesh is uniform.



A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary then defines a block from the vertices:



- Let us talk about the block ordering **hex (0 1 2 3 4 5 6 7)**, which is extremely important.
- **hex** blocks are defined by eight vertices in sequential order. Where the first vertex in the list represents the origin of the coordinate system (vertex **0** in this case).
- Starting from this vertex, we construct the block topology. So in this case, the first part of the block is made up by vertices **0 1 2 3** and the second part of the block is made up by vertices **4 5 6 7**.
- In this case, the vertices are ordered in such a way that if we look at the screen/paper (-**Z** direction), the vertices rotate counter-clockwise.

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary also defines edges:

```
65    edges
66    (
67        //arc 0 1 (0.5 -0.1 0)
68        //arc 4 5 (0.5 -0.1 1)
69    );
```

- Edges, are constructed from the vertices definition.
- Each edge joining two vertex is assumed to be straight by default.
- The user can specified any edge to be curved by entries in the block edges.
- Possible options are: arc, spline, polyline, BSpline, line.
- For example, to define an arc we first define the vertices to be connected to form an edge and then we give an interpolation point.
- In this case and as we do not specified anything, all edges are assumed to be straight lines.
- By the way, lines 67 and 68 are commented.

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary also defines the boundary patches:

```
71      boundary
72      (
73          movingWall
74          {
75              type wall;
76              faces
77              (
78                  (3 7 6 2)
79              );
80          }
81          fixedWalls
82          {
83              type wall;
84              faces
85              (
86                  (0 4 7 3)
87                  (2 6 5 1)
88                  (1 5 4 0)
89              );
90          }
91          frontAndBack
92          {
93              type empty;
94              faces
95              (
96                  (0 3 2 1)
97                  (4 5 6 7)
98              );
99          }
100     );
```

Name

Type

Connectivity

- In the section **boundary**, we define all the surface patches where we want to apply boundary conditions.
- This step is of paramount importance, because if we do not define the surface patches we will not be able to apply the boundary conditions.
- For example:
  - In line 73 we define the patch name **movingWall** (the name is given by the user).
  - In line 75 we give a **base type** to the surface patch. In this case **wall** (do not worry we are going to talk about this later on).
  - In line 78 we give the connectivity list of the vertices that made up the surface patch or face, that is, **(3 7 6 2)**. Have in mind that the vertices need to be neighbors and it does not matter if the ordering is clockwise or counter clockwise.
- Remember, faces are defined by a list of 4 vertex numbers, e.g., **(3 7 6 2)**.

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary also defines the boundary patches:

```
71      boundary
72      (
73          movingWall
74          {
75              type wall;
76              faces
77              (
78                  (3 7 6 2)
79              );
80          }
81          fixedWalls
82          {
83              type wall;
84              faces
85              (
86                  (0 4 7 3)
87                  (2 6 5 1)
88                  (1 5 4 0)
89              );
90          }
91          frontAndBack
92          {
93              type empty;
94              faces
95              (
96                  (0 3 2 1)
97                  (4 5 6 7)
98              );
99          }
100     );
```

- We can also group many faces into one patch, for example, take a look at lines 86-88. In this case:
  - The name of the patch is **fixedWalls**
  - The **base type** is **wall**
  - The surface patch is made up by the faces **(0 4 7 3)**, **(2 6 5 1),** and **(1 5 4 0)**.
- The base type **empty** (line 93) is used to define a 2D mesh. There is only one cell in the direction of the vector connecting faces **(0 3 2 1)** and **(4 5 6 7)**.
- If you do not define a boundary patch, it will be automatically grouped in the patch **defaultFaces** of type **empty**.
- The name and type of the surface patch can be changed outside of the *blockMeshDict*, we are going to address this later.

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

The *blockMeshDict* dictionary also defines how to merge multiple blocks:

```
102     mergePatchPairs
103     (
104         //(interface1 interface2)
105     );
```

- A mesh can be created using more than 1 block. To do so we proceed in the same way, the only difference is that we need to connect the blocks.
- We can merge blocks in the section **mergePatchPairs** (lines 102-105). This requires that the block patches to be merged are first defined in the **boundary** list (**interface1** and **interface2** in this case), `blockMesh` then connect the two blocks.
- Line 104 is commented. We are not merging blocks, we will talk about this during the meshing session.

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

- To sum up, the *blockMeshDict* dictionary generates a single block with:
  - **X**/**Y**/**Z** dimensions: **1.0**/**1.0**/**1.0**
  - Cells in the **X**, **Y** and **Z** directions: **20** x **20** x **1** cells.
  - One single **hex** block with straight lines.
  - Patch type **wall** and patch name **fixedWalls** at three sides.
  - Patch type **wall** and patch name **movingWall** at one side.
  - Patch type **empty** and patch name **frontAndBack** patch at two sides.

- If you are interested in visualizing the actual block topology, you can use `paraFoam` as follows,
  - `$> paraFoam -block`

A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

- As you can see, the *blockMeshDict* dictionary can be really tricky.
- If you deal with really easy geometries (rectangles, cylinders, and so on), then you can use `blockMesh` to do the meshing (and by the way you are the luckiest guy in the world), but this is the exception rather than the rule.
- When using `snappyHexMesh`, (a body fitted mesher that comes with OpenFOAM®) you will need to generate a background mesh using `blockMesh`. We are going to deal with this later on.
- Our best advice is to create a template and reuse it until the end of the world.
- Also, take advantage of macro syntax for parametrization, and **#calc** syntax to perform inline calculations (lines 30-35 in the *blockMeshDict* dictionary we just studied).
- We are going to deal with **#codeStream** syntax and **#calc** syntax during the programming session.

The mesher blockMesh has many more features that we did not address in this short overview. Refer to the User Guide for more Information.

A deeper view to my first OpenFOAM® case setup

The `constant/polyMesh/boundary` dictionary

- First at all, this file is automatically generated after you create the mesh using `blockMesh` or `snappyHexMesh`, or when you convert the mesh from a third-party format.
- In this file, the geometrical information related to the **base type** patch of each boundary of the domain is specified.
- The **base type** boundary condition is the actual surface patch where we are going to apply a **primitive type** boundary condition (or numerical boundary condition).
- The **primitive type** boundary condition assign a field value to the surface patch (**base type**).
- You define the **primitive type** patch (or the value of the boundary condition), in the directory **0** or time directories.

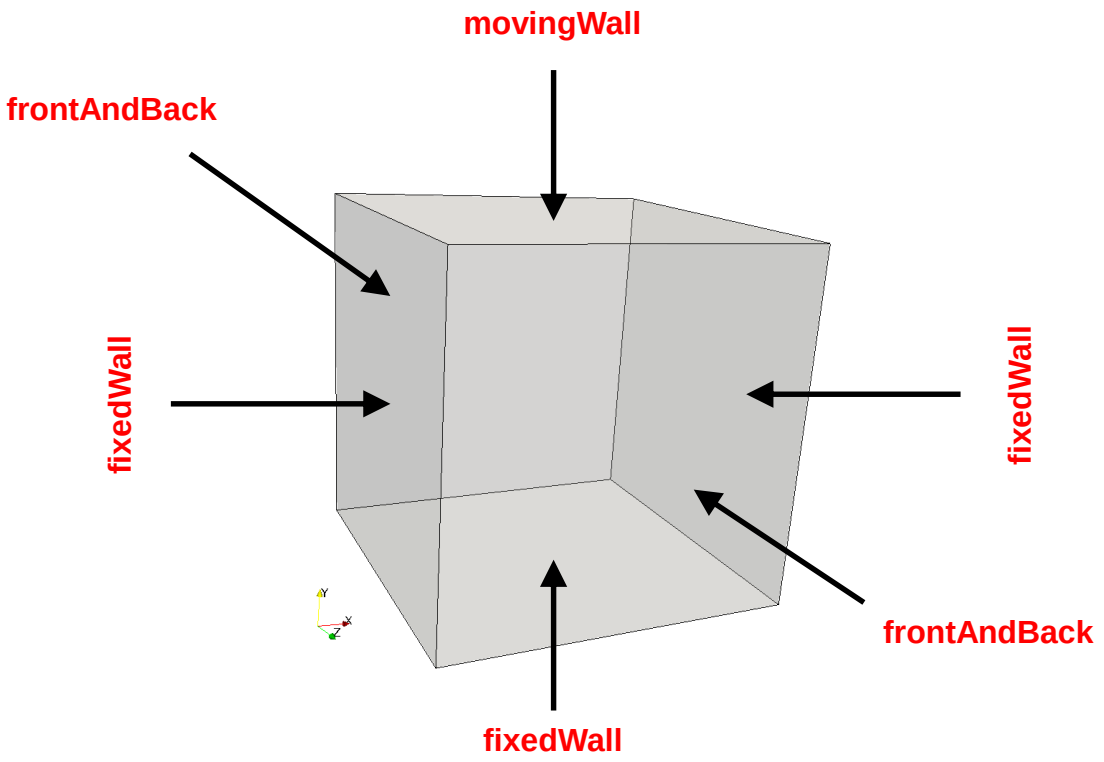
A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- In this case, the file *boundary* is divided as follows

```
18    3
19    (
20        movingWall
21        {
22            type            wall;
23            inGroups        1(wall);
24            nFaces          20;
25            startFace       760;
26        }
27        fixedWalls
28        {
29            type            wall;
30            inGroups        1(wall);
31            nFaces          60;
32            startFace       780;
33        }
34        frontAndBack
35        {
36            type            empty;
37            inGroups        1(empty);
38            nFaces          800;
39            startFace       840;
40        }
41    )
```

Number of surface patches
In the list bellow there must be 3 patches definition.

