Module 1

OpenFOAM® overview – First tutorial – Working our way in OpenFOAM®

Roadmap

OpenFOAM® brief overview

General description:

- OpenFOAM® stands for Open Source Field Operation and Manipulation.
- OpenFOAM® is first and foremost a C++ library used to solve partial differential equations (PDEs), and ordinary differential equations (ODEs).
- It comes with several ready-to-use or out-of-the-box solvers, pre-processing utilities, and post-processing utilities.
- It is licensed under the GNU General Public License (GPL).
    - That means it is freely available and distributed with the source code.
- It can be used in massively parallel computers. No need to pay for separate licenses.
- It is under active development.
- It counts with a wide-spread community around the world (industry, academia and research labs).

OpenFOAM® brief overview

Multi-physics simulation capabilities:

- OpenFOAM® has extensive multi-physics simulation capabilities, among others:
    - Computational fluid dynamics (incompressible and compressible flows).
    - Computational heat transfer and conjugate heat transfer.
    - Combustion and chemical reactions.
    - Multiphase flows and mass transfer.
    - Particle methods (DEM, DSMC, MD) and lagrangian particles tracking.
    - Stress analysis and fluid-structure interaction.
    - Rotating frames of reference, arbitrary mesh interface, dynamic mesh handling, and adaptive mesh refinement.
    - 6 DOF solvers, ODE solvers, computational aero-acoustics, computational electromagnetics, computational solid mechanics, MHD.

OpenFOAM® brief overview

Physical modeling capabilities:

- OpenFOAM® comes with many physical models, among others:
  - Extensive turbulence modeling capabilities (RANS, DES and LES).
  - Transport/rheology models. Newtonian and non-Newtonian viscosity models.
  - Thermophysical models and physical properties for liquids and gases.
  - Source terms models.
  - Lagrangian particle models.
  - Interphase momentum transfer models for multiphase flows.
  - Combustion, flame speed, chemical reactions, porous media, radiation, phase change.

OpenFOAM® brief overview

Under the hood you will find the following:

- Finite Volume Method (FVM) based solver.
- Collocated polyhedral unstructured meshes.
- Second order accuracy in space and time. Many discretization schemes available (including high order methods).
- Steady and transient solvers available.
- Pressure-velocity coupling via segregated methods (SIMPLE and PISO).
- But coupled solvers are under active development.
- Massive parallelism through domain decomposition.
- It comes with its own mesh generation tools.
- It also comes with many mesh manipulation and conversion utilities.
- It comes with many post-processing utilities.
- All components implemented in library form for easy re-use.

OpenFOAM® brief overview

OpenFOAM® vs. Commercial CFD applications:

- OpenFOAM® capabilities mirror those of commercial CFD applications.
- The main differences with commercial CFD applications are:
  - There is no native GUI.
    - Knowing your way around the Linux bash shell is extremely useful.
  - It does not come with predefined setups.
    - The users need to have a basic understanding of the CFD basics and be familiar with OpenFOAM® command line interface (CLI).
  - It is not a single executable.
    - Depending of what you are looking for, you will need to execute a specific application from the CLI.
  - It is not well documented, but the source code is available.
  - Access to complete source = no black magic. But to understand the source code you need to know object-oriented programming and C++.
  - Solvers can be tailored for a specific need, therefore OpenFOAM® is ideal for research and development.
  - It is free and has no limitation on the number of cores you can use.

OpenFOAM® brief overview

Developing new solvers (in case you need it):

- As the user has complete access to the source code, she/he has total freedom to modify existing solvers or use them as the starting point for new solvers.
- New solvers can be easily implemented using OpenFOAM® high level programming, *e.g.*:

$$\frac{\partial T}{\partial t} + \nabla \cdot (\phi T) - \nabla \cdot (\nu \nabla T) = 0 \longrightarrow$$

```
solve
(
        fvm::ddt(T)
      + fvm::div(phi,T)
      -  fvm::laplacian(nu,T)
        ==
        0
);
```

Correspondence between the implementation and the original equation is clear.

OpenFOAM® brief overview

OpenFOAM® is an excellent piece of C++ and software engineering. Decent piece of CFD code.

H. Jasak

Roadmap

1. ~~OpenFOAM® brief overview~~
2. **OpenFOAM® directory organization**
3. Directory structure of an application/utility
4. Applications/utilities in OpenFOAM®
5. Directory structure of an OpenFOAM® case
6. Running my first OpenFOAM® case setup blindfold
7. A deeper view to my first OpenFOAM® case setup
8. 3D Dam break – Free surface flow
9. Flow past a cylinder – From laminar to turbulent flow

OpenFOAM® directory organization

```
$WM_PROJECT_DIR
├── Allwmake
├── applications
├── bin
├── COPYING
├── doc
├── etc
├── platforms
├── README.org
├── src
├── test
├── tutorials
└── wmake
```

If you installed OpenFOAM® in the default location, the environment variable **`$WM_PROJECT_DIR`** should point to the following directory (depending on the installed version):

`$HOME/OpenFOAM/OpenFOAM-9`

or

`$HOME/OpenFOAM/OpenFOAM-dev`

In this directory you will find all the files containing OpenFOAM® installation.

In this directory you will also find additional files (such as *`README.org`*, *`COPYING`*, etc.), but the most important one is *`Allwmake`*, which compiles OpenFOAM®.

OpenFOAM® directory organization

```
$WM_PROJECT_DIR
├── Allwmake
├── applications
├── bin
├── COPYING
├── doc
├── etc
├── platforms
├── README.org
├── src
├── test
├── tutorials
└── wmake
```

OpenFOAM® environment variables

The entries starting with the symbol `$` are environment variables. You can find out the value of an environment variable by echoing its value, for example:

```
$> echo $WM_PROJECT_DIR
```

will print out the following information on the terminal,

```
$HOME/OpenFOAM/OpenFOAM-9
```

To list all the environment variables, type in the terminal window,

```
$> env
```

To list all the environment variables related to OpenFOAM®, type in the terminal:

```
$> env | grep -i “OpenFOAM”
```

OpenFOAM® directory organization

```
$WM_PROJECT_DIR
├── Allwmake
├── applications
├── bin
├── COPYING
├── doc
├── etc
├── platforms
├── README.org
├── src
├── test
├── tutorials
└── wmake
```

OpenFOAM® aliases

You can go to any of these directories by using the predefined aliases set by OpenFOAM® (refer to *$WM_PROJECT_DIR/etc/config.sh/aliases* or *$WM_PROJECT_DIR/etc/config.csh/aliases*).

Just to name a few of the aliases defined:

```
alias foam='cd $WM_PROJECT_DIR'
alias app='cd $FOAM_APP'
alias src='cd $FOAM_SRC'
alias tut='cd $FOAM_TUTORIALS'
```

For a complete list type `alias` in the terminal.

To list all the aliases related to OpenFOAM®, type in the terminal:

```
$> alias | grep -i "FOAM"
```

OpenFOAM® directory organization

```
$WM_PROJECT_DIR
├── Allwmake
├── applications
├── bin
├── COPYING
├── doc
├── etc
├── platforms
├── README.org
├── src
├── test
├── tutorials
└── wmake
```

Let us study each directory inside
`$WM_PROJECT_DIR`

- Any modification you add to the source code in **`WM_PROJECT_DIR`** will affect the whole library.
- Unless you know what are you doing, do not modify anything in the original installation (**`$WM_PROJECT_DIR`**), except for updates!

OpenFOAM® directory organization

The **`applications`** directory

```
$WM_PROJECT_DIR/applications
├── Allwmake
├── solvers
├── test
└── utilities
```

Let us visit the **`applications`** directory. Type in the terminal `app` or
`$> cd $WM_PROJECT_DIR/applications`. You will find the following sub-directories:

- **`solvers`**, which contains the source code for the distributed solvers.
- **`test`**, which contains the source code of several test cases that show the usage of some of the OpenFOAM® classes.
- **`utilities`**, which contains the source code for the distributed utilities.

There is also an *`Allwmake`* script, which will compile all the content of **`solvers`** and **`utilities`**. To compile the test cases in **`test`**, go to the desired test case directory and compile it by typing `wmake`.

OpenFOAM® directory organization

The **bin** directory

```
$WM_PROJECT_DIR/bin/
├── foamCleanCase
├── foamCleanTutorials
├── foamCloneCase
├── foamInfo
├── foamJob
├── foamLog
├── foamMonitor
├── foamNew
├── foamNewApp
├── foamNewBC
├── foamNewFunctionObject
├── paraFoam
├── ...
└── tools
```

The directory tree is not complete ⚠

Let us visit the **bin** directory:

- The **bin** directory contains many shell scripts, such as *foamNew*, *foamLog*, *foamJob*, *foamNewApp*, etc.
- This directory also contains the script *paraFoam* that will launch paraView.
- The `foamInfo` command prints detailed information about an application, a script, or a model (including boundary conditions, function objects and fvModels).
- To get more information type in the terminal
  - `$> foamInfo -help`

OpenFOAM® directory organization

The **doc** directory

```
$WM_PROJECT_DIR/doc/
├── Allwmake
├── codingStyleGuide.org
├── Doxygen
├── Guides
└── tools
```

Let us visit the **doc** directory:

- The **doc** directory contains the documentation of OpenFOAM®, namely; user guide, programmer's guide and Doxygen generated documentation in html format.
- The Doxygen documentation needs to be compiled by typing `Allwmake doc` in **$WM_PROJECT_DIR**.
- You can access the Doxygen documentation online, http://cpp.openfoam.org/v9

OpenFOAM® directory organization

The **`etc`** directory

```
$WM_PROJECT_DIR/etc/
├── bashrc
├── caseDicts
├── cellModels
├── codeTemplates
├── config.csh
├── config.sh
├── controlDict
├── cshrc
├── paraFoam
├── README.org
├── templates
└── thermoData
```

Let us visit the **`etc`** directory:

- The **`etc`** directory contains the environment files, global OpenFOAM® instructions, templates, and the default thermochemical database *`thermoData/thermoData`*

- In the directory **`caseDicts`**, you will find many templates related to the input files used to setup a case in OpenFOAM®. We recommend you take some time and explore these files.

- It also contains the super dictionary *`controlDict`*, where you can set several debug flags and the defaults units.

OpenFOAM® directory organization

The **platforms** directory

```
$WM_PROJECT_DIR/platforms/
├── linux64GccDPInt32Opt
│     ├── applications
│     ├── bin
│     ├── lib
│     └── src
└── linux64GccDPInt32OptSYSTEMOPENMPI
      └── src
```

Let us visit the **platforms** directory.

- This directory contains the binaries generated when compiling the **applications** directory and the libraries generated by compiling the source code in the **src** directory.
- After compilation, the binaries will be located in the directory **$WM_PROJECT_DIR/platforms/linux64GccDPInt32OptSYSTEMOPENMPI/bin $WM_PROJECT_DIR/platforms/linux64GccDPOpt/lib)**.
- After compilation, the libraries will be located in the directory **$WM_PROJECT_DIR/platforms/linux64GccDPInt32OptSYSTEMOPENMPI/lib**

OpenFOAM® directory organization

The **src** directory

```
$WM_PROJECT_DIR/src
├── Allwmake
├── combustionModels
├── finiteVolume
├── fvModels
├── lagrangian
├── MomentumTransportModels
├── ...
├── OpenFOAM
├── parallel
├── sampling
├── sixDoFRigidBodyMotion
├── thermophysicalModels
├── transportModels
└── waves
```

Let us visit the **src** directory. Type in the terminal `src` or `$> cd $WM_PROJECT_DIR/src`

- This directory contains the source code for all the foundation libraries, this is the core of OpenFOAM®.
- It is divided in different subdirectories and each of them can contain several libraries.

A few interesting directories are:

- **OpenFOAM**. This core library includes the definitions of the containers used for the operations, the field definitions, the declaration of the mesh and mesh features such as zones and sets.

The directory tree is not complete ⚠

OpenFOAM® directory organization

The **src** directory

```
$WM_PROJECT_DIR/src
├── Allwmake
├── combustionModels
├── finiteVolume
├── fvModels
├── lagrangian
├── MomentumTransportModels
├── ...
├── OpenFOAM
├── parallel
├── sampling
├── sixDoFRigidBodyMotion
├── thermophysicalModels
├── transportModels
└── waves
```

A few interesting directories are:

- **`finiteVolume`**. This library provides all the classes needed for the finite volume discretization, such as mesh handling, finite volume discretization operators (divergence, laplacian, gradient, fvc/fvm and so on), and boundary conditions. In the directory **`finiteVolume/lnInclude`** you also find the very important file *`fvCFD.H`*, which is included in most applications.
- **`MomentumTransportModels`**, which contains many turbulence models.
- **`sixDoFRigidBodyMotion`**. This core library contains the solver for rigid body motion.
- **`transportModels`**. This core library contains many transport models.

The directory tree is not complete ⚠

OpenFOAM® directory organization

The **tutorials** directory

```
$WM_PROJECT_DIR/tutorials/
├── Allclean
├── Allrun
├── Alltest
├── basic
├── combustion
├── compressible
├── discreteMethods
├── DNS
├── electromagnetics
├── financial
├── heatTransfer
├── incompressible
├── lagrangian
├── mesh
├── multiphase
├── resources
└── stressAnalysis
```

Let us visit the **tutorials** directory. Type in the terminal `tut` or

```
$> cd $WM_PROJECT_DIR/tutorials
```

- The **tutorials** directory contains example cases for each solver. These are the tutorials distributed with OpenFOAM®.
- They are organized according to the physics involved.
- Use these tutorials as the starting point to develop you own cases.
- A word of caution, do not use these tutorials as best practices, they are there just to show how to use the applications.

OpenFOAM® directory organization

The **wmake** directory

```
$WM_PROJECT_DIR/wmake/
├── makefiles
├── platforms
├── rules
├── scripts
├── src
├── wclean
├── wcleanLnIncludeAll
├── wcleanPlatform
├── wdep
├── wmake
├── ...
├── wmakeFilesAndOptions
├── wmakeLnInclude
├── wmakeLnIncludeAll
├── ...
└── wrmo
```

The directory tree is not complete ⚠

Let us visit the **`wmake`** directory.

- OpenFOAM® uses a special make command: `wmake`
- `wmake` understands the file structure in OpenFOAM® and uses default compiler directives set in this directory.
- If you add a new compiler name in OpenFOAM® *`bashrc`* file, you should also tell `wmake` how to interpret that name.
- In **`wmake/rules`** you will find the default settings for the available compilers.
- In this directory, you will also find a few scripts that are useful when organizing your files for compilation, or for cleaning up.

OpenFOAM® directory organization

OpenFOAM® user directory

```
$HOME/OpenFOAM/
├── $WM_PROJECT_USER_DIR   ←
└── $WM_PROJECT_DIR
```

- Let us now study OpenFOAM® user directory **`$WM_PROJECT_USER_DIR`**
- When working with OpenFOAM®, you can put your files wherever you want.
- To keep things in order, it is recommended to put your cases in your OpenFOAM® user directory or **`$WM_PROJECT_USER_DIR`**.
- It is also recommended to put your modified solvers, utilities, and libraries in your OpenFOAM® user directory or **`$WM_PROJECT_USER_DIR`**. This is done so you do not modify anything in the original installation.
- If you followed the standard installation instructions, the variable **`$WM_PROJECT_USER_DIR`** should point to the directory **`$HOME/OpenFOAM/USER_NAME-9`**, where **USER_NAME** is the name of the user (*e.g.*, johnDoe).

OpenFOAM® directory organization

Looking for information in OpenFOAM® source code

- To locate files, you can use the `find` command.
- If you want to locate a directory inside **`$WM_PROJECT_DIR`** that contains the string **fvPatch** in its name, you can proceed as follows,
  - $$\texttt{\$> find }\underbrace{\texttt{\$WM\_PROJECT\_DIR}}_{\text{Where to look for}}\;\underbrace{\texttt{-type d}}_{\text{Look for directories}}\;\underbrace{\texttt{-name}}_{\text{Case sensitive}}\;\underbrace{\texttt{“*fvPatch*”}}_{\text{Look for this (using wildcards)}}$$
- If you want to locate a file inside **`$WM_PROJECT_DIR`** that contains the string **fvPatch** in its name, you can proceed as follows,
  - $$\texttt{\$> find }\underbrace{\texttt{\$WM\_PROJECT\_DIR}}_{\text{Where to look for}}\;\underbrace{\texttt{-type f}}_{\text{Look for files}}\;\underbrace{\texttt{-name}}_{\text{Case sensitive}}\;\underbrace{\texttt{“*fvPatch*”}}_{\text{Look for this (using wildcards)}}$$
- If you want to find a string inside a file, you can use the `grep` command.
- For example, if you want to find the string **LES** inside all the files within the directory **`$FOAM_SOLVERS`**, you can proceed as follows,
  - `$> grep -r -n “LES” $FOAM_SOLVERS`

  The argument `-r` means recursive and `-n` will output the line number.

OpenFOAM® directory organization

Looking for information in OpenFOAM® source code

- Dictionaries are input files required by OpenFOAM®.
- As you can imagine, there are many dictionaries in OpenFOAM®. The easiest way to find all of them is to do a local search in the installation directory as follows,
- For instance, if you are interested in finding all the files that end with the **Dict** word in the **`tutorials`** directory, in the terminal type:
  - `$> find $FOAM_TUTORIALS -name “*Dict”`
    (Case sensitive search)
  - `$> find $FOAM_TUTORIALS –iname ‘*dict’`
    (Non-case sensitive search)

- When given the search string, you can use single quotes ‘ ’ or double-quotes “ ” (do not mixed them).
- We recommend to use single quotes, but it is up to you.

OpenFOAM® directory organization

Looking for information in OpenFOAM® source code

- A few more advanced commands to find information in your OpenFOAM® installation.
  - To find which tutorial files use the boundary condition **slip**:
    - `$> find $FOAM_TUTORIALS -type f | xargs grep -sl 'slip'`

      This command will look for all files inside the directory **`$FOAM_TUTORIALS`**, then the output is used by grep to search for the string **slip**.
  - To find where the source code for the boundary condition **slip** is located:
    - `$> find $FOAM_SRC -name "*slip*"`
  - To find what applications do not run in parallel:
    - `$> find $WM_PROJECT_DIR -type f | xargs grep -sl 'noParallel'`
- OpenFOAM® understands REGEX syntax.

OpenFOAM® directory organization

Environment variables

- Remember, OpenFOAM® uses its own environment variables.
- OpenFOAM® environment settings are contained in the **`OpenFOAM-9/etc`** directory.
- If you installed OpenFOAM® in the default location, they should be in:
  - **`$HOME/OpenFOAM/OpenFOAM-9/etc`**
- If you are running bash or ksh (if in doubt type in the terminal `echo $SHELL`), you sourced the *`$WM_PROJECT_DIR/etc/bashrc`* file by adding the following line to your *`$HOME/.bashrc`* file:
  - `source $HOME/OpenFOAM/OpenFOAM-9/etc/bashrc`

- By sourcing the file *`$WM_PROJECT_DIR/etc/bashrc`*, we start to use OpenFOAM® environment variables.
- By default, OpenFOAM® uses the system compiler and the system MPI compiler.
- When you use OpenFOAM® you are using its environment settings, that is, its path to libraries and compilers. So if you are doing software developing, and you are having compilation problems due to conflicting libraries or missing compilers, try to unload OpenFOAM® environment variables

Roadmap

1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. **Directory structure of an application/utility**
4. Applications/utilities in OpenFOAM®
5. Directory structure of an OpenFOAM® case
6. Running my first OpenFOAM® case setup blindfold
7. A deeper view to my first OpenFOAM® case setup
8. 3D Dam break – Free surface flow
9. Flow past a cylinder – From laminar to turbulent flow

Directory structure of an OpenFOAM® application/utility

Directory structure of a general solver

```
$WM_PROJECT_DIR/applications/solvers/solverName/
├── createFields.H
├── appName.C
└── Make
    ├── files
    └── options
```

The **`$WM_PROJECT_DIR/applications/solvers/solverName/`** directory contains the source code of the solver.