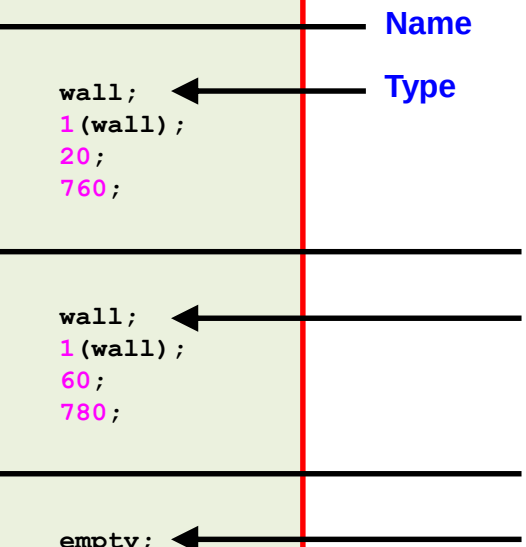


A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- In this case, the file *boundary* is divided as follows

```
18    3
19    (
20        movingWall
21        {
22            type            wall;
23            inGroups        1(wall);
24            nFaces          20;
25            startFace       760;
26        }
27        fixedWalls
28        {
29            type            wall;
30            inGroups        1(wall);
31            nFaces          60;
32            startFace       780;
33        }
34        frontAndBack
35        {
36            type            empty;
37            inGroups        1(empty);
38            nFaces          800;
39            startFace       840;
40        }
41    )
```



Name and type of the surface patches

- The name and type of the patch is given by the user.
- In this case the name and type was assigned in the dictionary *blockMeshDict*.
- You can change the name if you do not like it. Do not use strange symbols or white spaces.
- You can also change the **base type**. For instance, you can change the type of the patch **movingWal**l from **wall** to **patch**.
- When converting the mesh from a third party format, OpenFOAM® will try to recover the information from the original format. But it might happen that it does not recognizes the base type and name of the original. In this case you will need to modify this file manually.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- In this case, the file *boundary* is divided as follows

```
18    3
19    (
20        movingWall
21        {
22            type            wall;
23            inGroups        1(wall);
24            nFaces          20;
25            startFace       760;
26        }
27        fixedWalls
28        {
29            type            wall;
30            inGroups        1(wall);
31            nFaces          60;
32            startFace       780;
33        }
34        frontAndBack
35        {
36            type            empty;
37            inGroups        1(empty);
38            nFaces          800;
39            startFace       840;
40        }
41    )
```

inGroups keyword

- This keyword is optional. You can erase this information safely.
- It is used to group patches during visualization in ParaView/paraFoam. If you open this mesh in paraFoam you will see that there are two groups, namely: wall and empty.
- As usual, you can change the name.
- If you want to put a surface patch in two groups, you can proceed as follows:

  2(wall wall1)

  In this case the surface patch belongs to the groups **wall** and **wall1**.
- Groups can have more than one patch.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- In this case, the file *boundary* is divided as follows

```
18    3
19    (
20        movingWall
21        {
22            type            wall;
23            inGroups        1(wall);
24            nFaces          20;
25            startFace       760;
26        }
27        fixedWalls
28        {
29            type            wall;
30            inGroups        1(wall);
31            nFaces          60;
32            startFace       780;
33        }
34        frontAndBack
35        {
36            type            empty;
37            inGroups        1(empty);
38            nFaces          800;
39            startFace       840;
40        }
41    )
```

nFaces and startFace keywords

- Unless you know what you are doing, **you do not need to modify this information.**
- Basically, this is telling you the starting face and ending face of the patch.
- This is created automatically when generating the mesh or converting the mesh.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- In this case, the file *boundary* is divided as follows

```
18    3
19    (
20        movingWall
21        {
22            type            wall;
23            inGroups        1(wall);
24            nFaces          20;
25            startFace       760;
26        }
27        fixedWalls
28        {
29            type            wall;
30            inGroups        1(wall);
31            nFaces          60;
32            startFace       780;
33        }
34        frontAndBack
35        {
36            type            empty;
37            inGroups        1(empty);
38            nFaces          800;
39            startFace       840;
40        }
41    )
```

Remember

Boundary patches that are not recognize or assigned to a patch are grouped automatically in a default group named **defaultFaces** of type **empty**.

For instance, if you do not assign a patch to the patch **frontAndBack**, they will be grouped as follows:

```
defaultFaces
{
  type            empty;
  inGroups        1(empty);
  nFaces          800;
  startFace       840;
}
```

And as usual, you can manually change the name and type.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

Very important information on the boundary conditions 

- There are a few **base type** patches that are constrained or paired. This means that the type should be the same in the *boundary* file and in the numerical boundary condition defined in the field files, *e.g.*, the files *0/U* and *0/p*.
- In this case, the **base type** of the patch **frontAndBack** (defined in the file *boundary*), is consistent with the **primitive type** patch defined in the field files *0/U* and *0/p*. They are of the type **empty**.
- Also, the **base type** of the patches **movingWall** and **fixedWalls** (defined in the file *boundary*), is consistent with the **primitive type** patch defined in the field files *0/U* and *0/p*.
- This is extremely important, especially if you are converting meshes as not always the type of the patches is set as you would like.
- Hence, it is highly advisable to do a sanity check and verify that the **base type** of the patches (the type defined in the file *boundary*), is consistent with the **primitive type** of the patches (the patch type defined in the field files contained in the directory **0** (or whatever time directory you defined the boundary and initial conditions).
- If the **base type** and **primitive type** boundary conditions are not consistent, OpenFOAM® will complain.
- Do not worry, we are going to address boundary conditions later on.
- But for the moment, we will give you a brief introduction of how to pair boundary conditions and assign names to the boundary patches.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- The following **base type** boundary conditions are constrained or paired. That is, the type needs to be same in the *boundary* dictionary and field variables dictionaries (*e.g.* *U*, *p*).

constant/polyMesh/boundary	*0/U - 0/p (IC/BC)*
symmetry	**symmetry**
symmetryPlane	**symmetryPlane**
empty	**empty**
wedge	**wedge**
cyclic	**cyclic**
processor	**processor**

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- The **base type patch** can be any of the **primitive** or **derived type** boundary conditions available in OpenFOAM®. Mathematically speaking; they can be Dirichlet, Neumann or Robin boundary conditions.

constant/polyMesh/boundary	*0/U - 0/p (IC/BC)*
patch	**fixedValue** **zeroGradient** **inletOutlet** **slip** **totalPressure** **supersonicFreeStream** and so on … Refer to the doxygen documentation for a list of all numerical type boundary conditions available.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- The **wall** base type boundary condition is defined as follows:

constant/polyMesh/boundary	*0/U (IC/BC)*	*0/p (IC/BC)*
wall	**type fixedValue;** **value uniform (U V W);**	**zeroGradient**

- This boundary condition is not contained in the **patch** base type boundary condition group, because specialize modeling options can be used on this boundary condition.
- An example is turbulence modeling, where turbulence can be generated or dissipated at the walls.

A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

- The name of the **base type** boundary condition and the name of the **primitive type** boundary condition needs to be the same, if not, OpenFOAM® will complain.
- Pay attention to this, specially if you are converting the mesh from another format.

constant/polyMesh/boundary	*0/U (IC/BC)*	*0/p (IC/BC)*
movingWall **fixedWalls** **frontAndBack**	**movingWall** **fixedWalls** **frontAndBack**	**movingWall** **fixedWalls** **frontAndBack**

- As you can see, all the names are the same across all the dictionary files.

A deeper view to my first OpenFOAM® case setup

The **`system`** directory

(and by the way, open each file and go thru its content)

- The **`system`** directory consists of the following compulsory dictionary files:
  - *`controlDict`*
  - *`fvSchemes`*
  - *`fvSolution`*
- *`controlDict`* contains general instructions on how to run the case.
- *`fvSchemes`* contains instructions for the discretization schemes that will be used for the different terms in the equations.
- *`fvSolution`* contains instructions on how to solve each discretized linear equation system.
- Do not worry, we are going to study in details the most important entries of each dictionary (the compulsory entries).
- If you forget a compulsory keyword or give a wrong entry to the keyword, OpenFOAM® will complain and it will let you what are you missing. This applies for all the dictionaries in the hierarchy of the case directory.
- There are many optional parameters, to know all of them refer to the doxygen documentation or the source code. Hereafter we will try to introduce a few of them.
- OpenFOAM® will not complain if you are not using optional parameters, after all, they are optional. However, if the entry you use for the optional parameter is wrong OpenFOAM® will let you know.

A deeper view to my first OpenFOAM® case setup

The *controlDict* dictionary

```
17   application     icoFoam;
18
19   startFrom       startTime;
20
21   startTime       0;
22
23   stopAt          endTime;
24
25   endTime         50;
26
27   deltaT          0.01;
28
29   writeControl    runTime;
30
31   writeInterval   1;
32
33   purgeWrite      0;
34
35   writeFormat     ascii;
36
37   writePrecision  8;
38
39   writeCompression off;
40
41   timeFormat      general;
42
43   timePrecision   6;
44
45   runTimeModifiable true;
```

- The *controlDict* dictionary contains runtime simulation controls, such as, start time, end time, time step, saving frequency and so on.
- Most of the entries are self-explanatory.
- This case starts from time 0 (keyword **startFrom** – line 19 – and keyword **startTime** – line 21 –). If you have the initial solution in a different time directory, just enter the number in line 21.
- The case will stop when it reaches the desired time set using the keyword **stopAt** (line 23).
- It will run up to 50 seconds (keyword **endTime** – line 25 –).
- The time step of the simulation is 0.01 seconds (keyword **deltaT** – line 27 –).
- It will write the solution every second (keyword **writeInterval** – line 31 –) of simulation time (keyword **runTime** – line 29 –).
- It will keep all the solution directories (keyword **purgeWrite** – line 33 –). If you want to keep only the last 5 solutions just change the value to 5.
- It will save the solution in ascii format (keyword **writeFormat** – line 35 –) with a precision of 8 digits (keyword **writePrecision** – line 37 –).
- And as the option **runTimeModifiable** (line 45) is on (**true**), we can modify all these entries while we are running the simulation.
- FYI, you can modify the entries on-the-fly for most of the dictionaries files.

A deeper view to my first OpenFOAM® case setup

The *controlDict* dictionary

```
17  application     icoFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          banana;
24
25  endTime         50;
26
27  deltaT          0.01;
28
29  writeControl    runTime;
30
31  writeInterval   1;
32
33  purgeWrite      0;
34
35  writeFormat     ascii;
36
37  writePrecision  8;
38
39  writeCompression off;
40
41  timeFormat      general;
42
43  timePrecision   6;
44
45  runTimeModifiable true;
```

- So how do we know what options are available for each keyword?
- The hard way is to refer to the source code.
- The easy way is to use the **banana method**.
- So what is the **banana method**? This method consist in inserting a dummy word (that does not exist in the installation) and let OpenFOAM® list the available options.
- For example. If you add **banana** in line 23, you will get this output:

  banana is not in enumeration
  4
  (
  nextWrite
  writeNow
  noWriteNow
  endTime
)

- So your options are **nextWrite, writeNow, noWriteNow, endTime**
- We love to add bananas, it works with every dictionary.

A deeper view to my first OpenFOAM® case setup

The *controlDict* dictionary

```
17  application     icoFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          banana;
24
25  endTime         50;
26
27  deltaT          0.01;
28
29  writeControl    runTime;
30
31  writeInterval   1;
32
33  purgeWrite      0;
34
35  writeFormat     ascii;
36
37  writePrecision  8;
38
39  writeCompression off;
40
41  timeFormat      general;
42
43  timePrecision   6;
44
45  runTimeModifiable true;
```

- And how do we know that banana does not exist in the source code?
- Just type in the terminal:
  - `$> src`
  - `$> grep -r -n banana .`

- If you see some bananas in your output someone is messing around with your installation.
- You can use the same command to look for information in the OpenFOAM® installation, just replace the word banana for the word you are looking for.
- By the way, you can use any dummy word, but you have to be sure that it does not exist in OpenFOAM®.

A deeper view to my first OpenFOAM® case setup

The *controlDict* dictionary

```
17  application     icoFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          endTime;
24
25  //endTime         50;
26
27  deltaT          0.01;
28
29  writeControl    runTime;
30
31  writeInterval   1;
32
33  purgeWrite      0;
34
35  writeFormat     ascii;
36
37  writePrecision  8;
38
39  writeCompression off;
40
41  timeFormat      general;
42
43  timePrecision   6;
44
45  runTimeModifiable true;
```

- If you forget a compulsory keyword, OpenFOAM® will tell you what are you missing.
- So if you comment line 25, you will get this output:

```
--> FOAM FATAL IO ERROR
keyword endTime is undefined in dictionary …
```

- This output is just telling you that you are missing the keyword **endTime**.
- Do not pay attention to the words FATAL ERROR, maybe the developers of OpenFOAM® exaggerated a little bit.

A deeper view to my first OpenFOAM® case setup

The *fvSchemes* dictionary

```
17    ddtSchemes
18    {
19        default         backward;
20    }
21
22    gradSchemes
23    {
24        default         Gauss linear;
25        grad(p)         Gauss linear;
26    }
27
28    divSchemes
29    {
30        default         none;
31        div(phi,U)      Gauss linear;
32    }
33
34    laplacianSchemes
35    {
36        default         Gauss linear orthogonal;
37    }
38
39    interpolationSchemes
40    {
41        default         linear;
42    }
43
44    snGradSchemes
45    {
46        default         orthogonal;
47    }
```

- The *fvSchemes* dictionary contains the information related to the discretization schemes for the different terms appearing in the governing equations.
- As for the *controlDict* dictionary, the parameters can be changed on-the-fly.
- Also, if you want to know what options are available, just use the banana method.
- In this case we are using the **backward** method for time discretization (**ddtSchemes**). For gradients discretization (**gradSchemes**) we are using **Gauss linear** method. For the discretization of the convective terms (**divSchemes**) we are using **linear** interpolation for the term **div(phi,U)**.
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear** method with **orthogonal** corrections.
- The method we are using is second order accurate but oscillatory. We are going to talk about the properties of the numerical schemes later on.
- Remember, at the end of the day we want a solution that is second order accurate.

A deeper view to my first OpenFOAM® case setup

The *fvSolution* dictionary

```
17    solvers
18    {
19        p
20        {
21            solver          PCG;
22            preconditioner  DIC;
23            tolerance       1e-06;
24            relTol          0;

39        }
40
41        pFinal
42        {
43            $p;
44            relTol          0;
45        }
46
47        U
48        {
49            solver          smoothSolver;
50            smoother        symGaussSeidel;
51            tolerance       1e-08;
52            relTol          0;
53        }
54    }
55
56    PISO
57    {
58        nCorrectors     1;
59        nNonOrthogonalCorrectors 0;
60        pRefCell        0;
61        pRefValue       0;
62    }
```

- The *fvSolution* dictionary contains the instructions of how to solve each discretized linear equation system. The equation solvers, tolerances, and algorithms are controlled from the sub-dictionary **solvers**.
- In the dictionary file *fvSolution* (and depending on the solver you are using), you will find the additional sub-dictionaries **PISO, PIMPLE, SIMPLE**, and **relaxationFactors**. These entries will be described later.
- As for the *controlDict* *and* *fvSchemes* dictionaries, the parameters can be changed on-the-fly.
- Also, if you want to know what options are available just use the banana method.
- In this case, to solve the pressure (**p**) we are using the **PCG** method, with the preconditioner **DIC**, an absolute **tolerance** equal to 1e-06 and a relative tolerance **relTol** equal to 0.
- The entry **pFinal** refers to the final pressure correction (notice that we are using macro syntax), and we are using a relative tolerance **relTol** equal to 0. We are putting more computational effort in the last iteration.