- *`solverName/appName.C`*: is the actual source code of the solver.
- *`solverName/createFields.H`*: declares all the field variables and initializes the solution.
- The **`Make`** directory contains compilation instructions.
    - *`Make/files`*: names all the source files `(.C)`, it specifies the `solverName` name and location of the output file.
    - *`Make/options`*: specifies directories to search for include files and libraries to link the solver against.

Directory structure of an OpenFOAM® application/utility

Directory structure of a general utility

```
$WM_PROJECT_DIR/applications/utilities/utilityName/
├── utility_dictionary
├── utilityName.C
├── header_files.H
└── Make
    ├── files
    └── options
```

The **`$WM_PROJECT_DIR/utilities/utilityName/`** directory contains the source code of the utility.

- *`utilityName/utilityName.C`*: is the actual source code of the application.
- *`utilityName/header_files.H`*: header files required to compile the application.
- *`utilityName/utility_dictionary`*: application's master dictionary.
- The **`Make`** directory contains compilation instructions.
    - *`Make/files`*: names all the source files `(.C)`, it specifies the `utilityName` name and location of the output file.
    - *`Make/options`*: specifies directories to search for include files and libraries to link the solver against.

Directory structure of an OpenFOAM® application/utility

- For your own solvers and utilities, it is recommended to put the source code in the directory **`$WM_PROJECT_USER_DIR`** following the same structure as in **`$WM_PROJECT_DIR/applications/solvers`** and **`$WM_PROJECT_DIR/utilities/`**.
- Also, you will need to modify the files *`Make/files`* and *`Make/options`* to point to the new name and location of the compiled binaries and libraries to link the solver against.
- You can do anything you want to your own copies, so you do not risk messing things up.
- This is done so you do not modify anything in the original installation, except for updates!

Roadmap

1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. ~~Directory structure of an application/utility~~
4. **Applications/utilities in OpenFOAM®**
5. Directory structure of an OpenFOAM® case
6. Running my first OpenFOAM® case setup blindfold
7. A deeper view to my first OpenFOAM® case setup
8. 3D Dam break – Free surface flow
9. Flow past a cylinder – From laminar to turbulent flow

Applications/utilities in OpenFOAM®

- OpenFOAM® is not a single executable.
- Depending of what you want to do, you will need to use a specific application and there are many of them.
- If you are interested in knowing all the solvers, utilities, and libraries that come with your OpenFOAM® distribution, read the applications and libraries section in the user guide (chapter 3).
- In the directory **`$WM_PROJECT_DIR/doc`** you will find the documentation in pdf format.
- You can also access the online user guide. Go to the link http://cfd.direct/openfoam/user-guide/#contents, then go to chapter 3 (applications and libraries).
- If you want to get help on how to run an application, type in terminal

```
1. | $> application_name -help
```

- The option `-help` will not run the application; it will only show all the options available.
- You can also get all the help you want from the source code.

Applications/utilities in OpenFOAM®

- You will find all the applications in the directory **`$FOAM_SOLVERS`** (you can use the alias `sol` to go there).
- You will find all the utilities in the directory **`$FOAM_UTILITIES`** (you can use the alias `util` to go there).
- For example, in the directory **`$FOAM_SOLVERS`**, you will find the directories containing the source code for the solvers available in the OpenFOAM® installation (version 9):

  - **basic**
  - **combustion**
  - **compressible**
  - **discreteMethods**
  - **DNS**
  - **electromagnetics**
  - **financial**
  - **heatTransfer**
  - **incompressible**
  - **lagrangian**
  - **multiphase**
  - **stressAnalysis**

- The solvers are subdivided according to the physics they address.
- The utilities are also subdivided in a similar way.

Applications/utilities in OpenFOAM®

- For example, in the sub-directory **`incompressible`** you will find several sub-directories containing the source code of the following solvers:

  - **adjointShapeOptimizationFoam**
  - **boundaryFoam**
  - **icoFoam**
  - **nonNewtonianIcoFoam**
  - **pimpleFoam**
  - **pisoFoam**
  - **shallowWaterFoam**
  - **simpleFoam**

- Inside each directory, you will find a file with the extension *`*.C`* and the same name as the directory. This is the main file, where you will find the top-level source code and a short description of the solver or utility.
- For example, in the file *`incompressible/icoFoam/icoFoam.C`* you will find the following description:

Transient solver for incompressible, laminar flow of Newtonian fluids.

Applications/utilities in OpenFOAM®

- Remember, OpenFOAM® is not a single executable.
- You will need to find the solver or utility that best fit what you want to do.
- A few solvers that we will use during this course:
  - **icoFoam**: laminar incompressible unsteady solver. Be careful, do not use this solver for production runs as it has many limitations.
  - **simpleFoam**: incompressible steady solver for laminar/turbulent flows.
  - **pimpleFoam**: incompressible unsteady solver for laminar/turbulent flows.
  - **rhoSimpleFoam**: compressible steady solver for laminar/turbulent flows.
  - **rhoPimpleFoam**: unsteady compressible solver for (laminar/turbulent flows.
  - **interFoam**: unsteady multiphase solver for separated flows using the VOF method (laminar and turbulent flows).
  - **laplacianFoam**: Laplace equation solver.
  - **potentialFoam**: potential flow solver.
  - **scalarTransportFoam**: steady/unsteady general transport equation solver.

Applications/utilities in OpenFOAM®

- Take your time and explore the source code.
- Also, while exploring the source code be careful not to add unwanted modifications in the original installation.
- If you modify the source code, be sure to do the modifications in your user directory instead of the main source code.

Roadmap

1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. ~~Directory structure of an application/utility~~
4. ~~Applications/utilities in OpenFOAM®~~
5. **Directory structure of an OpenFOAM® case**
6. Running my first OpenFOAM® case setup blindfold
7. A deeper view to my first OpenFOAM® case setup
8. 3D Dam break – Free surface flow
9. Flow past a cylinder – From laminar to turbulent flow

Directory structure of an OpenFOAM® case

Directory structure of a general case

```
case_name
├── 0
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── system
│   ├── controlDict
│   ├── fvSchemes
│   └── fvSolution
└── time_directories
```

- OpenFOAM® uses a very particular directory structure for running cases.
- You should always follow the directory structure, otherwise, OpenFOAM® will complain.
- To keep everything in order, the case directory is often located in the path **`$WM_PROJECT_USER_DIR/run`**.
- This is not compulsory but highly advisable. You can copy the case files anywhere you want.
- The name of the case directory is given by the user (**do not use white spaces or strange symbols**).
- Depending of the solver or application you would like to use, you will need different files in each sub-directory.
- Remember, you always run the applications and utilities in the **top level** of the case directory (**the directory with the name case_name**). Not in the directory **`system`**, not in the directory **`constant`**, not in the directory **`0`**.

Directory structure of an OpenFOAM® case

Directory structure of a general case

```
case_name
├── 0
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── system
│    ├── controlDict
│    ├── fvSchemes
│    └── fvSolution
└── time_directories
```

`case_name`: the name of the case directory is given by the user (**do not use white spaces or strange symbols**).

This is the top-level directory, where you run the applications and utilities.

`system`: contains run-time control and solver numerics.

`constant`: contains physical properties, turbulence modeling properties, advanced physics and so on.

`constant/polyMesh`**:** contains the polyhedral mesh information.

`0`: contains boundary conditions (BC) and initial conditions (IC).

`time_directories`: contains the solution and derived fields. These directories are created by the solver automatically and according to the preset saving frequency, *e.g.*, 1, 2, 3, 4, … , 100.

Roadmap

Running my first OpenFOAM® case setup blindfold

Before we start – Always remember the directory structure

```
case_name
├── 0
├── constant
│   └── polyMesh
├── system
└── time_directories
```

- To keep everything in order, the case directory is often located in the path **`$WM_PROJECT_USER_DIR/run`**.
- This is not compulsory but highly advisable, you can put the case in any directory of your preference.
- The name of the case directory if given by the user (do not use white spaces).
- You run the applications and utilities in the top level of this directory.
- The directory **`system`** contains run-time control and solver numerics.
- The directory **`constant`** contains physical properties, turbulence modeling properties, advanced physics and so on.
- The directory **`constant/polyMesh`** contains the polyhedral mesh information.
- The directory **`0`** contains boundary conditions (BC) and initial conditions (IC).

Running my first OpenFOAM® case setup blindfold

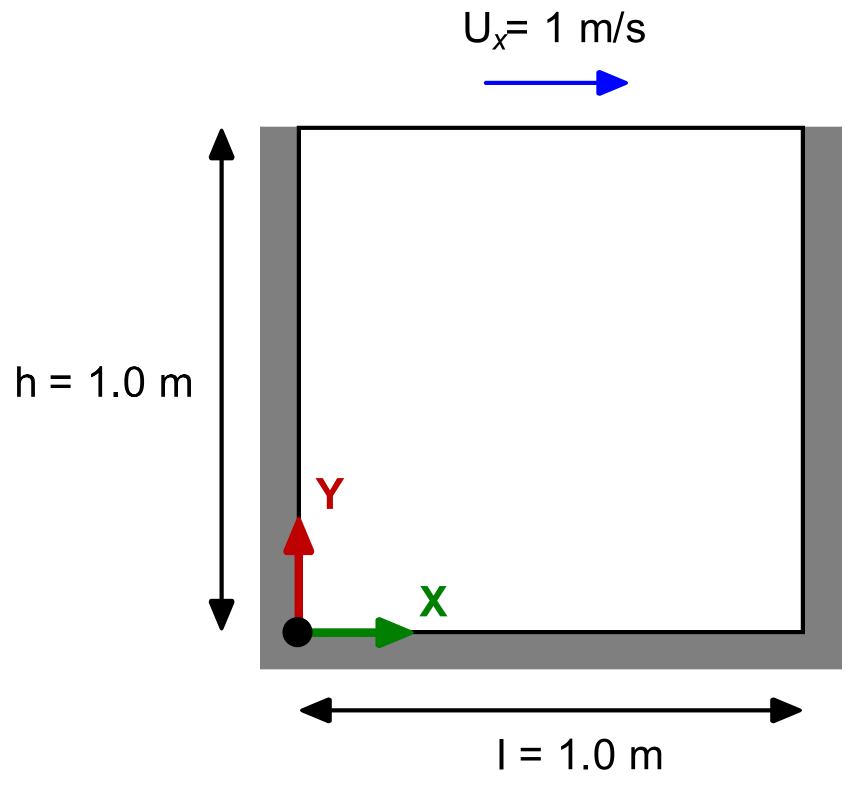
Before we start – Setting OpenFOAM® cases

- As you will see, it is quite difficult to remember all the dictionary files needed to run each application.
- It is even more difficult to recall the compulsory and optional entries of each input file.
- When setting a case from scratch in OpenFOAM®, what you need to do is find a tutorial or a case that close enough does what you want to do and then you can adapt it to your physics.
- Having this in mind, you have two sources of information:
    - **`$WM_PROJECT_DIR/tutorials`**
      (The tutorials distributed with OpenFOAM®)
    - **`$PTOFC`**
      (The tutorials used during this training)

- If you use a GUI, things are much easier. However, OpenFOAM® does not come with a native GUI interface.
- We are going to do things in the hard way (and maybe the smart way), we are going to use the Linux terminal

Running my first OpenFOAM® case setup blindfold

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



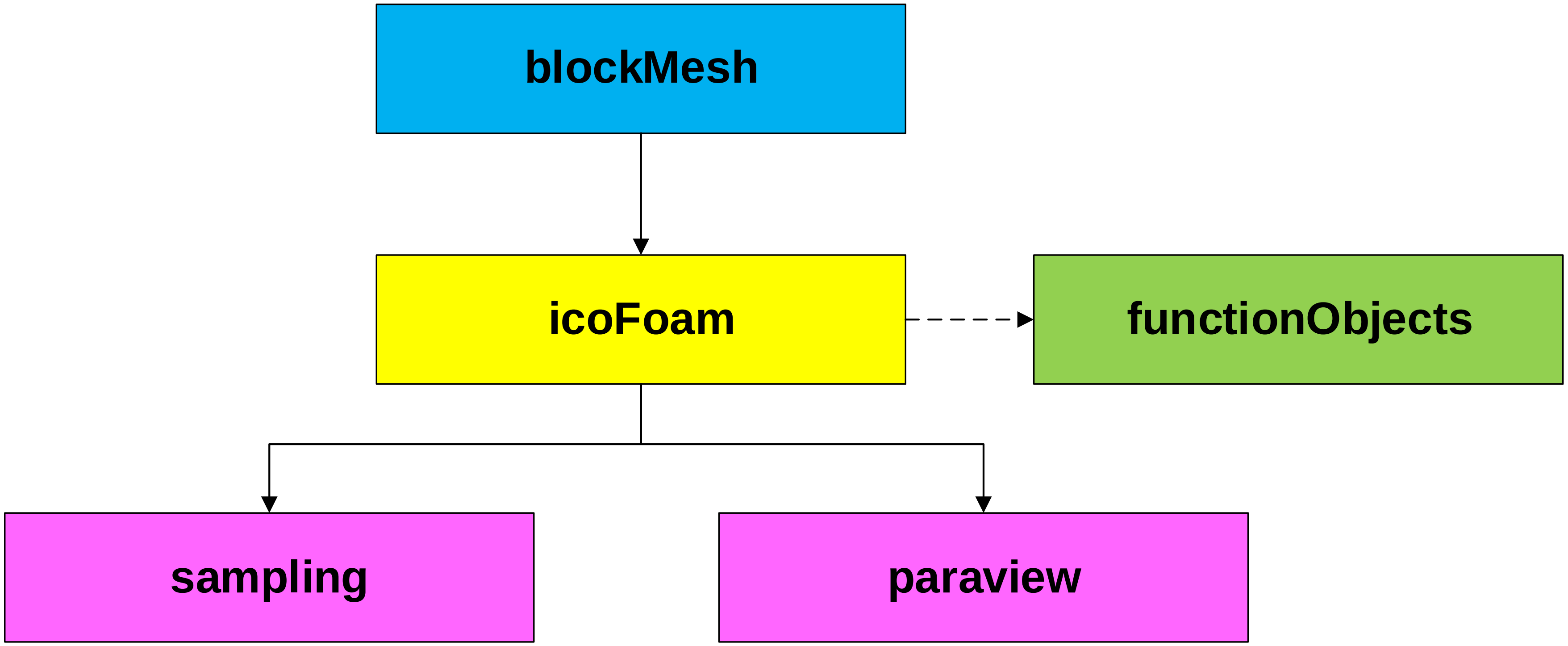
▨ **No-slip wall**

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 easily extended to 3D.
- 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 dictionary files have been already preset.

Running my first OpenFOAM® case setup blindfold

Workflow of the case



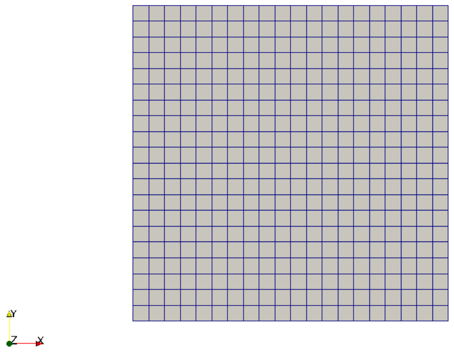
Running my first OpenFOAM® case setup blindfold

A word of caution about the solver icoFoam

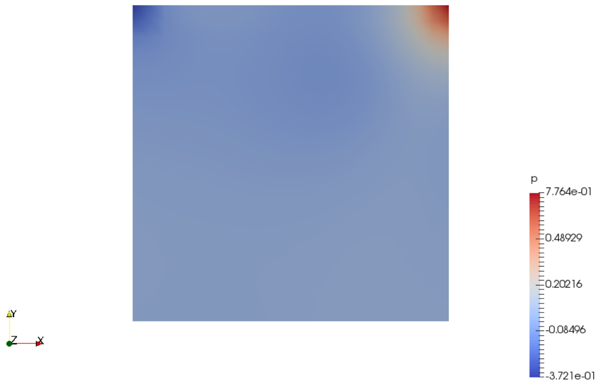
- The solver `icoFoam` is targeted for laminar incompressible unsteady solver.
- We do not recommend the use of this solver for production runs as it has limited post-processing features and no modeling capabilities.
- Instead of using `icoFoam`, you are better of with `pisoFoam` or `pimpleFoam`.

Running my first OpenFOAM® case setup blindfold

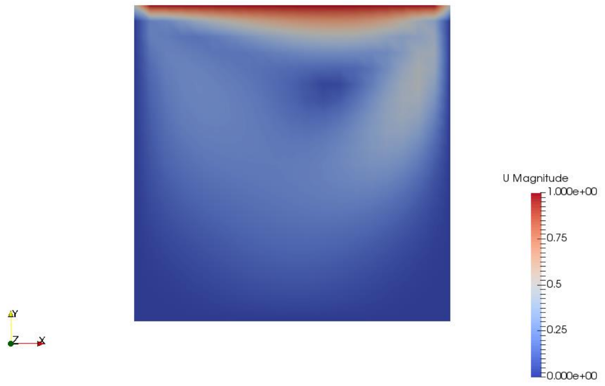
At the end of the day, you should get something like this



Mesh (very coarse and 2D)



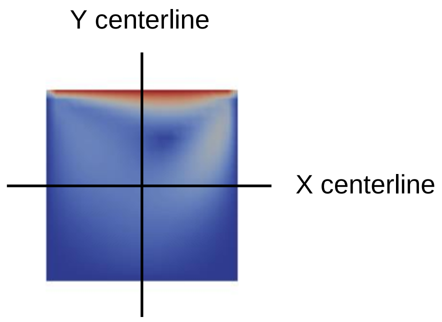
Pressure field (relative pressure)



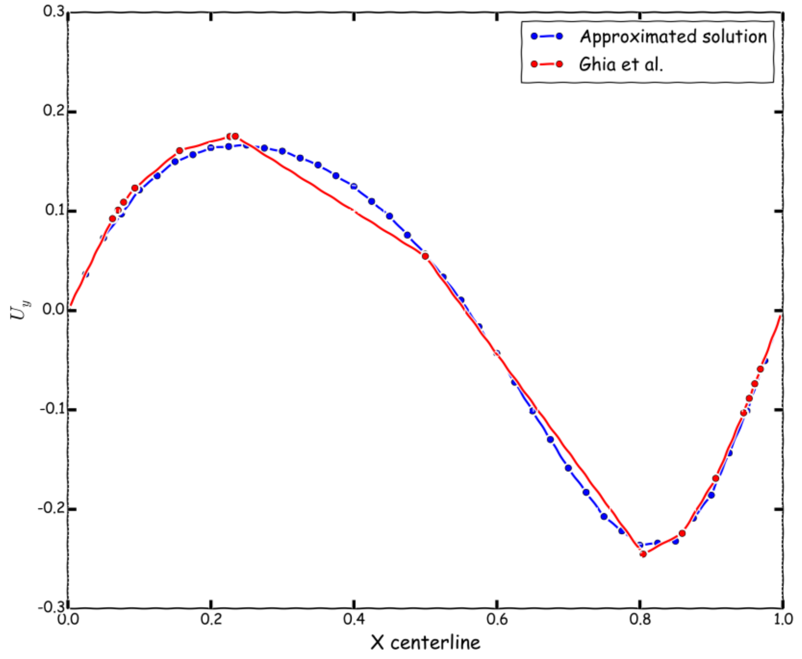
Velocity magnitude field

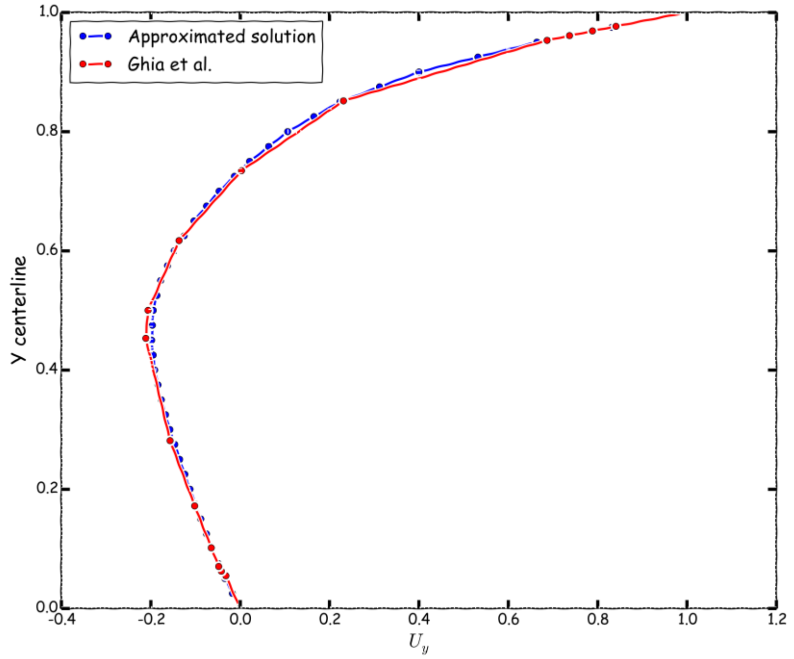
Running my first OpenFOAM® case setup blindfold

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 setup blindfold

- Let us run our first case. Go to the directory:

```
$PTOFC/101OF/cavity2D
```

- In the case directory, you will find a few scripts with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on.
- These scripts can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver`
- These scripts are human-readable, and we highly recommend you open them, get familiar with the steps, and type the commands in the terminal. In this way, you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, run the cases automatically using these scripts.
- In the case directory, you will also find the `README.FIRST` file. In this file, you will find some additional comments.