A deeper view to my first OpenFOAM® case setup

The *fvSolution* dictionary

```
17    solvers
18    {
19        p
20        {
21            solver          PCG;
22            preconditioner  DIC;
23            tolerance       1e-06;
24            relTol          0;

39        }
40
41        pFinal
42        {
43            $p;
44            relTol          0;
45        }
46
47        U
48        {
49            solver          smoothSolver;
50            smoother        symGaussSeidel;
51            tolerance       1e-08;
52            relTol          0;
53        }
54    }
55
56    PISO
57    {
58        nCorrectors     1;
59        nNonOrthogonalCcorrectors 0;
60        pRefCell        0;
61        pRefValue       0;
62    }
```

- To solve **U** we are using the **smoothSolver** method, with the smoother **symGaussSeidel**, an absolute **tolerance** equal to 1e-08 and a relative tolerance **relTol** equal to 0.
- The solvers will iterative until reaching any of the tolerance values set by the user or reaching a maximum value of iterations (optional entry).
- FYI, solving for the velocity is relative inexpensive, whereas solving for the pressure is expensive.
- The **PISO** sub-dictionary contains entries related to the pressure-velocity coupling method (the **PISO** method).
- In this case we are doing only one **PISO** correction and no orthogonal corrections.
- You need to do at least one **PISO** loop (**nCorrectors**).

A deeper view to my first OpenFOAM® case setup

The **`system`** directory
(optional dictionary files)

- In the **`system`** directory you will also find these two additional files:
  - *`decomposeParDict`*
  - *`sampleDict`*

- *`decomposeParDict`* is read by the utility `decomposePar`. This dictionary file contains information related to the mesh partitioning. This is used when running in parallel. We will address running in parallel later.
- *`sampleDict`* is read by the utility `postProcess`. This utility sample field data (points, lines or surfaces). In this dictionary file we specify the sample location and the fields to sample. The sampled data can be plotted using gnuplot or Python.

A deeper view to my first OpenFOAM® case setup

The *sampleDict* dictionary

```
17    type sets;
18
19    setFormat raw;
20
23    interpolationScheme cellPointFace;
24
26    fields
27    (
28        U
29    );
30
31    sets
32    (
33
34        l1
35        {
36            type            midPointAndFace;
43            axis            x;
44            start           ( -1  0.5 0);
45            end             ( 2  0.5 0);
46        }
47
48        l2
49        {
50            type            midPointAndFace;
54            axis            y;
55            start           (0.5 -1 0);
56            end             (0.5 2 0);
57        }
58
59    );
```

Type of sampling, sets will sample along a line.

Format of the output file, raw format is a generic format that can be read by many applications. The output file is human readable (ascii format).

Interpolation method at the solution level (location of the interpolation points).

Fields to sample.

Sample method. How to interpolate the solution to the sample entity (line in this case)

Location of the sample line. We define start and end point, and the axis of the sampling.

Sample method from the solution to the line.

Location of the sample line. We define start and end point, and the axis of the sampling.

A deeper view to my first OpenFOAM® case setup

The *sampleDict* dictionary

```
17    type sets;
18
19    setFormat raw;
20
23    interpolationScheme cellPointFace;
24
26    fields
27    (
28        U
29    );
30
31    sets
32    (
33
34        l1
35        {
36            type            midPointAndFace;
43            axis            x;
44            start           ( -1  0.5 0);
45            end             ( 2  0.5 0);
46        }
47
48        l2
49        {
50            type            midPointAndFace;
54            axis            y;
55            start           (0.5 -1 0);
56            end             (0.5 2 0);
57        }
58
59    );
```

The sampled information is always saved in the directory,

postProcessing/name_of_input_dictionary

As we are sampling the latest time solution (50) and using the dictionary *sampleDict*, the sampled data will be located in the directory:

postProcessing/sampleDict/50

The files *l1_U.xy* and *l2_U.xy* located in the directory **postProcessing/sampleDict/50** contain the sampled data. Feel free to open them using your favorite text editor.

Name of the output file

Name of the output file

A deeper view to my first OpenFOAM® case setup

The 0 directory

(and by the way, open each file and go thru its content)

- The **0** directory contains the initial and boundary conditions for all primitive variables, in this case *p* and *U*. The *U* file contains the following information (velocity vector):

```
17    dimensions      [0 1 -1 0 0 0 0];
18
19    internalField   uniform (0 0 0);
20
21    boundaryField
22    {
23        movingWall
24        {
25            type            fixedValue;
26            value           uniform (1 0 0);
27        }
28
29        fixedWalls
30        {
31            type            fixedValue;
32            value           uniform (0 0 0);
33        }
34
35        frontAndBack
36        {
37            type            empty;
38        }
39    }
```

Dimensions of the field $\frac{m}{s}$

Uniform initial conditions.

The velocity field is initialize to **(0 0 0)** in all the domain

Remember velocity is a vector with three components, therefore the notation **(0 0 0)**.

Note:
If you take some time and compare the files *0/U* and *constant/polyMesh/boundary*, you will see that the name and type of each **primitive type** patch (the patch defined in *0/U*), is consistent with the **base type** patch (the patch defined in the file *constant/polyMesh/boundary*).

A deeper view to my first OpenFOAM® case setup

The 0 directory

(and by the way, open each file and go thru its content)

- The **0** directory contains the initial and boundary conditions for all primitive variables, in this case p and U. The U file contains the following information (velocity):

```
17    dimensions      [0 1 -1 0 0 0 0];
18
19    internalField   uniform (0 0 0);
20
21    boundaryField
22    {
23        movingWall
24        {
25            type            fixedValue;
26            value           uniform (1 0 0);
27        }
28
29        fixedWalls
30        {
31            type            fixedValue;
32            value           uniform (0 0 0);
33        }
34
35        frontAndBack
36        {
37            type            empty;
38        }
39    }
```

Dimensions of the field $\frac{m}{s}$

Numerical boundary condition for the patch **movingWall**

Numerical boundary condition for the patch **fixedWalls**

Numerical boundary condition for the patch **frontAndBack** (this is a constrained boundary condition).

A deeper view to my first OpenFOAM® case setup

The 0 directory

(and by the way, open each file and go thru its content)

- The **0** directory contains the initial and boundary conditions for all primitive variables, in this case *p* and *U*. The *p* file contains the following information (modified pressure):

```
17    dimensions      [0 2 -2 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        movingWall
24        {
25            type            zeroGradient;
26        }
27
28        fixedWalls
29        {
30            type            zeroGradient;
31        }
32
33        frontAndBack
34        {
35            type            empty;
36        }
37    }
38
```

Dimensions of the field $\frac{m^2}{s^2}$

Uniform initial conditions.

The modified pressure field is initialize to **0** in all the domain. **This is relative pressure.**

Note:
If you take some time and compare the files *0/p* and *constant/polyMesh/boundary*, you will see that the name and type of each **primitive type** patch (the patch defined in *0/p*), is consistent with the **base type** patch (the patch defined in the file *constant/polyMesh/boundary*).

A deeper view to my first OpenFOAM® case setup

The 0 directory

(and by the way, open each file and go thru its content)

- The **0** directory contains the initial and boundary conditions for all primitive variables, in this case p and U. The p file contains the following information (modified pressure):

```
17    dimensions      [0 2 -2 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        movingWall
24        {
25            type            zeroGradient;
26        }
27
28        fixedWalls
29        {
30            type            zeroGradient;
31        }
32
33        frontAndBack
34        {
35            type            empty;
36        }
37    }
38
```

Dimensions of the field $\frac{m^2}{s^2}$

Numerical boundary condition for the patch **movingWall**

Numerical boundary condition for the patch **fixedWalls**

Numerical boundary condition for the patch **frontAndBack** (this is a constrained boundary condition).

A deeper view to my first OpenFOAM® case setup

A very important remark on the pressure field

- We just used `icoFoam` which is an incompressible solver.
- **Let us be really loud on this.** All the incompressible solvers implemented in OpenFOAM® (`icoFoam`, `simpleFoam`, `pisoFoam`, and `pimpleFoam`), use the modified pressure, that is,

$$P = \frac{p}{\rho} \quad \text{with units} \quad \frac{m^2}{s^2}$$

- Or in OpenFOAM® jargon: **`dimensions [0 2 -2 0 0 0 0]`**
- So when visualizing or post processing the results **do not forget to multiply the pressure by the density** in order to get the right units of the physical pressure, that is,

$$\frac{kg}{m \cdot s^2}$$

- Or in OpenFOAM® jargon: **`dimensions [1 -1 -2 0 0 0 0]`**

A deeper view to my first OpenFOAM® case setup

- Coming back to the headers, and specifically the headers related to the field variable dictionaries (e.g. *U*, *p*).
- In the header of your field variables, the class type should be consistent with the type of field variable you are using.
- If the field variable is a scalar, the class should be **volScalarField**.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

A deeper view to my first OpenFOAM® case setup

- Coming back to the headers, and specifically the headers related to the field variable dictionaries (e.g. *U*, *p*).
- In the header of your field variables, the class type should be consistent with the type of field variable you are using.
- If the field variable is a vector, the class should be **volVectorField.**

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

A deeper view to my first OpenFOAM® case setup

- Coming back to the headers, and specifically the headers related to the field variable dictionaries (e.g. U, p).
- In the header of your field variables, the class type should be consistent with the type of field variable you are using.
- If the field variable is a tensor (e.g. the velocity gradient tensor), the class should be **volTensorField**.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volTensorField;
    object      gradU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

A deeper view to my first OpenFOAM® case setup

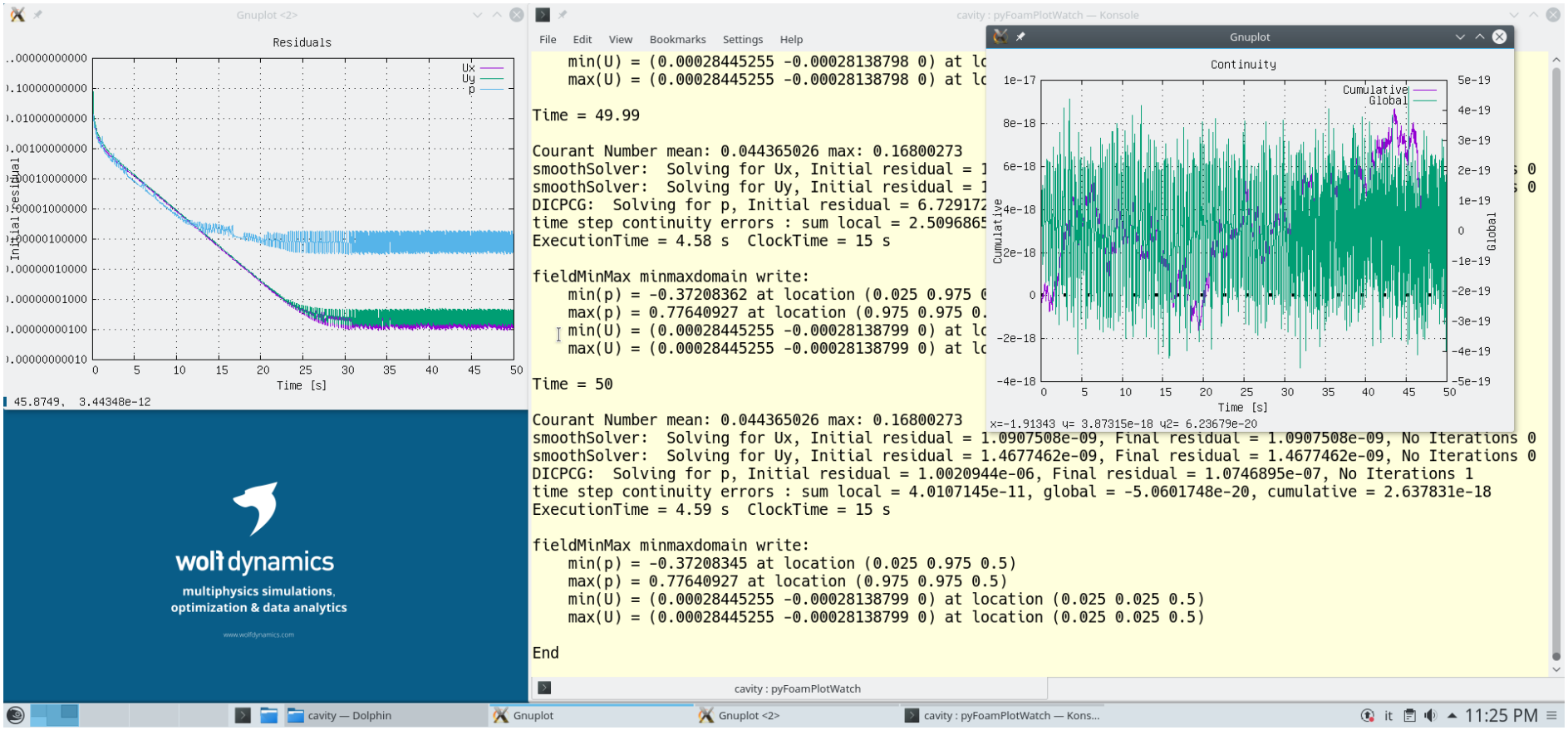
The *log.icoFoam* file

- If you followed the previous instructions you should now have the *log.icoFoam* file. This file contains all the residuals and convergence information.
- We already plotted this information using `foamLog` and `gnuplot`.
- Let us plot this information again but this time using PyFoam, in the terminal type:
  - `$> pyFoamPlotWatcher.py log.icoFoam --with-courant`
- The script `pyFoamPlotWatcher.py` will plot the information in the log file, even if the simulation is not running.
- The option `--with-courant` will plot the courant number.
- Remember, to use PyFoam you will need to source it. Type in the terminal:
  - `$> anaconda2` or `$> anaconda3`

A deeper view to my first OpenFOAM® case setup

The *log.icoFoam* file

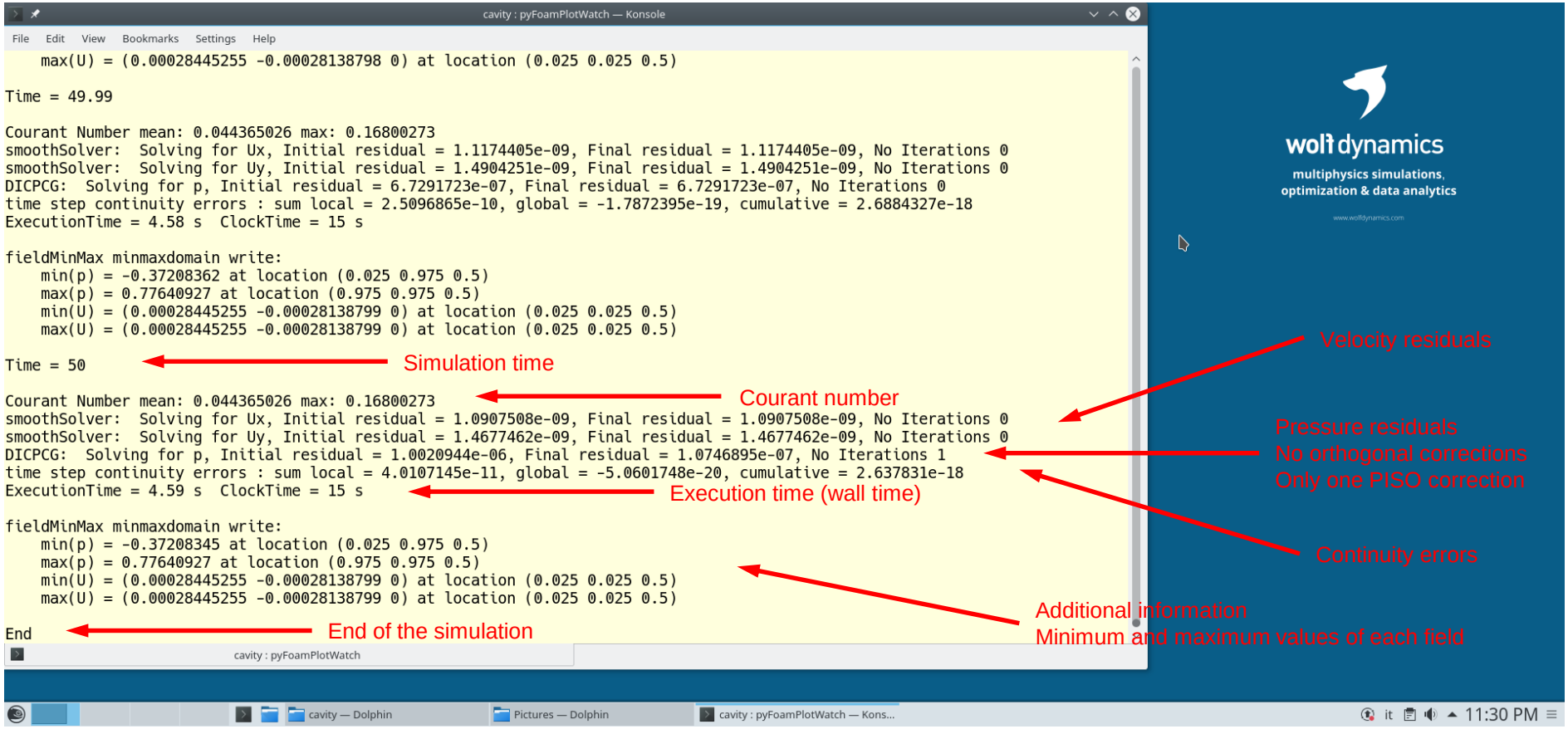
- In this case, `pyFoamPlotWatcher` is plotting the initial residuals, continuity errors and courant number.



A deeper view to my first OpenFOAM® case setup

The output screen

- Finally, let us talk about the output screen, which shows a lot of information.