Running my first OpenFOAM® case setup blindfold

Loading OpenFOAM® environment

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

- To source OpenFOAM®**,** type in the terminal:
  - `$> of9`

- To use PyFoam (a plotting utility) you will need to source it. Type in the terminal:
  - `$> anaconda3`

- Remember, every time you open a new terminal window you need to source OpenFOAM® and PyFoam.
- Also, you might need to load OpenFOAM® again after loading 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 setup blindfold

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 quantitative post-processing and qualitative visualization using `paraFoam` and OpenFOAM® utilities.
- 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.
- A final word, we are going to use the solver `icoFoam` but have in mind that this is a very basic solver with no modeling capabilities and limited post-processing features.
- Therefore, is better to use `pisoFoam` or `pimpleFoam` which are equivalent to `icoFoam` but with many more features.

Running my first OpenFOAM® case setup blindfold

Running the case blindfold

- Let us run this case blindfold.
- Later we will study in detail each file and directory.
- Remember, the variable `$PTOFC` is pointing to the path where you unpacked the tutorials.
- You can create this environment variable or write down the path to the directory.
- In the terminal window type:

1.
```
$> cd $PTOFC/101OF/cavity
```
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 setup blindfold

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 (this step is optional). 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 (the `latestTime` flag). 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 setup blindfold

Crash introduction to paraFoam

Menu Bar

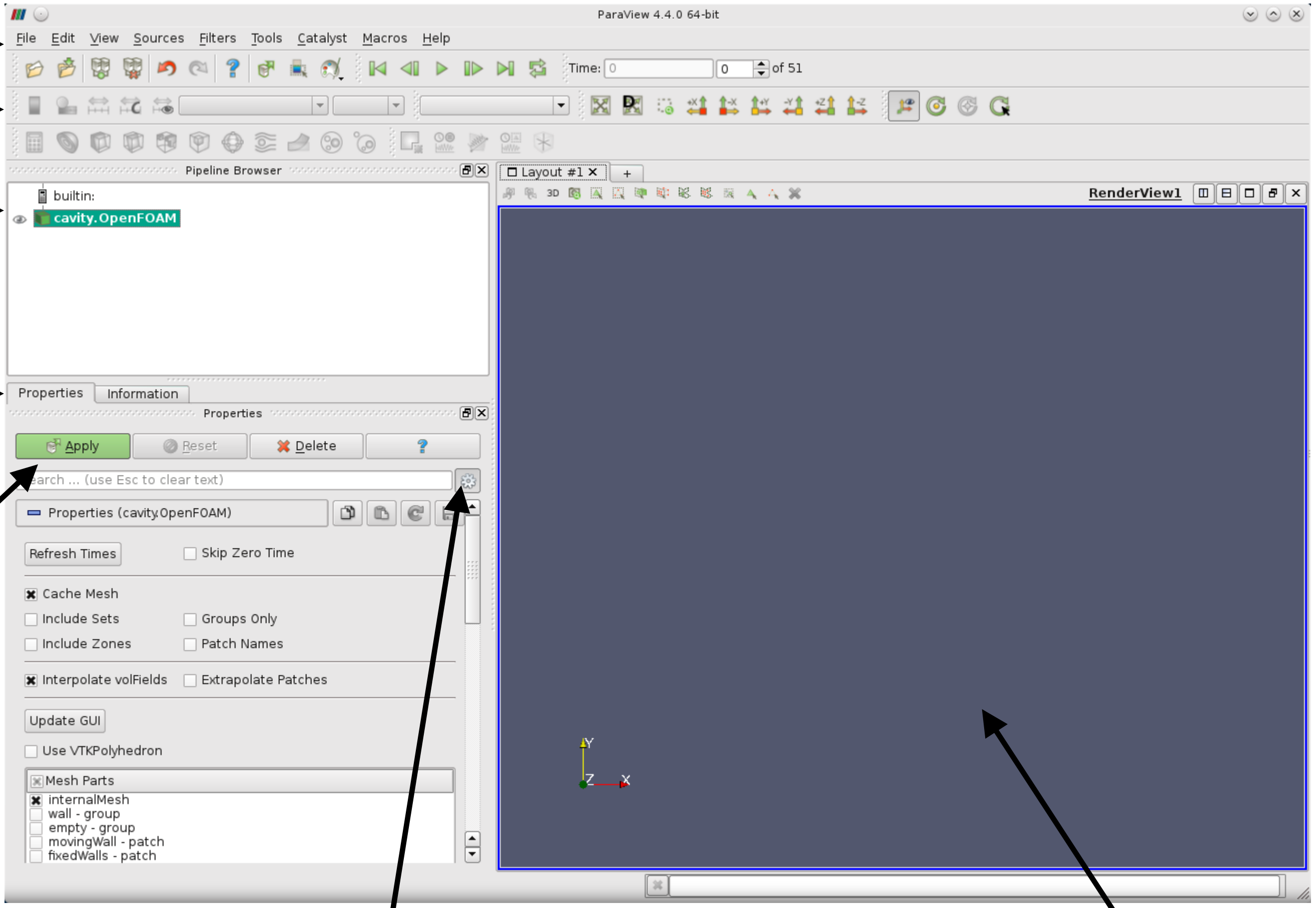
Toolbars

Pipeline Browser

Properties panel

`Apply button`

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



- [✓] Cache Mesh
- [ ] Skip Zero Time
- [ ] Include Sets
- [ ] Groups Only
- [ ] Include Zones
- [ ] Patch Names
- [✓] Interpolate volFields
- [ ] Extrapolate Patches
- [ ] Use VTKPolyhedron

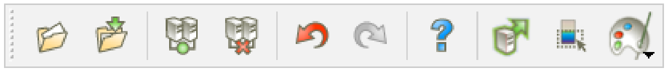
Mesh Parts
- [✓] internalMesh
- [ ] wall - group
- [ ] empty - group
- [ ] movingWall - patch
- [ ] fixedWalls - patch

Advanced Toggle

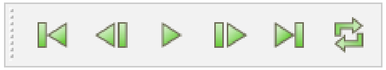
3D View/Canvas

Running my first OpenFOAM® case setup blindfold

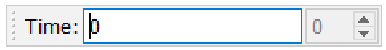
Crash introduction to paraFoam – Toolbars



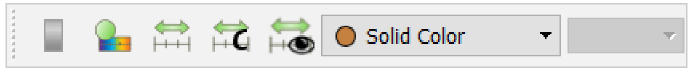
- Main Controls



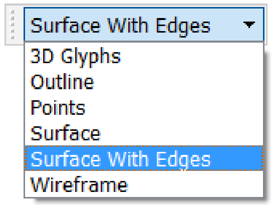
- VCR Controls (animation controls)



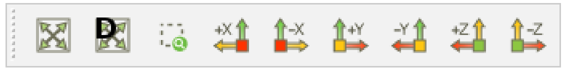
- Current Time Controls



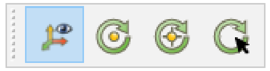
- Active Variable Controls



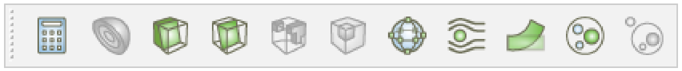
- Representation Toolbar



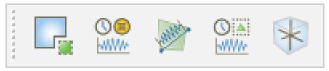
- Camera Controls (view orientation)



- Center Axes Controls



- Common Filters



- Data Analysis Toolbar

Running my first OpenFOAM® case setup blindfold

Crash introduction to paraFoam – Mesh visualization

Select `Surface With Edges` in the Representation Toolbar

Fit to screen

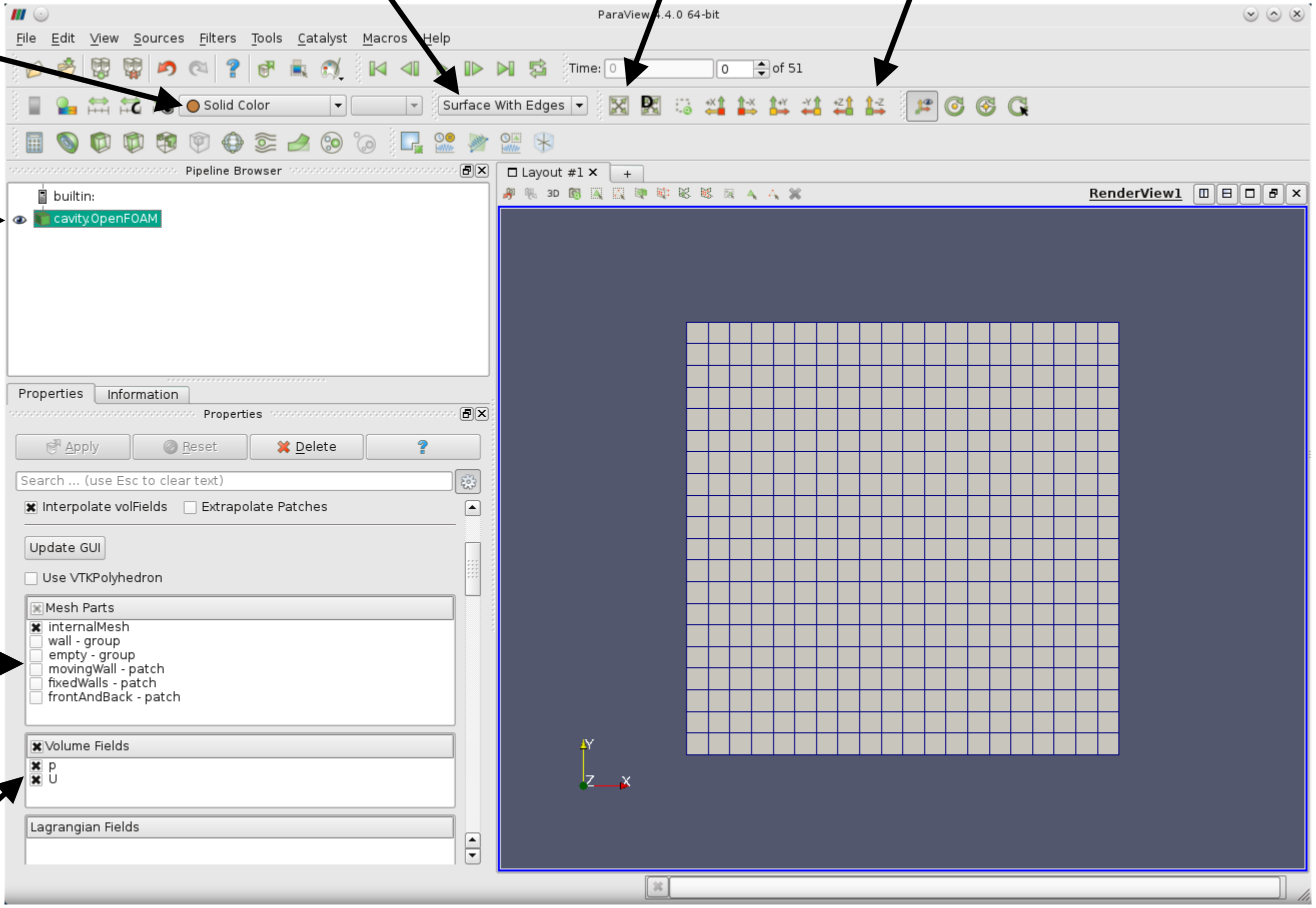
Select the -Z view

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**



Properties

- [x] Interpolate volFields
- [ ] Extrapolate Patches
- [ ] Use VTKPolyhedron

Mesh Parts
- [x] internalMesh
- [ ] wall - group
- [ ] empty - group
- [ ] movingWall - patch
- [ ] fixedWalls - patch
- [ ] frontAndBack - patch

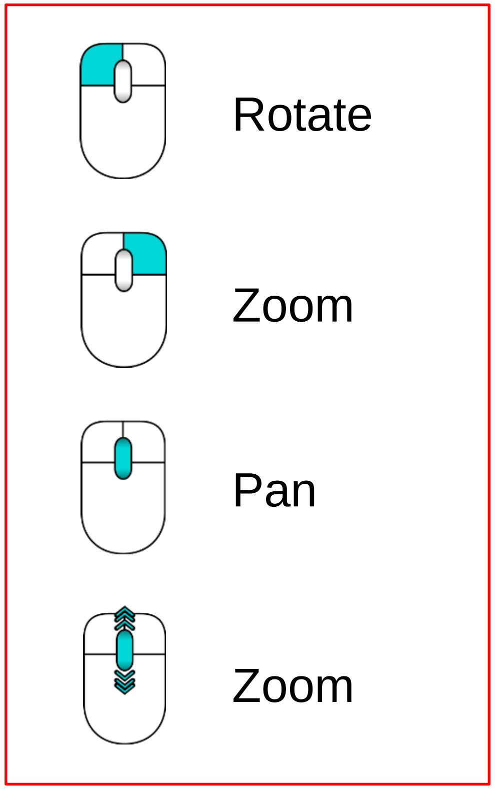
- [x] Volume Fields
  - [x] p
  - [x] U

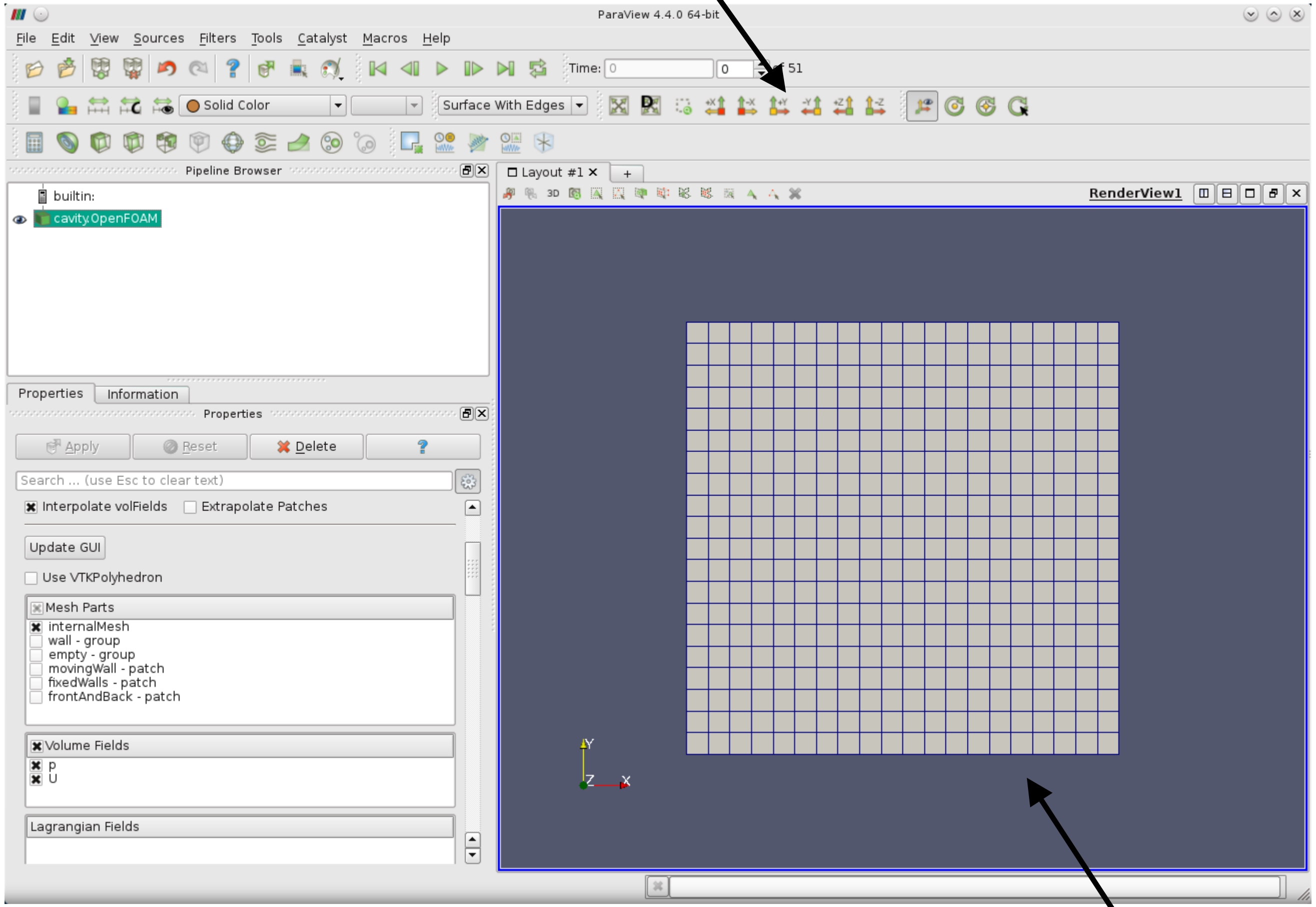
Running my first OpenFOAM® case setup blindfold

Crash introduction to paraFoam – 3D View and mouse interaction

Select view orientation in the Camera Controls

Mouse interaction in the 3D view





☒ Interpolate volFields ☐ Extrapolate Patches

☐ Use VTKPolyhedron

Mesh Parts
- ☒ internalMesh
- ☐ wall - group
- ☐ empty - group
- ☐ movingWall - patch
- ☐ fixedWalls - patch
- ☐ frontAndBack - patch