A deeper view to my first OpenFOAM® case setup

The output screen

- By default, OpenFOAM ® does not show the minimum and maximum information. To print out this information, we use **functionObjects**. We are going to address **functionObjects** in detail when we deal with post-processing and sampling.
- But for the moment, what we need to know is that we add **functionObjects** at the end of the `controlDict` dictionary. In this case, we are using a **functionObject** that prints the minimum and maximum information of the selected fields.
- This information complements the residuals information and it is saved in the **postProcessing** directory. It gives a better indication of stability, boundedness and consistency of the solution.

```
49    functions
50    {
51
52    ////////////////////////////////////////////////////////////////////////
53
54    minmaxdomain
55    {
56        type fieldMinMax;
57
58        functionObjectLibs ("libfieldFunctionObjects.so");
59
60        enabled true; //true or false
61
62        mode component;
63
64        writeControl timeStep;
65        writeInterval 1;
66
67        log true;
68
69        fields (p U);
70    }
91
92    };
```

Name of the folder where the output of the functionObject will be saved

functionObject to use

Turn on/off functionObject

Output interval of functionObject

Save output of the functionObject in a ascii file

Field variables to sample

A deeper view to my first OpenFOAM® case setup

The output screen

- Another very important output information is the CFL or Courant number.
- In one dimension, the CFL number is defined as,

$$CFL = \frac{u\,\Delta t}{\Delta x}$$

- The CFL number is a measure of how much information (u) traverses a computational grid cell (Δx) in a given time-step (Δt).
- The Courant number imposes the **CFL number condition,** which is the maximum allowable CFL number a numerical scheme can use. For the n - dimensional case, the CFL number condition becomes,

$$CFL = \Delta t \sum_{i=1}^{n} \frac{u_i}{\Delta x_i} \leq CFL_{max}$$

A deeper view to my first OpenFOAM® case setup

The output screen

- The CFL number is a necessary condition to guarantee the stability of the numerical method.
- But not all numerical methods have the same stability constrains.
- By the way, when we talk about numerical methods we are referring to implicit and explicit methods.
- In OpenFOAM®, most of the solvers are implicit, which means they are **unconditionally stable**. In other words, they are not constrained to the **CFL number condition**.
- However, the fact that you are using a numerical method that is **unconditionally stable**, does not mean that you can choose a time step of any size.
- The time-step must be chosen in such a way that it resolves the time-dependent features, and it maintains the solver stability.
- For the moment and for the sake of simplicity, let us try to keep the CFL number below 2.0 and preferably less than 1.0
- Other properties of the numerical method that you should observe are: conservationess, boundedness, transportiveness, and accuracy. We are going to address these properties and the CFL number when we deal with the FVM theory.

A deeper view to my first OpenFOAM® case setup

The output screen

- To control the CFL number you can change the time step or you can change the mesh (the easiest way is by changing the time step).
- For a time step of 0.01 seconds, this is the output we get,

```
Time = 49.99

Courant Number mean: 0.044365026 max: 0.16800273
smoothSolver:  Solving for Ux, Initial residual = 1.1174405e-09, Final residual = 1.1174405e-09, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 1.4904251e-09, Final residual = 1.4904251e-09, No Iterations 0
DICPCG:  Solving for p, Initial residual = 6.7291723e-07, Final residual = 6.7291723e-07, No Iterations 0
time step continuity errors : sum local = 2.5096865e-10, global = -1.7872395e-19, cumulative = 2.6884327e-18
ExecutionTime = 4.47 s  ClockTime = 5 s

fieldMinMax minmaxdomain output:
    min(p) = -0.37208362 at location (0.025 0.975 0.5)
    max(p) = 0.77640927 at location (0.975 0.975 0.5)
    min(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)
    max(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)

Time = 50

Courant Number mean: 0.044365026 max: 0.16800273
smoothSolver:  Solving for Ux, Initial residual = 1.0907508e-09, Final residual = 1.0907508e-09, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 1.4677462e-09, Final residual = 1.4677462e-09, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1.0020944e-06, Final residual = 1.0746895e-07, No Iterations 1
time step continuity errors : sum local = 4.0107145e-11, global = -5.0601748e-20, cumulative = 2.637831e-18
ExecutionTime = 4.47 s  ClockTime = 5 s

fieldMinMax minmaxdomain output:
    min(p) = -0.37208345 at location (0.025 0.975 0.5)
    max(p) = 0.77640927 at location (0.975 0.975 0.5)
    min(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)
    max(U) = (0.00028445255 -0.00028138799 0) at location (0.025 0.025 0.5)
```

CFL number at time step n - 1

CFL number at time step n

A deeper view to my first OpenFOAM® case setup

The output screen

- To control the CFL number you can change the time step or you can change the mesh (the easiest way is by changing the time step).
- For a time step of 0.1 seconds, this is the output we get,

```
Time = 49.9

Courant Number mean: 0.4441161 max: 1.6798756
smoothSolver:  Solving for Ux, Initial residual = 0.00016535808, Final residual = 2.7960145e-09, No Iterations 5
smoothSolver:  Solving for Uy, Initial residual = 0.00015920267, Final residual = 2.7704949e-09, No Iterations 5
DICPCG:  Solving for p, Initial residual = 0.0015842846, Final residual = 5.2788554e-07, No Iterations 26
time step continuity errors : sum local = 8.6128916e-09, global = 3.5439859e-19, cumulative = 2.4940081e-17
ExecutionTime = 0.81 s  ClockTime = 1 s

fieldMinMax minmaxdomain output:
    min(p) = -0.34322821 at location (0.025 0.975 0.5)
    max(p) = 0.73453489 at location (0.975 0.975 0.5)
    min(U) = (0.0002505779 -0.00025371425 0) at location (0.025 0.025 0.5)
    max(U) = (0.0002505779 -0.00025371425 0) at location (0.025 0.025 0.5)

Time = 50

Courant Number mean: 0.44411473 max: 1.6798833
smoothSolver:  Solving for Ux, Initial residual = 0.00016378098, Final residual = 2.7690608e-09, No Iterations 5
smoothSolver:  Solving for Uy, Initial residual = 0.00015720331, Final residual = 2.7354499e-09, No Iterations 5
DICPCG:  Solving for p, Initial residual = 0.0015662416, Final residual = 5.2290439e-07, No Iterations 26
time step continuity errors : sum local = 8.5379223e-09, global = -3.6676527e-19, cumulative = 2.4573316e-17
ExecutionTime = 0.81 s  ClockTime = 1 s

fieldMinMax minmaxdomain output:
    min(p) = -0.34244269 at location (0.025 0.975 0.5)
    max(p) = 0.73656831 at location (0.975 0.975 0.5)
    min(U) = (0.00025028679 -0.00025338014 0) at location (0.025 0.025 0.5)
    max(U) = (0.00025028679 -0.00025338014 0) at location (0.025 0.025 0.5)
```

CFL number at time step n - 1

CFL number at time step n

A deeper view to my first OpenFOAM® case setup

The output screen

- To control the CFL number you can change the time step or you can change the mesh (the easiest way is by changing the time step).
- For a time step of 0.5 seconds, this is the output we get,

```
Time = 2

Courant Number mean: 1.6828931 max: 5.6061178
smoothSolver:  Solving for Ux, Initial residual = 0.96587058, Final residual = 4.9900041e-09, No Iterations 27
smoothSolver:  Solving for Uy, Initial residual = 0.88080685, Final residual = 9.7837781e-09, No Iterations 25
DICPCG:  Solving for p, Initial residual = 0.95568243, Final residual = 7.9266324e-07, No Iterations 33
time step continuity errors : sum local = 6.3955627e-06, global = 1.3227253e-17, cumulative = 1.4125109e-17
ExecutionTime = 0.04 s  ClockTime = 0 s

fieldMinMax minmaxdomain output:
    min(p) = -83.486425 at location (0.975 0.875 0.5)
    max(p) = 33.078468 at location (0.025 0.925 0.5)
    min(U) = (0.1309243 -0.13648118 0) at location (0.025 0.025 0.5)
    max(U) = (0.1309243 -0.13648118 0) at location (0.025 0.025 0.5)

Time = 2.5

Courant Number mean: 8.838997 max: 43.078153
#0  Foam::error::printStack(Foam::Ostream&) at ??:?
#1  Foam::sigFpe::sigHandler(int) at ??:?
#2  ? in "/lib64/libc.so.6"
#3  Foam::symGaussSeidelSmoother::smooth(Foam::word const&, Foam::Field<double>&, Foam::lduMatrix const&, Foam::Field<double> const&,
Foam::FieldField<Foam::Field, double> const&, Foam::UPtrList<Foam::lduInterfaceField const> const&, unsigned char, int) at ??:?
#4  Foam::symGaussSeidelSmoother::smooth(Foam::Field<double>&, Foam::Field<double> const&, unsigned char, int) const at ??:?
#5  Foam::smoothSolver::solve(Foam::Field<double>&, Foam::Field<double> const&, unsigned char) const at ??:?
#6  ? at ??:?
```

CFL number at time step n - 1

Compare these values with the values of the previous cases. For the physics involve these values are unphysical.

CFL number at time step n (way too high)

The solver crashed.
The offender? Time step too large.

A deeper view to my first OpenFOAM® case setup

The output screen

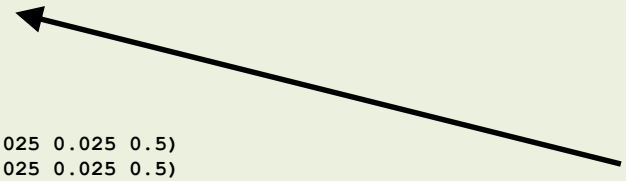
- Another output you should monitor are the continuity errors.
- These numbers should be small (it does not matter if they are negative or positive).
- If these values increase in time (about the order of 1e-3), you better control the case setup because something is wrong.
- The continuity errors are defined in the following file

$WM_PROJECT_DIR/src/finiteVolume/cfdTools/incompressible/continuityErrs.H

```
Time = 50

Courant Number mean: 0.44411473 max: 1.6798833
smoothSolver:  Solving for Ux, Initial residual = 0.00016378098, Final residual = 2.7690608e-09, No Iterations 5
smoothSolver:  Solving for Uy, Initial residual = 0.00015720331, Final residual = 2.7354499e-09, No Iterations 5
DICPCG:  Solving for p, Initial residual = 0.0015662416, Final residual = 5.2290439e-07, No Iterations 26
time step continuity errors : sum local = 8.5379223e-09, global = -3.6676527e-19, cumulative = 2.4573316e-17
ExecutionTime = 0.81 s  ClockTime = 1 s

fieldMinMax minmaxdomain output:
    min(p) = -0.34244269 at location (0.025 0.975 0.5)
    max(p) = 0.73656831 at location (0.975 0.975 0.5)
    min(U) = (0.00025028679 -0.00025338014 0) at location (0.025 0.025 0.5)
    max(U) = (0.00025028679 -0.00025338014 0) at location (0.025 0.025 0.5)
```



A deeper view to my first OpenFOAM® case setup

Error output

- If you forget a keyword or a dictionary file, give a wrong option to a compulsory or optional entry, misspelled something, add something out of place in a dictionary, use the wrong dimensions, forget a semi-colon and so on, OpenFOAM® will give you the error `FOAM FATAL IO ERROR`.
- This error does not mean that the actual OpenFOAM® installation is corrupted. It is telling you that you are missing something or something is wrong in a dictionary.
- Maybe the guys of OpenFOAM® went a little bit extreme here.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.x-5d8318b22cbe
Exec   : icoFoam
Date   : Nov 02 2014
Time   : 00:33:41
Host   : "linux-cfd"
PID    : 3675
Case   : /home/cfd/my_cases_course/cavity
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM FATAL IO ERROR:
```

A deeper view to my first OpenFOAM® case setup

Error output

- Also, before entering into panic read carefully the output screen because OpenFOAM® is telling you what is the error and how to correct it.

```
Build  : 4.x-5d8318b22cbe
Exec   : icoFoam
Date   : Nov 02 2014
Time   : 00:33:41
Host   : "linux-cfd"
PID    : 3675
Case   : /home/cfd/my_cases_course/cavity
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time


--> FOAM FATAL IO ERROR:

banana_endTime is not in enumeration:
4
(
endTime
nextWrite
noWriteNow
writeNow
)

file: /home/cfd/my_cases_course/cavity/system/controlDict.stopAt at line 24.

    From function NamedEnum<Enum, nEnum>::read(Istream&) const
    in file lnInclude/NamedEnum.C at line 72.

FOAM exiting
```

The origin of the error

Possible options to correct the error

Location of the error

A deeper view to my first OpenFOAM® case setup

Error output

- It is very important to read the screen and understand the output.

> *"Experience is simply the name we give our mistakes."*

- Train yourself to identify the errors. Hereafter we list a few possible errors.
- Missing compulsory file p

```
--> FOAM FATAL IO ERROR:
cannot find file

file: /home/joegi/my_cases_course/4x/101OF/cavity/0/p at line 0.

    From function regIOobject::readStream()
    in file db/regIOobject/regIOobjectRead.C at line 73.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- Missing keyword **class** in file *p*

```
--> FOAM FATAL IO ERROR:
keyword class is undefined in dictionary "/home/joegi/my_cases_course/4x/101OF/cavity/0/p"

file: /home/joegi/my_cases_course/4x/101OF/cavity/0/p from line 10 to line 13.

    From function dictionary::lookupEntry(const word&, bool, bool) const
    in file db/dictionary/dictionary.C at line 442.

FOAM exiting
```

- Misspelled word in file *boundary*

```
--> FOAM FATAL IO ERROR:
unexpected class name spolyBoundaryMesh expected polyBoundaryMesh
    while reading object boundary

file: /home/joegi/my_cases_course/4x/101OF/cavity/constant/polyMesh/boundary at line 15.

    From function regIOobject::readStream(const word&)
    in file db/regIOobject/regIOobjectRead.C at line 136.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- Mismatching patch name in file *p*

```
--> FOAM FATAL IO ERROR:
Cannot find patchField entry for xmovingWall

file: /home/joegi/my_cases_course/4x/101OF/cavity/0/p.boundaryField from line 25 to line 35.

    From function GeometricField<Type, PatchField, GeoMesh>::GeometricBoundaryField::readField(const
DimensionedField<Type, GeoMesh>&, const dictionary&)
    in file /home/joegi/OpenFOAM/OpenFOAM-4.x/src/OpenFOAM/lnInclude/GeometricBoundaryField.C at line 209.

FOAM exiting
```

- Missing compulsory keyword in *fvSchemes*

```
--> FOAM FATAL IO ERROR:
keyword div(phi,U) is undefined in dictionary
"/home/joegi/my_cases_course/4x/101OF/cavity/system/fvSchemes.divSchemes"

file: /home/joegi/my_cases_course/4x/101OF/cavity/system/fvSchemes.divSchemes from line 30 to line 30.

    From function dictionary::lookupEntry(const word&, bool, bool) const
    in file db/dictionary/dictionary.C at line 442.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- Missing entry in file *fvSolution* at keyword **PISO**

```
--> FOAM FATAL IO ERROR:
"ill defined primitiveEntry starting at keyword 'PISO' on line 68 and ending at line 68"

file: /home/joegi/my_cases_course/4x/101OF/cavity/system/fvSolution at line 68.

    From function primitiveEntry::readEntry(const dictionary&, Istream&)
    in file lnInclude/IOerror.C at line 132.

FOAM exiting
```

- Incompatible dimensions. Likely the offender is the file *U*

```
--> FOAM FATAL ERROR:
incompatible dimensions for operation
    [U[0 1 -2 1 0 0 0] ] + [U[0 1 -2 2 0 0 0] ]

    From function checkMethod(const fvMatrix<Type>&, const fvMatrix<Type>&)
    in file /home/joegi/OpenFOAM/OpenFOAM-4.x/src/finiteVolume/lnInclude/fvMatrix.C at line 1295.

FOAM aborting

#0  Foam::error::printStack(Foam::Ostream&) at ??:?
#1  Foam::error::abort() at ??:?
#2  void Foam::checkMethod<Foam::Vector<double> >(Foam::fvMatrix<Foam::Vector<double> > const&,
Foam::fvMatrix<Foam::Vector<double> > const&, char const*) at ??:?
#3  ? at ??:?
#4  ? at ??:?
#5  __libc_start_main in "/lib64/libc.so.6"
#6  ? at /home/abuild/rpmbuild/BUILD/glibc-2.19/csu/../sysdeps/x86_64/start.S:125
Aborted
```

A deeper view to my first OpenFOAM® case setup

Error output

- Missing keyword **deltaT** in file *controlDict*

```
--> FOAM FATAL IO ERROR:
keyword deltaT is undefined in dictionary "/home/joegi/my_cases_course/4x/101OF/cavity/system/controlDict"

file: /home/joegi/my_cases_course/4x/101OF/cavity/system/controlDict from line 17 to line 69.

    From function dictionary::lookupEntry(const word&, bool, bool) const
    in file db/dictionary/dictionary.C at line 442.

FOAM exiting
```

- Missing file *points* in directory polyMesh. Likely you are missing the mesh.

```
--> FOAM FATAL ERROR:
Cannot find file "points" in directory "polyMesh" in times 0 down to constant

    From function Time::findInstance(const fileName&, const word&, const IOobject::readOption, const word&)
    in file db/Time/findInstance.C at line 203.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- Unknown boundary condition type.