☒ Volume Fields
- ☒ p
- ☒ U

3D View/Canvas

Running my first OpenFOAM® case setup blindfold

Crash introduction to paraFoam – Fields visualization

Select `Last Frame` in the VCR Controls

Current Time Controls

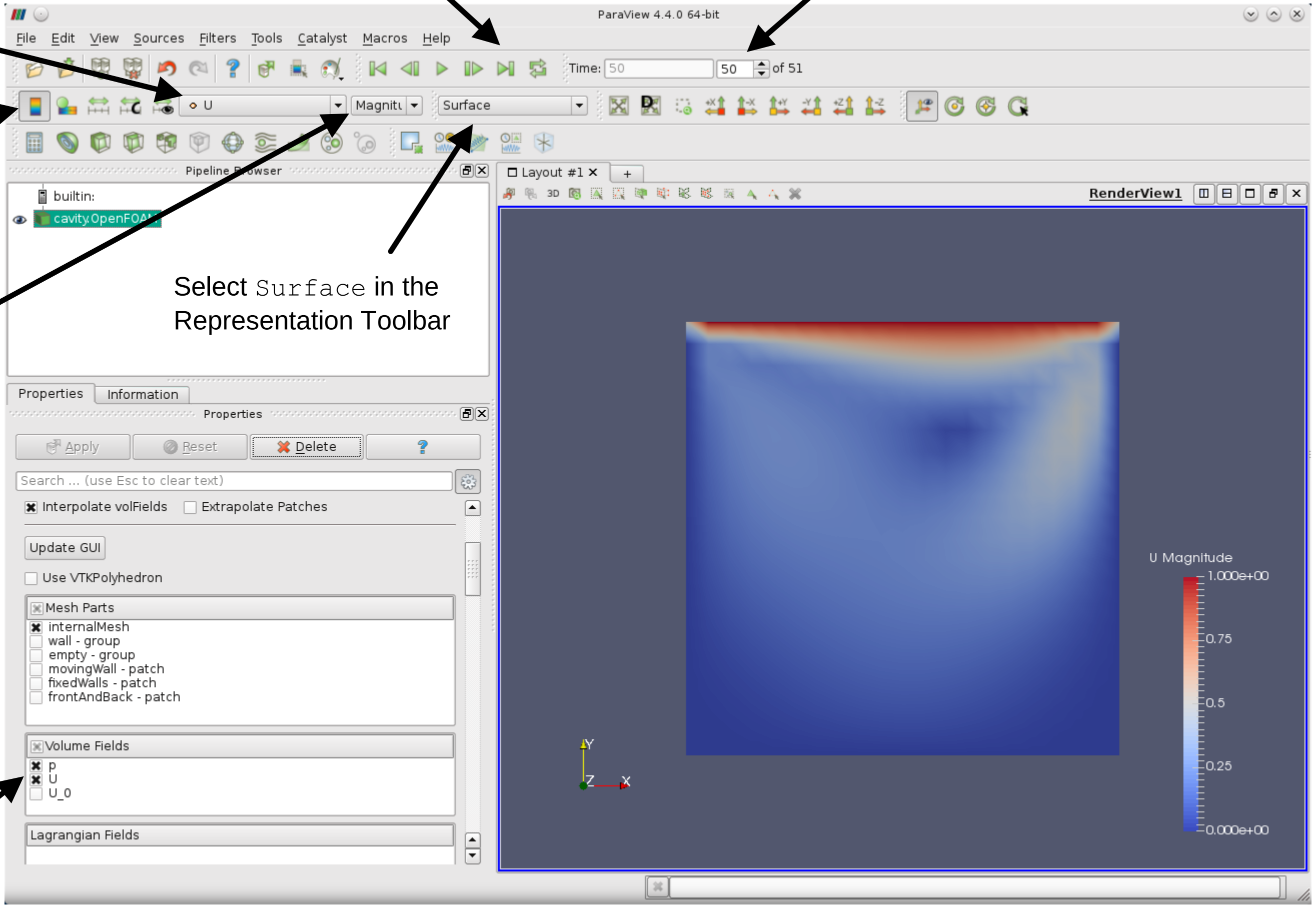
Select **U** in Active Variable Controls

Turn on/off color bar

Select `Surface` in the Representation Toolbar

Select `Magnitude` in the drop down menu

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



Interpolate volFields: ✓ | Extrapolate Patches: ☐

Update GUI

Use VTKPolyhedron: ☐

Mesh Parts
- ✓ internalMesh
- ☐ wall - group
- ☐ empty - group
- ☐ movingWall - patch
- ☐ fixedWalls - patch
- ☐ frontAndBack - patch

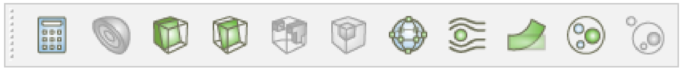
Volume Fields
- ✓ p
- ✓ U
- ☐ U_0

Lagrangian Fields

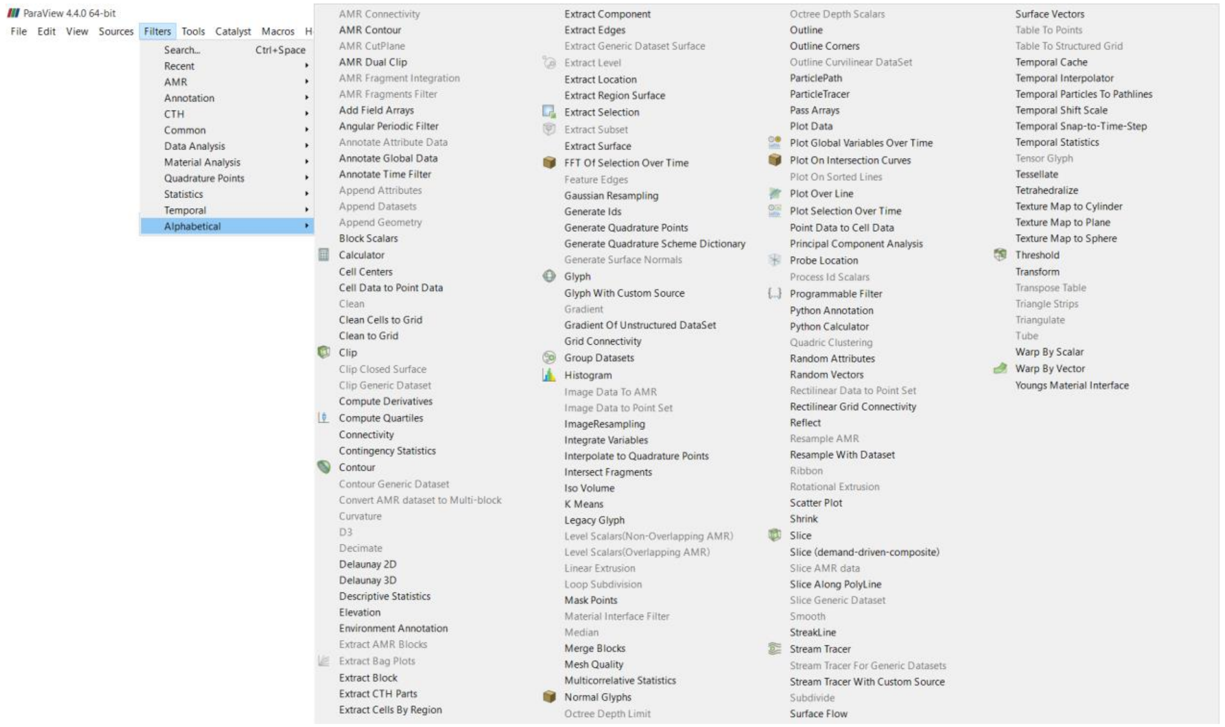
Running my first OpenFOAM® case setup blindfold

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



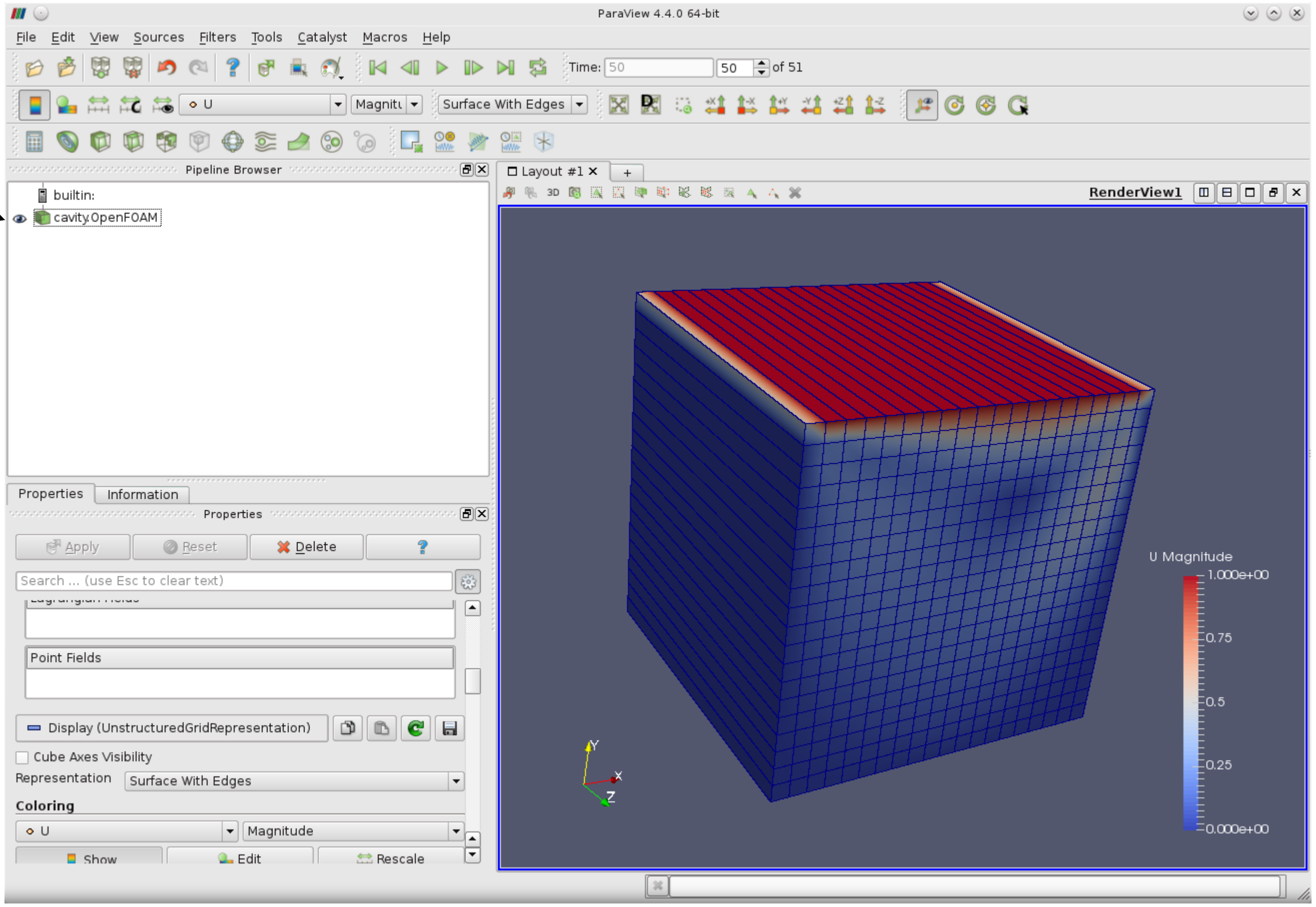
- You can access all the filters from the menu `Filter`.



Running my first OpenFOAM® case setup blindfold

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 setup blindfold

Crash introduction to paraFoam – Slice filter

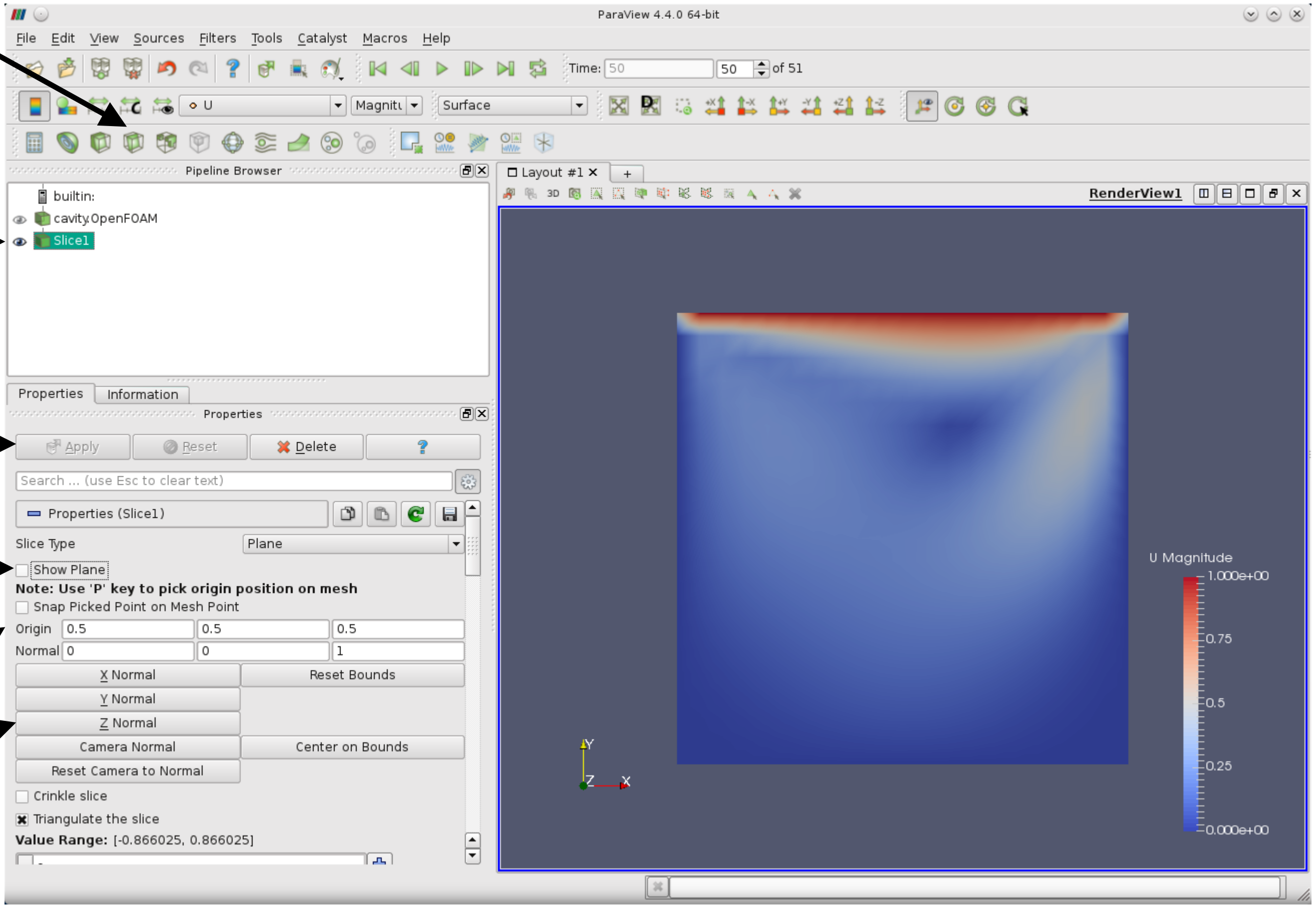
1. Select the `Slice` filter

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)



Time: 50 | 50 | of 51

U | Magnitude | Surface

Search ... (use Esc to clear text)

Slice Type: Plane

☐ Show Plane

Note: Use 'P' key to pick origin position on mesh

☐ Snap Picked Point on Mesh Point

Origin 0.5 0.5 0.5

Normal 0 0 1

☐ Crinkle slice

☒ Triangulate the slice

Value Range: [-0.866025, 0.866025]

Running my first OpenFOAM® case setup blindfold

Crash introduction to paraFoam – Glyph filter

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

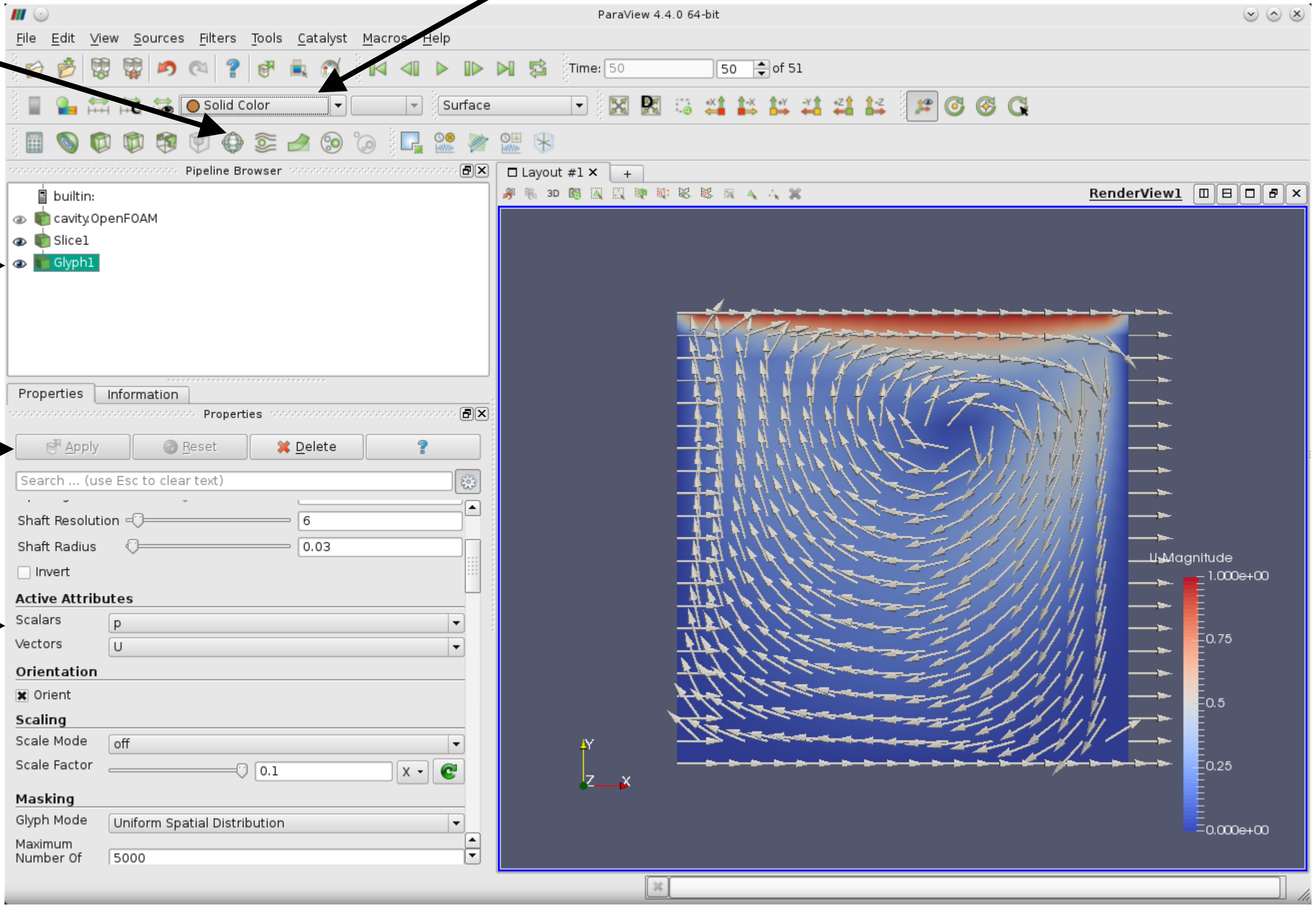
Notice that the filter `Glyph` was applied on the `Slice1` filter.

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.

4. Color the colors using Solid Color



Pipeline Browser:
- builtin:
- cavity.OpenFOAM
- Slice1
- Glyph1

Properties:

Shaft Resolution 6
Shaft Radius 0.03
Invert

Active Attributes
Scalars p
Vectors U

Orientation
☒ Orient

Scaling
Scale Mode off
Scale Factor 0.1

Masking
Glyph Mode Uniform Spatial Distribution
Maximum Number Of 5000

Running my first OpenFOAM® case setup blindfold

Crash introduction to paraFoam – Plot Over Line filter

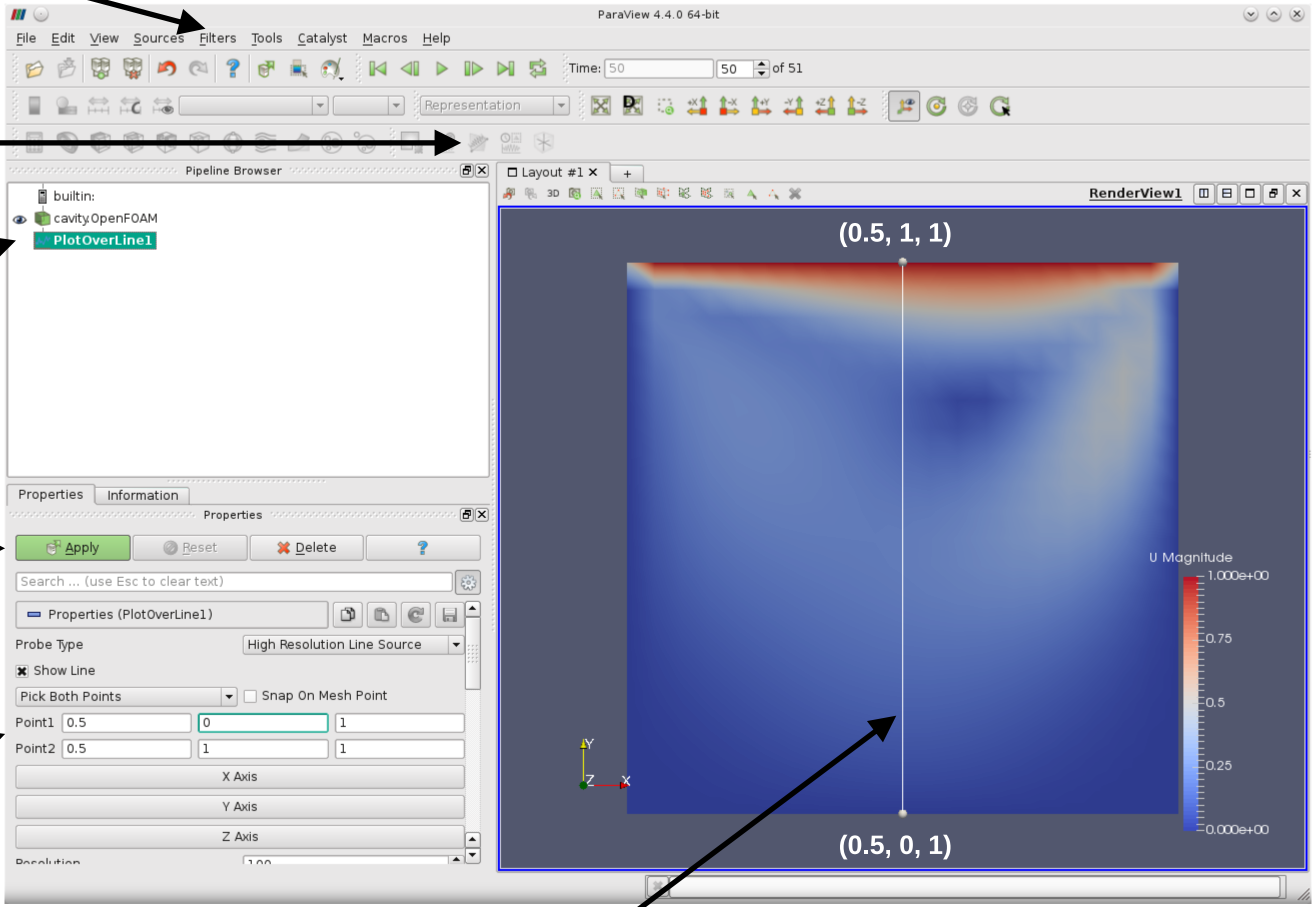
1.a. Select the `Plot Over Line` filter.

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



Probe Type: High Resolution Line Source
Show Line [✓]
Pick Both Points
Snap On Mesh Point [ ]

Point1	0.5	0	1
Point2	0.5	1	1

Line

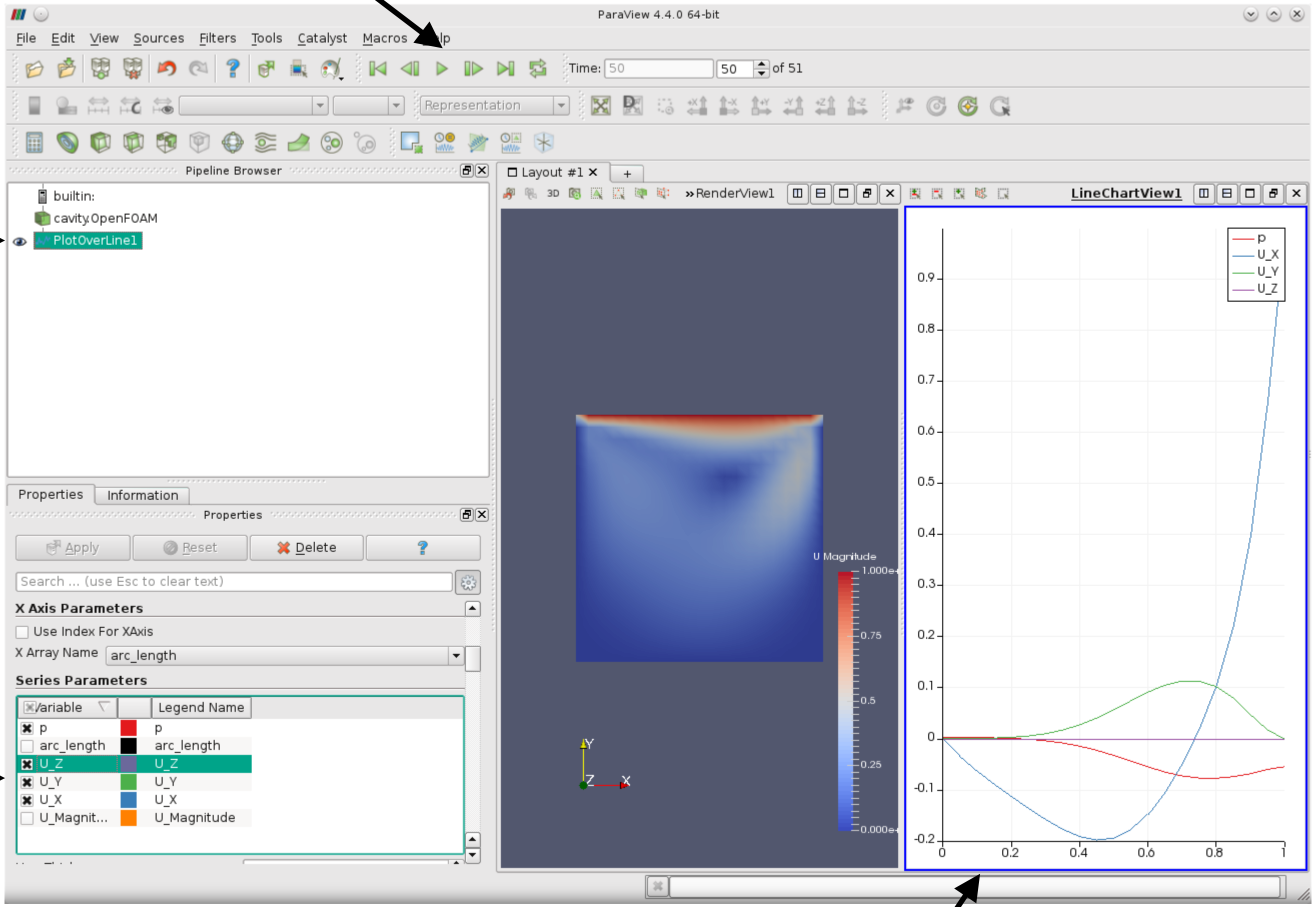
Running my first OpenFOAM® case setup blindfold

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



X Axis Parameters

Use Index For XAxis

X Array Name: arc_length

Series Parameters

Variable		Legend Name
☒ p	🟥	p
☐ arc_length	⬛	arc_length
☒ U_Z	🟪	U_Z
☒ U_Y	🟩	U_Y
☒ U_X	🟦	U_X
☐ U_Magnit...	🟧	U_Magnitude

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

Running my first OpenFOAM® case setup blindfold

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 just 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, it is likely that we will ask you for this file.
- We might also ask for the standard error stream (stderr).

Running my first OpenFOAM® case setup blindfold

Running the case blindfold with log files

- There are many ways to save the *log* files.
- From now on, we will use the Linux `tee` command to save *log* files.
- To save a *log* file of the simulation or the output of any utility, you can proceed as follows:

```
1.  $> foamCleanTutorials
2.  $> blockMesh | tee log.blockMesh
3.  $> checkMesh | tee log.checkMesh
4.  $> icoFoam | tee log.icoFoam
```

The vertical bar or pipelining operator is used to concatenate commands

- You can use your favorite text editor to read the log file (e.g., gedit, vi, emacs).

Running my first OpenFOAM® case setup blindfold

Running the case blindfold with log files

- In step 1 we erase the mesh and all the folders, except for **`0`**, **`constant`** and **`system`**. This script comes with your OpenFOAM® installation.
- In step 2, we generate the mesh using the meshing tool `blockMesh`. We also redirect the standard output to an ascii file with the name *`log.blockMesh`* (it can be any name). The `tee` command will redirect the screen output to the file *`log.blockMesh`* and at the same time will show you the information on the screen.
- In step 3 we check the mesh quality. We also redirect the standard output to an ascii file with the name *`log.checkMesh`* (it can be any name).
- In step 4 we run the simulation. We also redirect the standard output to an ascii file with the name *`log.icoFoam`* (it can be any name). Remember, the `tee` command will redirect the screen output to the file *`log.icoFoam`* and at the same time will show you the information on the screen.

Running my first OpenFOAM® case setup blindfold

Running the case blindfold with log files

- To postprocess the information contained in the solver log file *`log.icoFoam`*, we can use the utility `foamLog`. Type in the terminal:
  - `$> foamLog log.icoFoam`

- This utility will extract the information inside the file *`log.icoFoam`*. The extracted information is saved in an editable/plottable format in the directory **`logs`**.
- At this point we can use `gnuplot` to plot the residuals.
- Type in the terminal:
  - `$> gnuplot`

Running my first OpenFOAM® case setup blindfold

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
   ```

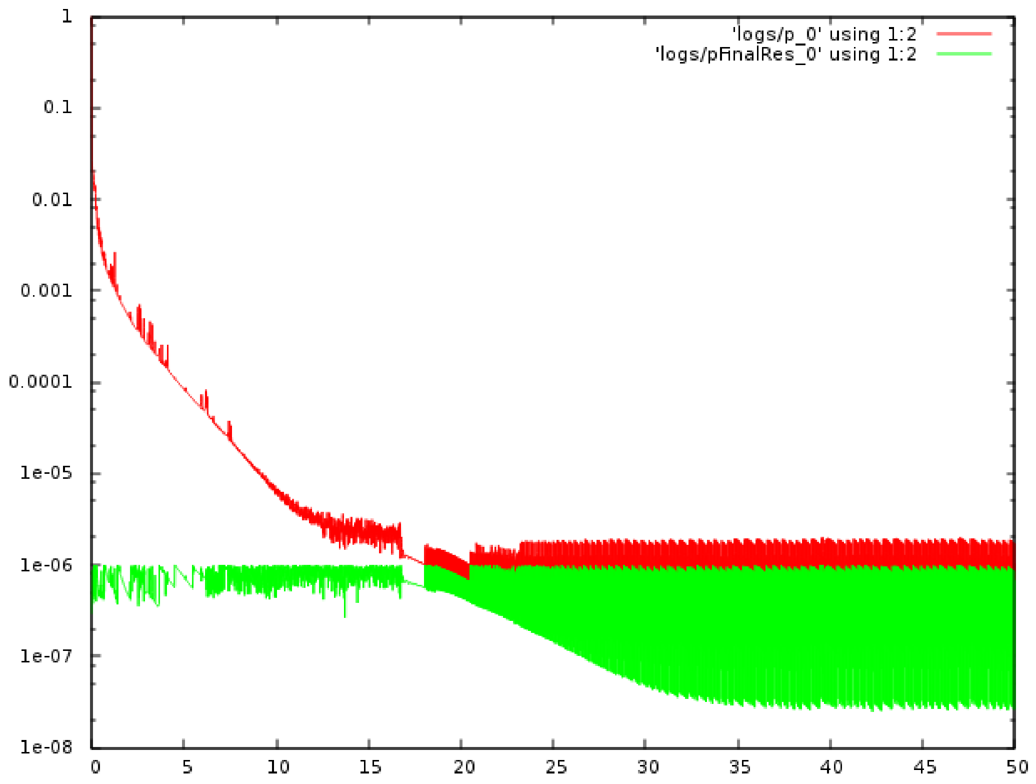
7.
   ```
   gnuplot> plot [30:50][] ‘logs/Ux_0’ u 1:2 w l title ‘Ux’,‘logs/Uy_0’ u 1:2 w l title ‘Uy’
   ```
   Set the x range from 30 to 50 and plot tow files and set legend titles

8.
   ```
   gnuplot> exit
   ```
   To exit gnuplot

Running my first OpenFOAM® case setup blindfold

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 setup blindfold

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):
    - `$> pyFoamPlotRunner.py [options] <foamApplication>`
- If you are using the lab workstations, you will need to source PyFoam. To source PyFoam, type in the terminal:
    - `$> anaconda3`
- If you need help or want to know all the options available,
    - `$> pyFoamPlotRunner.py --help`

Running my first OpenFOAM® case setup blindfold

Running the case blindfold with log files and plotting the residuals

- To run this case with `pyFoamPlotRunner.py`, in the terminal type:
  - `$> pyFoamPlotRunner.py icoFoam`

- 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,
  - `$> icoFoam | tee log.icoFoam`

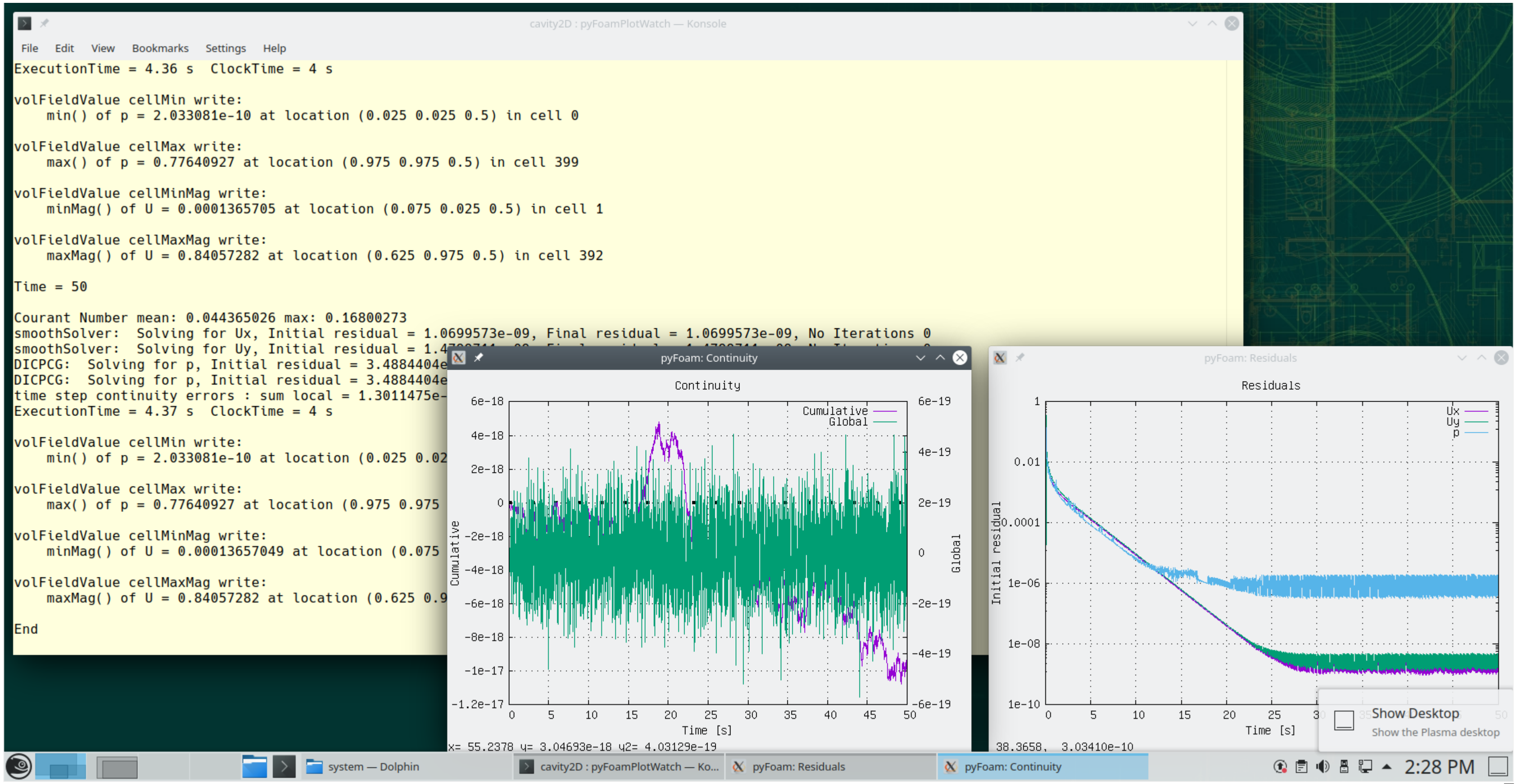
- Then, in a new terminal window launch `pyFoamPlotWatcher`, as follows,
  - `$> pyFoamPlotWatcher.py log.icoFoam`

- You can also use `pyFoamPlotWatcher.py` to plot the information saved in an old *`log`* file.

Running my first OpenFOAM® case setup blindfold

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.



```
ExecutionTime = 4.36 s  ClockTime = 4 s

volFieldValue cellMin write:
    min() of p = 2.033081e-10 at location (0.025 0.025 0.5) in cell 0

volFieldValue cellMax write:
    max() of p = 0.77640927 at location (0.975 0.975 0.5) in cell 399

volFieldValue cellMinMag write:
    minMag() of U = 0.0001365705 at location (0.075 0.025 0.5) in cell 1

volFieldValue cellMaxMag write:
    maxMag() of U = 0.84057282 at location (0.625 0.975 0.5) in cell 392

Time = 50

Courant Number mean: 0.044365026 max: 0.16800273
smoothSolver:  Solving for Ux, Initial residual = 1.0699573e-09, Final residual = 1.0699573e-09, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 1.4
DICPCG:  Solving for p, Initial residual = 3.4884404e
DICPCG:  Solving for p, Initial residual = 3.4884404e
time step continuity errors : sum local = 1.3011475e-
ExecutionTime = 4.37 s  ClockTime = 4 s

volFieldValue cellMin write:
    min() of p = 2.033081e-10 at location (0.025 0.02

volFieldValue cellMax write:
    max() of p = 0.77640927 at location (0.975 0.975

volFieldValue cellMinMag write:
    minMag() of U = 0.00013657049 at location (0.075

volFieldValue cellMaxMag write:
    maxMag() of U = 0.84057282 at location (0.625 0.9

End
```

Running my first OpenFOAM® case setup blindfold

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.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         50;
```

Running my first OpenFOAM® case setup blindfold

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.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          writeNow;

endTime         50;
```

Running my first OpenFOAM® case setup blindfold

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.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    object      controlDict;
}


runTimeModifiable true;

```

Running my first OpenFOAM® case setup blindfold

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:  9                                     |
|   \\  /    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/5x/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 $\longrightarrow$ PID 3100

Running my first OpenFOAM® case setup blindfold

Stopping the simulation

- When working locally, we usually proceed in this way:

  - `$> icoFoam | tee 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 setup blindfold

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
```

Roadmap

1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. ~~Directory structure of an application/utility~~
4. ~~Applications/utilities in OpenFOAM®~~
5. ~~Directory structure of an OpenFOAM® case~~
6. ~~Running my first OpenFOAM® case setup blindfold~~
7. **A deeper view to my first OpenFOAM® case setup**
8. 3D Dam break – Free surface flow
9. Flow past a cylinder – From laminar to turbulent flow

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 13, if not, OpenFOAM® will complain.
  - According to the dictionary you are using, the **class** keyword (line 11) 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:  9                                     |
5     |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6     |    \\/     M anipulation  |                                                 |
7     \*---------------------------------------------------------------------------*/
8     FoamFile
9     {
10        format      ascii;
11        class       dictionary;
12        object      controlDict;
13    }
```

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.