```
--> FOAM FATAL IO ERROR:
Unknown patchField type sfixedValue for patch type wall

Valid patchField types are :

74
(
SRFFreestreamVelocity
SRFVelocity
SRFWallVelocity
activeBaffleVelocity

...
...
...

variableHeightFlowRateInletVelocity
waveTransmissive
wedge
zeroGradient
)


file: /home/joegi/my_cases_course/4x/101OF/cavity/0/U.boundaryField.movingWall from line 25 to line 26.

    From function fvPatchField<Type>::New(const fvPatch&, const DimensionedField<Type, volMesh>&, const
dictionary&)
    in file /home/joegi/OpenFOAM/OpenFOAM-4.x/src/finiteVolume/lnInclude/fvPatchFieldNew.C at line 143.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- This one is specially hard to spot

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.x-5d8318b22cbe
Exec   : icoFoam
Date   : Nov 02 2014
Time   : 00:33:41
Host   : "linux-cfd"
PID    : 3675
fileName::stripInvalid() called for invalid fileName /home/cfd/my_cases_course/cavity0
    For debug level (= 2) > 1 this is considerd fatal
Aborted
```

- This error is related to the name of the working directory. In this case the name of the working directory is **`cavity 0`** (there is a white space between the word cavity and the number 0).
- Do not use white spaces or funny symbols when naming directories and files.

- Instead of **`cavity 0`** you should use **`cavity_0`**.

A deeper view to my first OpenFOAM® case setup

Error output

- You should worry about the `SIGFPE` error signal. This error signal indicates that something went really wrong (erroneous arithmetic operation).
- This message (that seems a little bit difficult to understand), is giving you a lot information.
- For instance, this output is telling us that the error is due to `SIGFPE` and the class associated to the error is `lduMatrix`. It is also telling you that the `GAMGSolver` solver is the affected one (likely the offender is the pressure).

```
#0  Foam::error::printStack(Foam::Ostream&) at ??:?
#1  Foam::sigFpe::sigHandler(int) at ??:?
#2   in "/lib64/libc.so.6"
#3  Foam::DICPreconditioner::calcReciprocalD(Foam::Field<double>&, Foam::lduMatrix const&) at ??:?
#4  Foam::DICSmoother::DICSmoother(Foam::word const&, Foam::lduMatrix const&, Foam::FieldField<Foam::Field, double>
const&, Foam::FieldField<Foam::Field, double> const&, Foam::UPtrList<Foam::lduInterfaceField const> const&) at ??:?
#5  Foam::lduMatrix::smoother::addsymMatrixConstructorToTable<Foam::DICSmoother>::New(Foam::word const&,
Foam::lduMatrix const&, Foam::FieldField<Foam::Field, double> const&, Foam::FieldField<Foam::Field, double> const&,
Foam::UPtrList<Foam::lduInterfaceField const> const&) at ??:?
#6  Foam::lduMatrix::smoother::New(Foam::word const&, Foam::lduMatrix const&, Foam::FieldField<Foam::Field, double>
const&, Foam::FieldField<Foam::Field, double> const&, Foam::UPtrList<Foam::lduInterfaceField const> const&,
Foam::dictionary const&) at ??:?
#7  Foam::GAMGSolver::initVcycle(Foam::PtrList<Foam::Field<double> >&, Foam::PtrList<Foam::Field<double> >&,
Foam::PtrList<Foam::lduMatrix::smoother>&, Foam::Field<double>&, Foam::Field<double>&) const at ??:?
#8  Foam::GAMGSolver::solve(Foam::Field<double>&, Foam::Field<double> const&, unsigned char) const at ??:?
#9  Foam::fvMatrix<double>::solveSegregated(Foam::dictionary const&) at ??:?
#10  Foam::fvMatrix<double>::solve(Foam::dictionary const&) at ??:?
#11
 at ??:?
#12  __libc_start_main in "/lib64/libc.so.6"
#13
 at /home/abuild/rpmbuild/BUILD/glibc-2.17/csu/../sysdeps/x86_64/start.S:126
Floating point exception
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- OpenFOAM® follows same general syntax rules as in C++.
- Commenting in OpenFOAM® (same as in C++):

```
// This is a line comment

/*
    This is a block comment
*/
```

- The **#include** directive (same as in C++):

```
#include "initialConditions"
```

Do not forget to create the respective include file *`initialConditions`*.

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Scalars, vectors, lists and dictionaries.
  - Scalars in OpenFOAM® are represented by a single value, e.g.,

    3.14159

  - Vectors in OpenFOAM® are represented as a list with three components, e.g.,

    (1.0 0.0 0.0)

  - A second order tensor in OpenFOAM® is represented as a list with nine components, e.g.,

```
(
    1.0  0.0  0.0
    0.0  1.0  0.0
    0.0  0.0  1.0
)
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Scalars, vectors, lists and dictionaries.
  - List entries are contained within parentheses **( )**. A list can contain scalars, vectors, tensors, words, and so on.
    - A list of scalars is represented as follows:

```
name_of_the_list
(
    0
    1
    2
);
```

    - A list of vectors is represented as follows:

```
name_of_the_list
(
    (0 0 0)
    (1 0 0)
    (2 0 0)
);
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Scalars, vectors, lists and dictionaries.
  - List entries are contained within parentheses **( )**. A list can contain scalars, vectors, tensors, words, and so on.
    - A list of words is represented as follows

```
name_of_the_list
(
    "word1"
    "word2"
    "word3"
);
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Scalars, vectors, lists and dictionaries.
  - OpenFOAM® uses dictionaries to specify data in an input file (dictionary file).
  - A dictionary in OpenFOAM® can contain multiple data entries and at the same time dictionaries can contain sub-dictionaries.
  - To specify a dictionary entry, the name is follow by the keyword entries in curly braces:

```
solvers                                   <-- Dictionary solvers
{
    p                                     <-- Sub-dictionary p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }
}
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Macro expansion.
  - We first declare a variable (**x = 10**) and then we use it through the **$** macro substitution (**$x**).

```
vectorField     (20 0 0);                 //Declare variable

internalField   uniform $vectorField;     //Use declared variable


scalarField     101328;                   //Declare variable

type            fixedValue;
value           uniform $scalarField;     //Use declared variable
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Macro expansion.
  - You can use macro expansion to duplicate and access variables in dictionaries

```
p                       // Declare/create the dictionary p
{
    solver              PCG;
    preconditioner      DIC;
    tolerance           1e-06;
    relTol              0;
}


$p;                     //To create a copy of the dictionary p
$p.solver;              //To access the variable solver in the dictionary p
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Inline calculations.
  - You can use the directive **#calc** to do inline calculations, the syntax is as follows:

```
X = 10.0;                          //Declare variable
Y = 3.0;                           //Declare variable

Z    #calc    "$X*$Y – 12.0";      //Do inline calculation. The result is
                                   //saved in the variable Z
```

- With inline calculations you can access all the mathematical functions available in C++.
- Macro expansions and inline calculations are really useful to parametrize dictionaries and avoid repetitive tasks.

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Instead of writing (the poor man's way):

```
leftWall
{
    type    fixedValue;
    value   uniform (0 0 0);
}
rightWall
{
    type    fixedValue;
    value   uniform (0 0 0);
}
topWall
{
    type    fixedValue;
    value   uniform (0 0 0);
}
```

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- You can write (the lazy way):

```
"(left|right|top)Wall"
{
    type    fixedValue;
    value   uniform (0 0 0);
}
```

- You could also try (even lazier):

```
".*Wall"
{
    type    fixedValue;
    value   uniform (0 0 0);
}
```

- OpenFOAM® understands the syntax of regular expressions (regex or regexp).

A deeper view to my first OpenFOAM® case setup

Dictionary files general features

- Switches: they are used to enable or disable a function or a feature in the dictionaries.
- Switches are logical values. You can use the following values:

false	**true**
off	**on**
no	**yes**
n	**y**
f	**t**
none	**true**

- You can find all the valid switches in the following file:

  OpenFOAM-4.x/src/OpenFOAM/primitives/bools/Switch/Switch.C

A deeper view to my first OpenFOAM® case setup

Solvers and utilities help

- If you need help about a solver or utility, you can use the option `-help`. For instance:
  - `$> icoFoam -help`

  will print some basic help and usage information about `icoFoam`

- Remember, you have the source code there so you can always check the original source.

A deeper view to my first OpenFOAM® case setup

Solvers and utilities help

- To get more information about the boundary conditions and post-processing utilities available in OpenFOAM®, please read the Doxygen documentation. Just look for the **Using OpenFOAM** section at the bottom of the page.
- If you did not compile the Doxygen documentation, you can access the information online, http://cpp.openfoam.org/v4/