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

- Notice that line 19 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,

```
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.
- Also, depending on 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 briefly address a few important inputs of the *`blockMeshDict`* dictionary.
- Do not worry, we are going to revisit this dictionary during the meshing session.
- However, have in mind that rarely you will use this utility to generate a mesh for complex geometries.
- Go to the directory **`system`** and open *`blockMeshDict`* dictionary with your favorite text editor, we will use gedit.

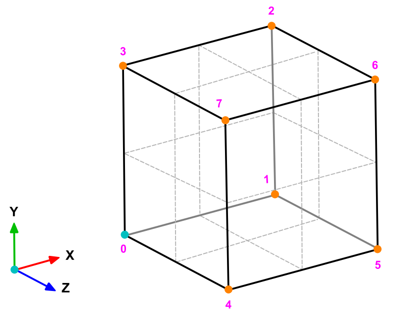
A deeper view to my first OpenFOAM® case setup

The *system/blockMeshDict* dictionary

- The *blockMeshDict* dictionary first defines 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. These variables are related to the number of cells in each direction.
- Finally, notice that the vertex numbering starts from 0 (as the counters in c++). This numbering applies for blocks as well.

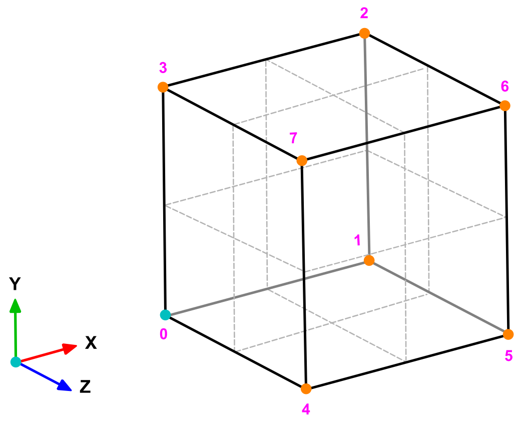


A deeper view to my first OpenFOAM® case setup

The *blockMeshDict* dictionary.

- In lines 60-63, we define the block topology.
- In line 62, the **hex** keyword means that it is a structured hexahedral block.
- In this case, we are generating a rectangular mesh using a single block.
- In the same line, **(0 1 2 3 4 5 6 7)** are the vertices used to define the block (and yes, the order is important).
- Each hex block is defined by eight vertices in sequential order and where the first vertex in the list represents the origin of the coordinate system (vertex **0** in this case).
- **($xcells $ycells $zcells)** is the number of mesh cells in each direction (**X Y Z**). Notice that we are using macro syntax, and we compute the values using inline calculations.
- **simpleGrading (1 1 1)** is the grading or mesh stretching in each direction (**X Y Z**), in this case the mesh is uniform. We will deal with mesh grading/stretching in the next case.

```
60      blocks
61      (
62          hex (0 1 2 3 4 5 6 7) ($xcells $ycells $zcells) simpleGrading (1 1 1)
63      );
64
65      edges
66      (

69      );
```

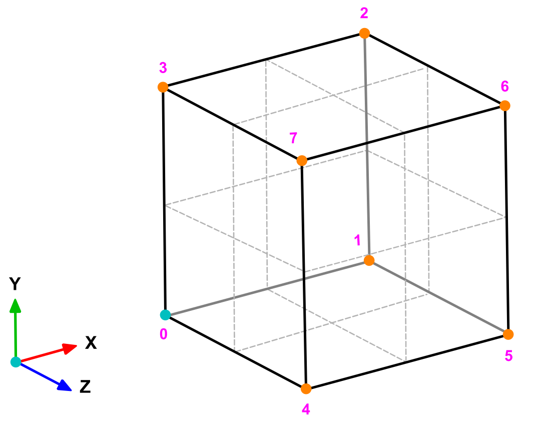


A deeper view to my first OpenFOAM® case setup

The *blockMeshDict* dictionary.

- Let us talk about the block ordering **hex (0 1 2 3 4 5 6 7)**, which is extremely important (line 62).
- **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.
- 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** (notice that we start from vertex **4** which is the projection in the **Z**-direction of vertex **0**).
- Notice that the vertices are ordered in such a way that if we look at the screen/paper (-z direction), the vertices rotate counter-clockwise.
- If you add a second block, you must identify the first vertex and starting from it, you must construct the block topology (in this case you will need to merges faces, you will find more information about merging faces in the supplement lectures).

```
60      blocks
61      (
62          hex (0 1 2 3 4 5 6 7) ($xcells $ycells $zcells) simpleGrading (1 1 1)
63      );
64
65      edges
66      (

69      );
```

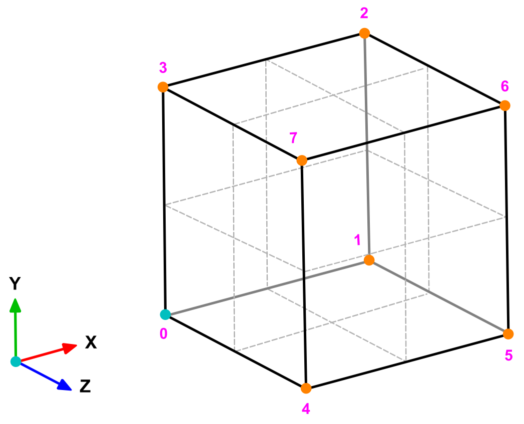


A deeper view to my first OpenFOAM® case setup

The *blockMeshDict* dictionary.

- In lines 65-69, we define edges.
- Edges, are constructed from the vertices definition.
- Each edge joining two vertices is assumed to be straight by default.
- The user can specify any edge to be curved by entries in the section **edges**.
- Possible options are Bspline, arc, line, polyline, project, projectCurve, spline.
- 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.
- To define a polyline, we first define the vertices to be connected to form an edge and then we give a list of the coordinates of the interpolation points.
- In this case and as we do not specify anything, all edges are assumed to be straight lines.

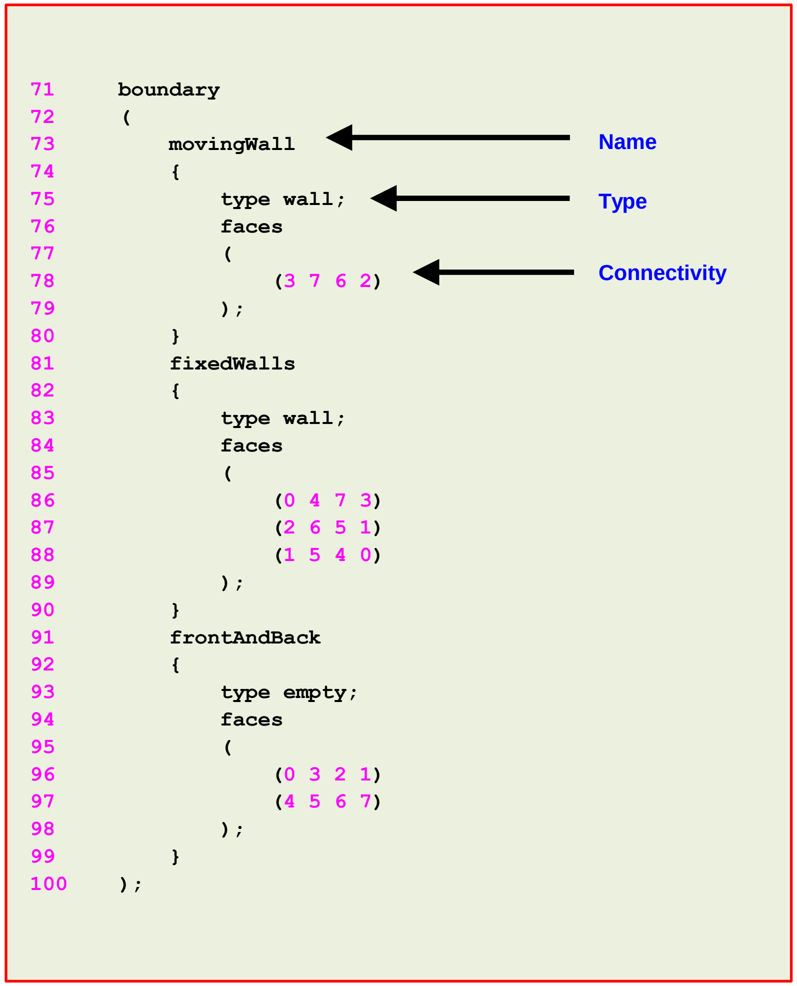
```
60      blocks
61      (
62          hex (0 1 2 3 4 5 6 7) ($xcells $ycells $zcells) simpleGrading (1 1 1)
63      );
64
65      edges
66      (

69      );
```



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     );
```

- 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 counterclockwise.
- 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

- To sum up, the *blockMeshDict* dictionary generates in this case 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 easy geometries (rectangles, cylinders, and so on), then you can use `blockMesh` to do the meshing, 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.
- 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.

A deeper view to my first OpenFOAM® case setup

The `constant/polyMesh/boundary` dictionary

- First of 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 (or surface patch) of the domain is specified.
- The **base type** boundary condition is the actual surface patch where we are going to apply a **numerical type** boundary condition (or numerical boundary condition).
- The **numerical type** boundary condition assign a field value to the surface patch (**base type**).
- We define the **numerical type** patch (or the value of the boundary condition), in the directory **0** or time directories.

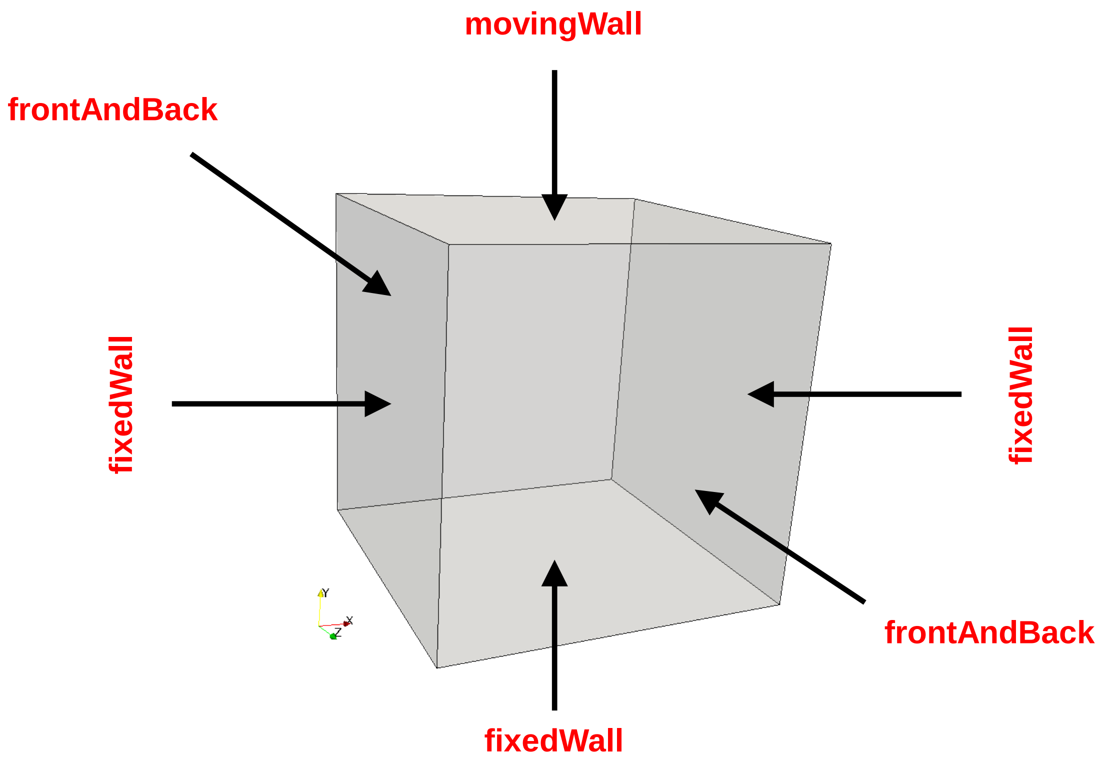
A deeper view to my first OpenFOAM® case setup

The *constant/polyMesh/boundary* dictionary

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

```
3
(
    movingWall
    {
        type            wall;
        inGroups        1(wall);
        nFaces          20;
        startFace       760;
    }
    fixedWalls
    {
        type            wall;
        inGroups        1(wall);
        nFaces          60;
        startFace       780;
    }
    frontAndBack
    {
        type            empty;
        inGroups        1(empty);
        nFaces          800;
        startFace       840;
    }
)
```

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

```
3
(
    movingWall
    {
        type            wall;
        inGroups        1(wall);
        nFaces          20;
        startFace       760;
    }
    fixedWalls
    {
        type            wall;
        inGroups        1(wall);
        nFaces          60;
        startFace       780;
    }
    frontAndBack
    {
        type            empty;
        inGroups        1(empty);
        nFaces          800;
        startFace       840;
    }
)
```

Name

Type

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 **movingWall** 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 file. 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

```
3
(
    movingWall
    {
        type            wall;
        inGroups        1(wall);
        nFaces          20;
        startFace       760;
    }
    fixedWalls
    {
        type            wall;
        inGroups        1(wall);
        nFaces          60;
        startFace       780;
    }
    frontAndBack
    {
        type            empty;
        inGroups        1(empty);
        nFaces          800;
        startFace       840;
    }
)
```

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.

nFaces and startFace keywords

- Unless you know what are you doing, you do not need to modify this information.
- This information is related to the starting face and ending face of the boundary patch in the mesh data structure.
- This information 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

- 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 **numerical 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 **numerical 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 **numerical 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 **numerical type** boundary conditions are not consistent, OpenFOAM® will complain.
- Do not worry, we are going to address boundary conditions later on.

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 **numerical** 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	**calculated** **fixedValue** **flowRateInletVelocity** **freestream** **inletOutlet** **slip** **totalPressure** **zeroGradient** … and so on Refer to the doxygen documentation or the source code for a list of all numerical 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 (0 0 0);
		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 **numerical 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 detail 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**
- 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.
- Remember, 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

```
ddtSchemes
{
    default         backward;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;

   div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    //default         Gauss linear orthogonal;
    default         Gauss linear limited 1;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    //default         orthogonal;
    default         limited 1;
}
```

- 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 **limited 1** corrections (to handle mesh non-orthogonality and non-uniformity).
- The method we are using is second order accurate but oscillatory. We are going to talk about the properties of the numerical schemes later.
- 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

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;

    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }
}

PISO
{
    nCorrectors     1;
    nNonOrthogonalCorrectors 1;
    pRefCell        0;
    pRefValue       0;
}
```

- 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.
- In this case, we are using the same tolerances for **p** and **pFinal.** However, you can use difference tolerances, where usually you use a tighter tolerance in **pFinal**.

A deeper view to my first OpenFOAM® case setup

The *fvSolution* dictionary

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;

    }

    pFinal
    {
        $p;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }
}

PISO
{
    nCorrectors     1;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

- 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 relatively 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

```
type sets;

setFormat raw;

interpolationScheme cellPointFace;

fields
(
    U
);

sets
(

    l1
    {
        type                lineFace;
        axis                x;
        start               ( -1   0.5  0);
        end                 ( 2   0.5  0);
    }

    l2
    {
        type                lineFace;
        axis                y;
        start               (0.5  -1  0);
        end                 (0.5  2  0);
    }

);
```

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

```
type sets;

setFormat raw;

interpolationScheme cellPointFace;

fields
(
    U
);

sets
(

    l1
    {
        type                lineFace;
        axis                x;
        start               ( -1  0.5 0);
        end                 ( 2  0.5 0);
    }

    l2
    {
        type                lineFace;
        axis                y;
        start               (0.5 -1 0);
        end                 (0.5 2 0);
    }

);
```

Name of the output file

Name of the output file

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.

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):

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

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}
```

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

Uniform initial conditions.

The velocity field is initialized 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 **numerical 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):

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

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}
```

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):

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}

```

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

Uniform initial conditions.

The modified pressure field is initialized 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 **numerical 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):

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}

```

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,

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

- 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*, *gradU*, and so on).
- In the header of the field variables, the class type should be consistent with the type of field variable you are using.
- Be careful with this, specially if you are copying and pasting files.
- If the field variable is a scalar, the class should be **volScalarField**.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

A deeper view to my first OpenFOAM® case setup

- If the field variable is a vector, the class should be **volVectorField.**

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       volVectorField;
    object      U;
}
```

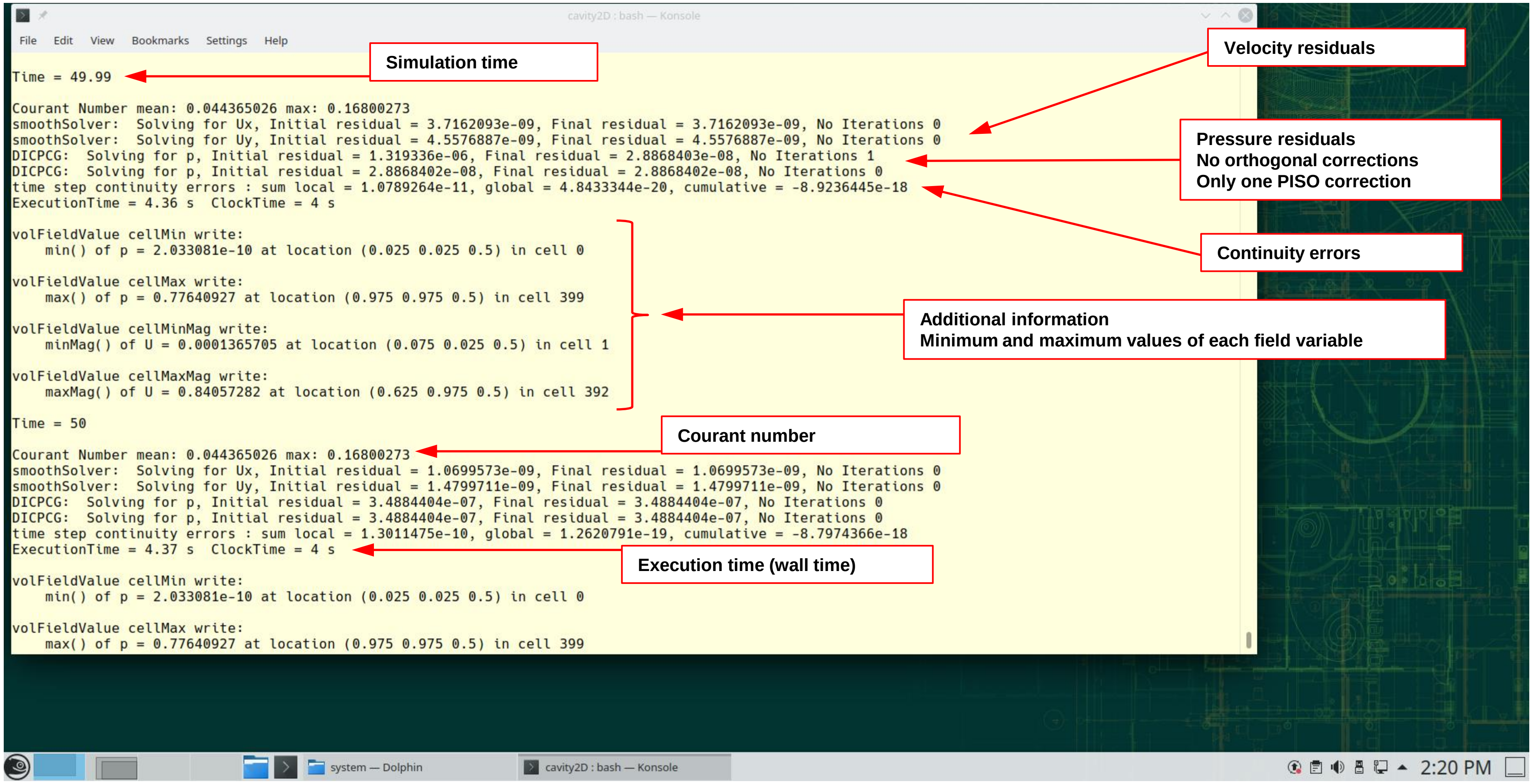
- If the field variable is a tensor (e.g., the velocity gradient tensor), the class should be **volTensorField**.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       volTensorField;
    object      gradU;
}
```

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.



cavity2D : bash — Konsole
File Edit View Bookmarks Settings Help
Simulation time
Velocity residuals
Pressure residuals
No orthogonal corrections
Only one PISO correction
Continuity errors
Additional information
Minimum and maximum values of each field variable
Courant number
Execution time (wall time)
system — Dolphin
cavity2D : bash — Konsole
2:20 PM

```
Time = 49.99

Courant Number mean: 0.044365026 max: 0.16800273
smoothSolver:  Solving for Ux, Initial residual = 3.7162093e-09, Final residual = 3.7162093e-09, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 4.5576887e-09, Final residual = 4.5576887e-09, No Iterations 0
DICPCG:  Solving for p, Initial residual = 1.319336e-06, Final residual = 2.8868403e-08, No Iterations 1
DICPCG:  Solving for p, Initial residual = 2.8868402e-08, Final residual = 2.8868402e-08, No Iterations 0
time step continuity errors : sum local = 1.0789264e-11, global = 4.8433344e-20, cumulative = -8.9236445e-18
ExecutionTime = 4.36 s  ClockTime = 4 s

volFieldValue cellMin write:
    min() of p = 2.033081e-10 at location (0.025 0.025 0.5) in cell 0

volFieldValue cellMax write:
    max() of p = 0.77640927 at location (0.975 0.975 0.5) in cell 399

volFieldValue cellMinMag write:
    minMag() of U = 0.0001365705 at location (0.075 0.025 0.5) in cell 1

volFieldValue cellMaxMag write:
    maxMag() of U = 0.84057282 at location (0.625 0.975 0.5) in cell 392

Time = 50

Courant Number mean: 0.044365026 max: 0.16800273
smoothSolver:  Solving for Ux, Initial residual = 1.0699573e-09, Final residual = 1.0699573e-09, No Iterations 0
smoothSolver:  Solving for Uy, Initial residual = 1.4799711e-09, Final residual = 1.4799711e-09, No Iterations 0
DICPCG:  Solving for p, Initial residual = 3.4884404e-07, Final residual = 3.4884404e-07, No Iterations 0
DICPCG:  Solving for p, Initial residual = 3.4884404e-07, Final residual = 3.4884404e-07, No Iterations 0
time step continuity errors : sum local = 1.3011475e-10, global = 1.2620791e-19, cumulative = -8.7974366e-18
ExecutionTime = 4.37 s  ClockTime = 4 s

volFieldValue cellMin write:
    min() of p = 2.033081e-10 at location (0.025 0.025 0.5) in cell 0

volFieldValue cellMax write:
    max() of p = 0.77640927 at location (0.975 0.975 0.5) in cell 399
```

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 online monitors.
- In OpenFOAM®, the monitors are referred to as **functionObject**.
- We are going to address **functionObject** in detail when we deal with post-processing and sampling.
- But for the moment, what we need to know is that we add **functionObject** 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.
- Minimum and maximum values of the field variables give a better indication of stability, boundedness and consistency of the solution.

A deeper view to my first OpenFOAM® case setup

The output screen

- There are many ways to define **functionObjects** in OpenFOAM.
- In this case we are using packed **functionObjects**.
- The packed **functionObjects** are located in the directory **`$WM_PROJECT_DIR/etc/caseDicts`**
- All **functionObjects** are defined in the block (or sub-dictionary) `functions` (lines 49-222).

```
49      functions
50      {
51
169     cellMin
170     {
171         #includeEtc "caseDicts/postProcessing/minMax/cellMin.cfg"
172         enabled         true;      //true or false
173         log             true;      //write to screen
174         fields   (p);
175     }
177     cellMax
178     {
179         #includeEtc "caseDicts/postProcessing/minMax/cellMax.cfg"
180         enabled         true;
181         log             true;
182         fields   (p);
183     }
186     cellMinMag
187     {
188         #includeEtc "caseDicts/postProcessing/minMax/cellMinMag.cfg"
189         enabled         true;
190         log             true;
191         fields   (U);
192     }
194     cellMaxMag
195     {
196         #includeEtc "caseDicts/postProcessing/minMax/cellMaxMag.cfg"
197         enabled         true;
198         log             true;
199         fields   (U);
200     }
222     };
```

Name of the functionObject.
This is also the name of the folder where the output of the functionObject will be saved.

Location and type of the packed functionObject.
- cellMin – minimum value of a scalar field.
- cellMax – maximum value of a scalar field.
- cellMinMag – minimum value of a vector field (magnitude).
- cellMaxMag – maximum value of a vector field (magnitude).

Options related to the functionObject.
The functionObject can have more options than the ones shown here.
In this case we are using the following options:
- enabled = Enable/disable functionObject.
- log = Print to screen output of the functionObject

Fields to sample

A deeper view to my first OpenFOAM® case setup

The output screen

- A packed **functionObject** can be expanded as follows,

```
min_scalar
{
    type            volFieldValue;

    libs            ("libfieldFunctionObjects.so");

    enabled         true;
    log             true;

    writeControl    timeStep;
    writeInterval   1;

    writeLocation   true;

    regionType      all;

    operation       min;

    fields
    (
        p
    );
}
```

functionObject to use.

Library to use.

Enable/disable functionObject.
Print to screen output of the functionObject.

Output interval of the functionObject

Write location of the minimum value (or maximum value)

Compute the functionObject in all the domain

Compute the minimum value.
Possible options (among many): min, max, minMag, maxMag.

Apply the functionObject to these field variables.

- This expanded **functionObject** is equivalent to the packed **functionObject** listed in lines 169-175.
- In lines 77-162 the expanded **functionObject** are listed (notice that these lines are commented).
- In the expanded **functionObject**, besides the compulsory entries, we can also define optional entries.

A deeper view to my first OpenFOAM® case setup

The output screen

- Another very important output information is the CFL or Courant number.
- 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,

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

- 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 5.0 and preferably close to 1.0 (for good accuracy).
- 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 you should get for this case,

```
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 you should get for this case,

```
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 you should get for this case,

```
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 involved 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-2), 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)
```

Continuity errors

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:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 5.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  : 6.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/6/101OF/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

- Mismatching patch name in file *p*

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

file: /home/joegi/my_cases_course/6/101OF/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-6/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/6/101OF/cavity/system/fvSchemes.divSchemes"

file: /home/joegi/my_cases_course/6/101OF/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/6/101OF/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-6/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/6/101OF/cavity/system/controlDict"

file: /home/joegi/my_cases_course/6/101OF/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/6/101OF/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-6/src/finiteVolume/lnInclude/fvPatchFieldNew.C at line 143.

FOAM exiting
```

A deeper view to my first OpenFOAM® case setup

Error output

- This one is especially hard to spot,

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 6.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 blank space between the word cavity and the number 0).
- Do not use blank spaces or funny symbols when naming directories and files.
- Instead of **`cavity 0`** you could 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
*/
```

- As in C++, you can use the **#include** directive in your dictionaries (do not forget to create the respective include file):

```
#include "initialConditions"
```

- 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 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

- 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 followed by the keyword entries in curly braces:

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

    U
    {
        solver           PBiCGStab;
        preconditioner   DILU;
        tolerance        1e-06;
        relTol           0;
    }
     ...
     ...
     ...
}
```

$\leftarrow$ **Dictionary solvers**

$\leftarrow$ **Sub-dictionary p**

$\leftarrow$ **Sub-dictionary U**

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
```

- 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

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

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

- 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 regeaxp).

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 very useful to parametrize dictionaries and avoid repetitive tasks.
- 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:

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

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

OpenFOAM-9/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

- You can also use the utility `foamInfo`:
  - `$> foamInfo` $\underbrace{\text{search\_string}}_{\text{Look for this string}}$

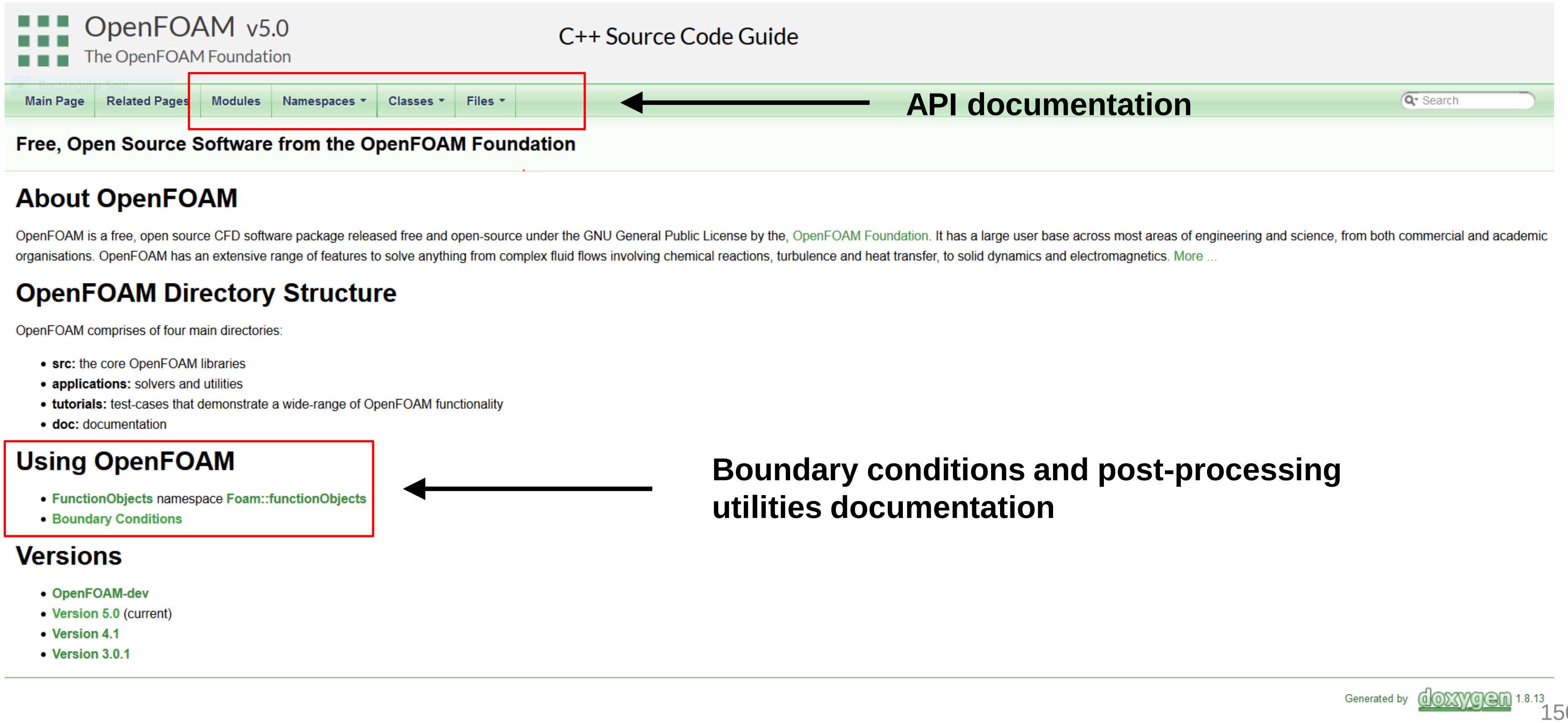
- This command prints the following for an application, a script, or a model (including boundary conditions, function objects and fvModels).
  - File: the location of the relevant source code header file.
  - Description details from the header file.
  - Usage details from the header file.
  - Models: lists other models belonging to the same family, where applicable.

- To get more information type in the terminal
  - `$> foamInfo -help`

A deeper view to my first OpenFOAM® case setup

Solvers and utilities help

- To get more information about the boundary conditions, post-processing utilities, and the API read the Doxygen documentation.
- If you did not compile the Doxygen documentation, you can access the information online, http://cpp.openfoam.org/v6/



Free, Open Source Software from the OpenFOAM Foundation

About OpenFOAM

OpenFOAM is a free, open source CFD software package released free and open-source under the GNU General Public License by the, OpenFOAM Foundation. It has a large user base across most areas of engineering and science, from both commercial and academic organisations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics. More ...

OpenFOAM Directory Structure

OpenFOAM comprises of four main directories:

- **src:** the core OpenFOAM libraries
- **applications:** solvers and utilities
- **tutorials:** test-cases that demonstrate a wide-range of OpenFOAM functionality
- **doc:** documentation

Using OpenFOAM

- FunctionObjects namespace Foam::functionObjects
- Boundary Conditions

Versions

- OpenFOAM-dev
- Version 5.0 (current)
- Version 4.1
- Version 3.0.1

A deeper view to my first OpenFOAM® case setup

Exercises

- Run the case with Re = 10 and Re = 1000. Feel free to change any variable to achieve the Re value (velocity, viscosity or length). Do you see an unsteady behavior in any of the cases? What about the computing time, what simulation is faster?
- Run the tutorial with Re = 100, a mesh with 120 x 120 x 1 cells, and using the default setup (original *`controlDict`*, *`fvSchemes`* and *`fvSolution`*). Did the simulation converge? Did it crash? Any comments.
- If your simulation crashed, try to solve the problem.
  (Hint: try to reduce the time-step to get a CFL less than 1)
- Besides reducing the time-step, can you find another solution?
  (Hint: look at the PISO options)
- Change the **base type** of the boundary patch **movingWall** to **patch**. (the *`boundary`* file). Do you get the same results? Can you comment on this?
- Try to extent the problem to 3D and use a uniform mesh (20 x 20 x 20). Compare the solution at the mid section of the 3D simulation with the 2D solution. Are the solutions similar?
- How many time discretization schemes are there in OpenFOAM®? Try to use a different discretization scheme.
- Run the simulation using **Gauss upwind** instead of **Gauss linear** for the term **div(phi,U)** (`fvSchemes`). Do you get the same quantitative results?
- Sample the field variables **U** and **P** at a different location and plot the results using gnuplot.
- What density value do you think we were using? What about dynamic viscosity?

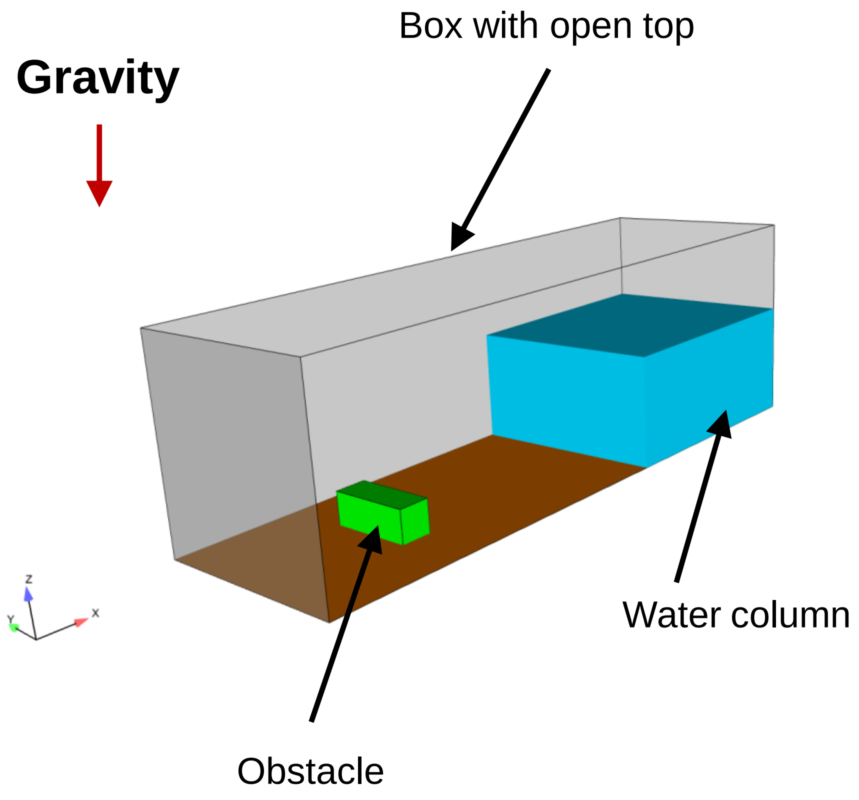
Hint: the physical pressure is equal to the modified pressure and $\nu = \mu/\rho$

Roadmap

1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. ~~Directory structure of an application/utility~~
4. ~~Applications/utilities in OpenFOAM®~~
5. ~~Directory structure of an OpenFOAM® case~~
6. ~~Running my first OpenFOAM® case setup blindfold~~
7. ~~A deeper view to my first OpenFOAM® case setup~~
8. **3D Dam break – Free surface flow**
9. Flow past a cylinder – From laminar to turbulent flow

3D Dam break – Free surface flow

Dam break free surface flow



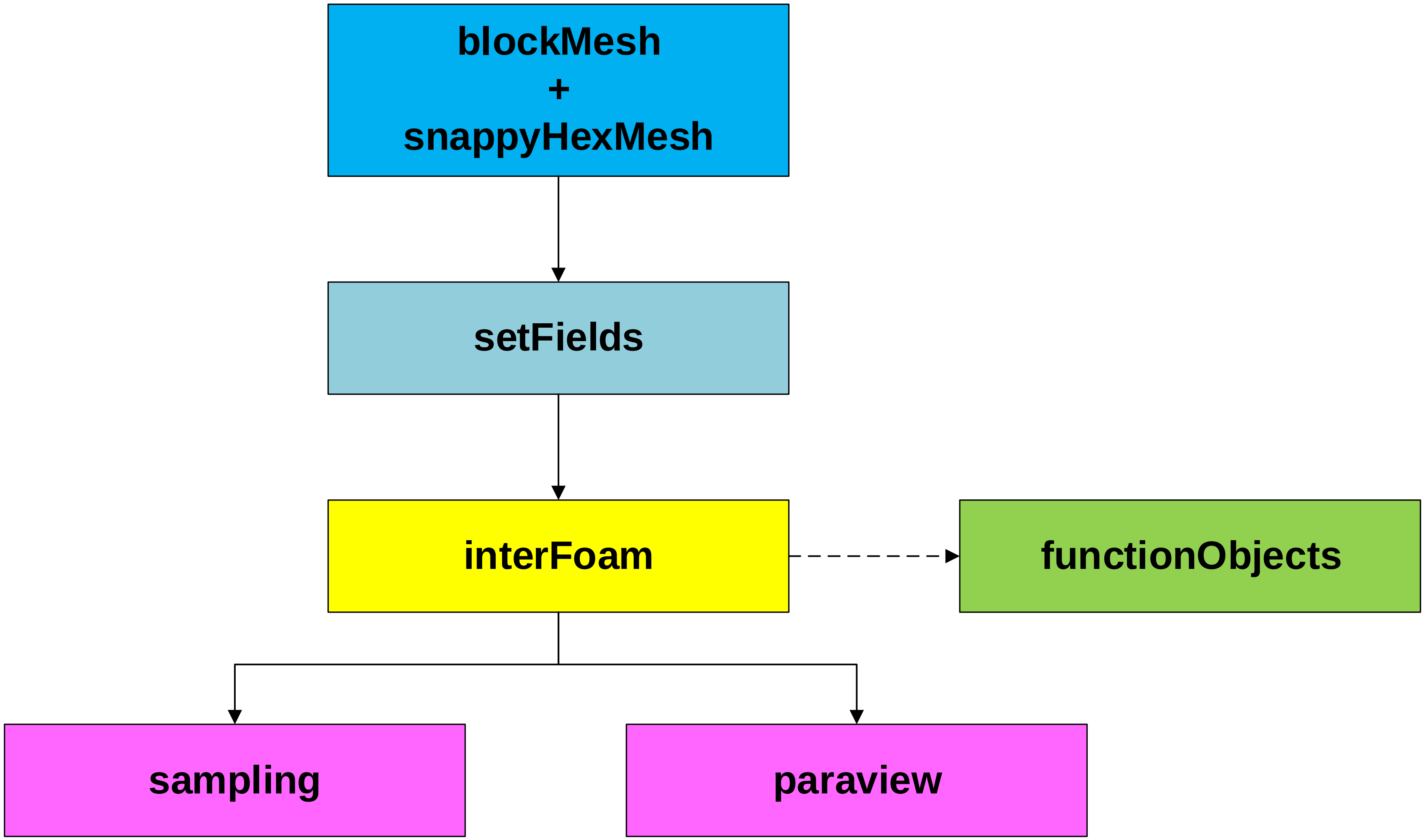
Physical and numerical side of the problem:

- In this case we are going to use the volume of fluid (VOF) method.
- This method solves the incompressible Navier-Stokes equations plus an additional equation to track the phases (free surface location).
- As this is a multiphase case, we need to define the physical properties for each phase involved (viscosity, density and surface tension).
- The working fluids are water and air.
- Additionally, we need to define the gravity vector and initialize the two flows.
- This is a three-dimensional and unsteady case.
- The details of the case setup can be found in the following reference:

  A Volume-of-Fluid Based Simulation Method for Wave Impact Problems.
  Journal of Computational Physics 206(1):363-393.
  June, 2005.

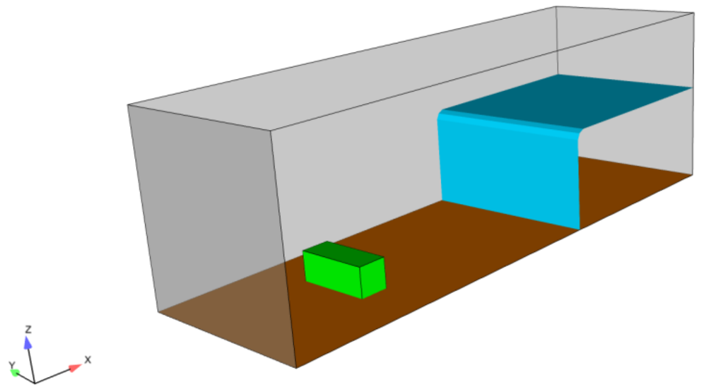
3D Dam break – Free surface flow

Workflow of the case

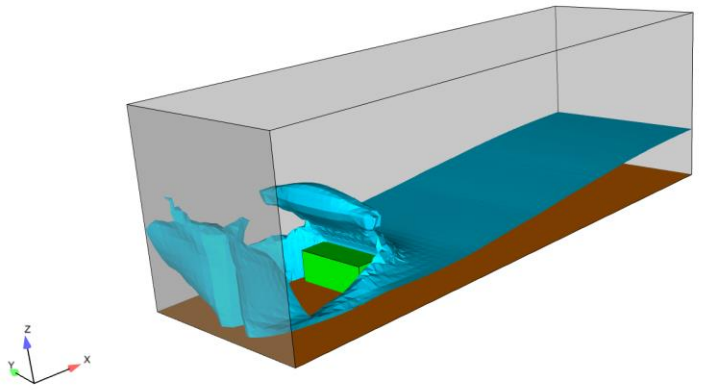


3D Dam break – Free surface flow

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



Initial conditions – Coarse mesh

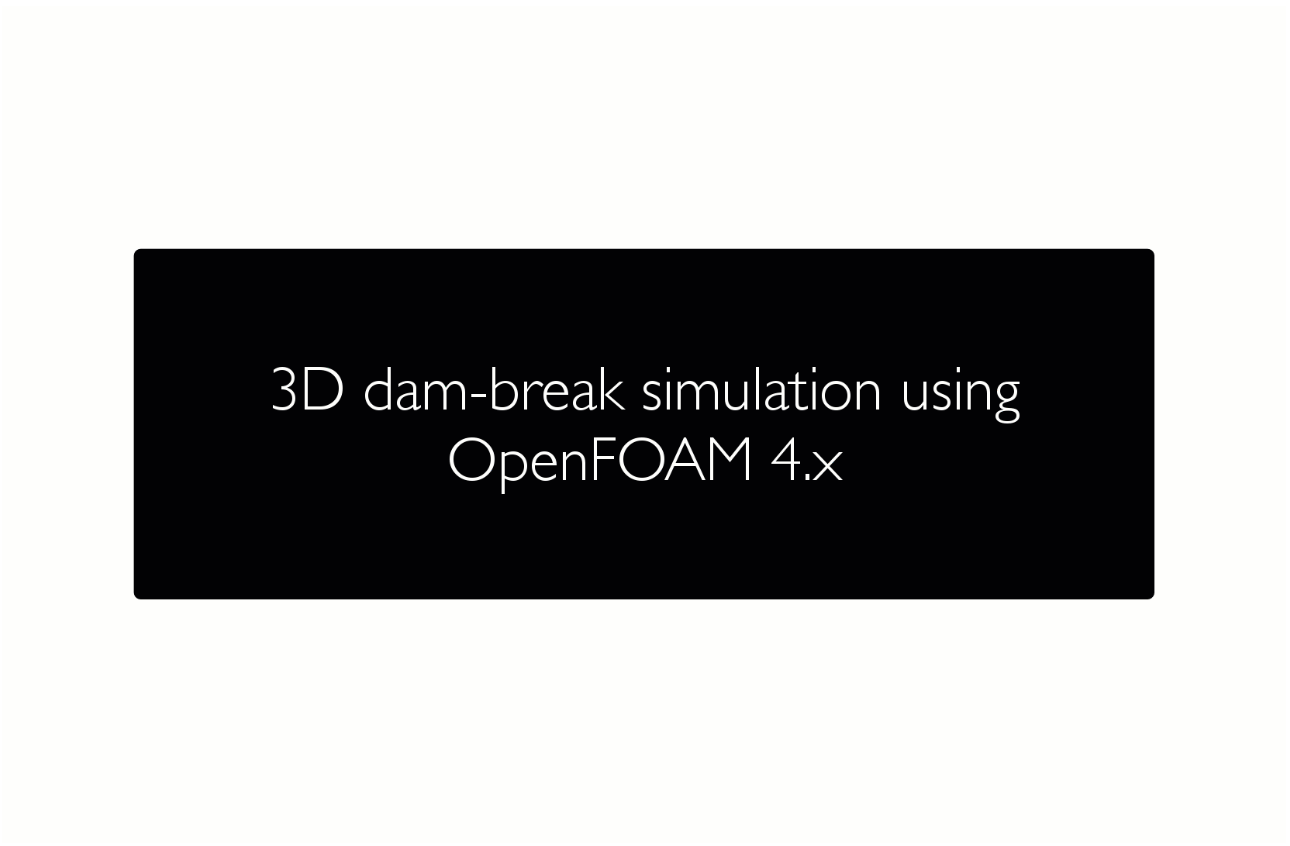


Solution at Time = 1 second – Coarse mesh

3D Dam break – Free surface flow

VOF Fraction (Free surface tracking) – Very fine mesh

http://www.wolfdynamics.com/validations/3d_db/dbreak.gif



3D Dam break – Free surface flow

- Let us run this case. Go to the directory:

```
$PTOFC/101OF/3d_damBreak
```

- In the case directory, you will find a few scripts with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on.
- These scripts can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver`
- These scripts are human-readable, and we highly recommend you open them, get familiar with the steps, and type the commands in the terminal. In this way, you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, run the cases automatically using these scripts.
- In the case directory, you will also find the `README.FIRST` file. In this file, you will find some additional comments.

3D Dam break – Free surface flow

What are we going to do?

- We will use this case to introduce the multiphase solver `interFoam`.
- `interFoam` is a solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach
- We will define the physical properties of two phases, and we are going to initialize these phases.
- We will define the gravity vector in the dictionary *`g`*.
- After finding the solution, we will visualize the results. This is an unsteady case so now we are going to see things moving.
- We are going to briefly address how to post-process multiphase flows.
- We are going to generate the mesh using snappyHexMesh, but for the purpose of this tutorial we are not going to discuss the dictionaries.
- Remember, different solvers have different input dictionaries.

3D Dam break – Free surface flow

The **`constant`** directory

- In this directory, we will find the following compulsory dictionary files:

  - *`g`*
  - *`transportProperties`*
  - *`momentumTransport`*

- *`g`* contains the definition of the gravity vector.
- *`transportProperties`* contains the definition of the physical properties of each phase.
- *`momentumTransport`* contains the definition of the turbulence model to use.

3D Dam break – Free surface flow

The g dictionary file

```
8     FoamFile
9     {
10        format      ascii;
11        class       uniformDimensionedVectorField;
12        object      g;
13    }

16    dimensions      [0 1 -2 0 0 0 0];
17    value           (0 0 -9.81);
```

- This dictionary file is located in the directory **`constant`**.
- For multiphase flows, this dictionary is compulsory.
- In this dictionary we define the gravity vector (line 17).
- Pay attention to the **class** type (line 11).

3D Dam break – Free surface flow

The *transportProperties* dictionary file

Primary phase

```
17    phases (water air);
18
19    water
20    {
21        transportModel  Newtonian;
22        nu              [0 2 -1 0 0 0 0] 1e-06;
23        rho             [1 -3 0 0 0 0 0] 1000;
24    }
25
26    air
27    {
28        transportModel  Newtonian;
29        nu              [0 2 -1 0 0 0 0] 1.48e-05;
30        rho             [1 -3 0 0 0 0 0] 1;
31    }
32
33    sigma           [1 0 -2 0 0 0 0] 0.07;
```

- This dictionary file is located in the directory **`constant`**.
- We first define the name of the phases (line 17). In this case we are defining the names **water** and **air**. The first entry in this list is the primary phase (**water**).
- The name of the primary phase is the one you will use to initialize the solution.
- The name of the phases is given by the user.
- In this file we set the kinematic viscosity (**nu**), density (**rho**) and transport model (**transportModel**) of the phases.
- We also define the surface tension (**sigma**).

3D Dam break – Free surface flow

The `momentumTransport` dictionary file

- In this dictionary file we select the turbulence model.
- In this case we use a RANS turbulence model (kEpsilon).

```
simulationType    RAS;

RAS
{
    RASModel kEpsilon;

    turbulence on;

    printCoeffs on;
}
```

3D Dam break – Free surface flow

The **0** directory

- In this directory, we find the input files that contain the boundary and initial conditions for all the primitive variables.
- As we are solving the incompressible RANS Navier-Stokes equations using the VOF method, we will find the following field files:

• *alpha.water*	(volume fraction of water phase)
• *p_rgh*	(pressure field minus hydrostatic component)
• *U*	(velocity field)
• *k*	(turbulent kinetic energy field)
• *epsilon*	(rate of dissipation of turbulence energy field)
• *nut*	(turbulence viscosity field)

3D Dam break – Free surface flow

The file *0/alpha.water*

```
17    dimensions       [0 0 0 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        front
24        {
25            type               zeroGradient;
26        }
27        back
28        {
29            type               zeroGradient;
30        }
31        left
32        {
33            type               zeroGradient;
34        }
35        right
36        {
37            type               zeroGradient;
38        }
39        bottom
40        {
41            type               zeroGradient;
42        }
43        top
44        {
45            type               inletOutlet;
46            inletValue         uniform 0;
47            value              uniform 0;
48        }
49        stlSurface
50        {
51            type                zeroGradient;
52        }
53
54    }
```

- This file contains the boundary and initial conditions for the non-dimensional scalar field **alpha.water**
- This file is named `alpha.water`, because the primary phase is water (we defined the primary phase in the `transportProperties` dictionary).
- Initially, this field is initialized as 0 in the whole domain (line 19). This means that there is no water in the domain at time 0. Later, we will initialize the water column and this file will be overwritten with a non-uniform field for the **internalField**.
- For the **front**, **back**, **left**, **right**, **bottom** and **stlSurface** patches we are using a **zeroGradient** boundary condition (we are just extrapolating the internal values to the boundary face).
- For the **top** patch we are using an **inletOutlet** boundary condition. This boundary condition avoids backflow into the domain. If the flow is going out it will use **zeroGradient** and if the flow is coming back it will assign the value set in the keyword **inletValue** (line 46).

3D Dam break – Free surface flow

The file *0/p_rgh*

```
17    dimensions      [1 -1 -2 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        front
24        {
25            type              fixedFluxPressure;
26            value             uniform 0;
27        }
28        back

33        left

38        right

43        bottom

48        top
49        {
50            type              totalPressure;
51            p0                uniform 0;
52            U                 U;
53            phi               phi;
54            rho               rho;
55            psi               none;
56            gamma             1;
57            value             uniform 0;
58        }
59        stlSurface
60        {
61            type              fixedFluxPressure;
62            value             uniform 0;
63        }
64
65    }
```

- This file contains the boundary and initial conditions for the dimensional scalar field **p_rgh**. The dimensions of this field are given in Pascal (line 17)
- This scalar field contains the value of the static pressure field minus the hydrostatic component.
- This field is initialized as 0 in the whole domain (line 19).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using the **fixedFluxPressure** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **totalPressure** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).

3D Dam break – Free surface flow

The file *0/U*

```
17    dimensions      [0 -1 -1 0 0 0 0];
18
19    internalField   uniform (0 0 0);
20
21    boundaryField
22    {
23        front
24        {
25            type            fixedValue;
26            value           uniform (0 0 0);
27        }
28        back

33        left

38        right

43        bottom

48        top
49        {
50            type            pressureInletOutletVelocity;
51            value           uniform (0 0 0);
52        }
53        stlSurface
54        {
55            type            fixedValue;
56            value           uniform (0 0 0);
57        }
58
59    }
```

- This file contains the boundary and initial conditions for the dimensional vector field **U**.
- We are using uniform initial conditions and the numerical value is **(0 0 0)** (keyword **internalField** in line 19).
- The **front**, **back**, **left, right, bottom** and **stlSurface** patches are no-slip walls, therefore we impose a **fixedValue** boundary condition with a value of **(0 0 0)** at the wall.
- For the **top** patch we are using the **pressureInlterOutletVelocity** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).

3D Dam break – Free surface flow

The file *0/k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.1;

boundaryField
{
    “(front|back|left|right|bottom|stlSurface)”
    {
        type            kqRWallFunction;
        value           $internalField;
    }

    top
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

}
```

- This file contains the boundary and initial conditions for the dimensional scalar field **k.**
- This scalar (turbulent kinetic energy), is related to the turbulence model.
- This field is initialized as 0.1 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using the **kqRWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **inletOutlet** boundary condition, this boundary condition handles backflow (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The file *0/epsilon*

```
dimensions      [0 2 -3 0 0 0 0];

internalField   uniform 0.1;

boundaryField
{
    "(front|back|left|right|bottom|stlSurface)"
    {
        type            epsilonWallFunction;
        value           $internalField;
    }

    top
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }

}
```

- This file contains the boundary and initial conditions for the dimensional scalar field **epsilon**.
- This scalar (rate of dissipation of turbulence energy), is related to the turbulence model.
- This field is initialized as 0.1 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using the **epsilonWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **inletOutlet** boundary condition, this boundary condition handles backflow (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The file *0/nut*

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

internalField   uniform 0;

boundaryField
{
    "(front|back|left|right|bottom|stlSurface)"
    {
        type            nutkWallFunction;
        value           $internalField;
    }

    top
    {
        type            calculated;
        value           $internalField;;
    }

}
```

- This file contains the boundary and initial conditions for the dimensional scalar field **nut**.
- This scalar (turbulent viscosity), is related to the turbulence model.
- This field is initialized as 0 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using the **nutkWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **calculated** boundary condition, this boundary condition computes the value of nut from k and epsilon (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The **`system`** directory

- 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.

3D Dam break – Free surface flow

The `controlDict` dictionary

```
17    application      interFoam;
18
19    startFrom        startTime;
20
21    startTime        0;
22
23    stopAt           endTime;
24
25    endTime          8;
26
27    deltaT           0.0001;
28
29    writeControl     adjustableRunTime;
30
31    writeInterval    0.02;
32
33    purgeWrite       0;
34
35    writeFormat      ascii;
36
37    writePrecision   8;
38
39    writeCompression uncompressed;
40
41    timeFormat       general;
42
43    timePrecision    8;
44
45    runTimeModifiable yes;
46
47    adjustTimeStep   yes;
48
49    maxCo            1.0;
50    maxAlphaCo       0.5;
51    maxDeltaT        0.01;
```

- This case starts from time 0 (**startTime**), and it will run up to 8 seconds (**endTime**).
- The initial time step of the simulation is 0.0001 seconds (**deltaT**).
- It will write the solution every 0.02 seconds (**writeInterval**) of simulation time (**runTime**). It will automatically adjust the time step (**adjustableRunTime**), in order to save the solution at the precise write interval.
- It will keep all the solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**).
- The write precision is 8 digits (**writePrecision**). It will only save eight digits in the output files.
- And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.
- In line 47 we turn on the option **adjustTimeStep**. This option will automatically adjust the time step to achieve the maximum desired courant number (lines 49-50). We also set a maximum time step in line 51.
- Remember, the first time step of the simulation is done using the value set in line 27 and then it is automatically scaled to achieve the desired maximum values (lines 49-51).

3D Dam break – Free surface flow

The `controlDict` dictionary

```
55    functions
56    {

          ...
79        minmaxdomain_scalar
          ...

          ...
111       minmaxdomain_vector
          ...

          ...
139       mindomain_scalar
          ...

          ...
145       mindomain_vector
          ...

          ...
151       maxdomain_scalar
          ...

          ...
157       maxdomain_vector
          ...

230   };
```

- Let us take a look at the monitors definition (**functionObjects)**.
- In lines 79-161 we define the following **functionObject:**
  - **minmaxdomain_scalar**
  - **minmaxdomain_vector**
  - **mindomain_scalar**
  - **mindomain_vector**
  - **maxdomain_scalar**
  - **maxdomain_vector**
- These **functionObject** are used to compute the minimum and maximum values of the field variables (**p p_rgh U alpha.water k epsilon**).
- Notice that we are not using packed **functionObject**.

3D Dam break – Free surface flow

The `controlDict` dictionary

```
55      functions
56      {

166      water_in_domain
167      {
168          type                volRegion;
169          functionObjectLibs ("libfieldFunctionObjects.so");
170
171          enabled             true;
172          log                 true;
173
174          //writeControl      outputTime;
175          writeControl    timeStep;
176          writeInterval   1;
177
178          writeFields       false;
179          writeLocation     false;
180
181          regionType        all;
182
183        operation        volIntegrate;
184         fields
185         (
186             alpha.water
187         );
188     }

230     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 166-188 we define the **water_in_domain functionObject** which computes the volume integral (**volIntegrate**) of the field variable **alpha.water** in all the domain.
- Basically, we are monitoring the quantity of water in the domain.

3D Dam break – Free surface flow

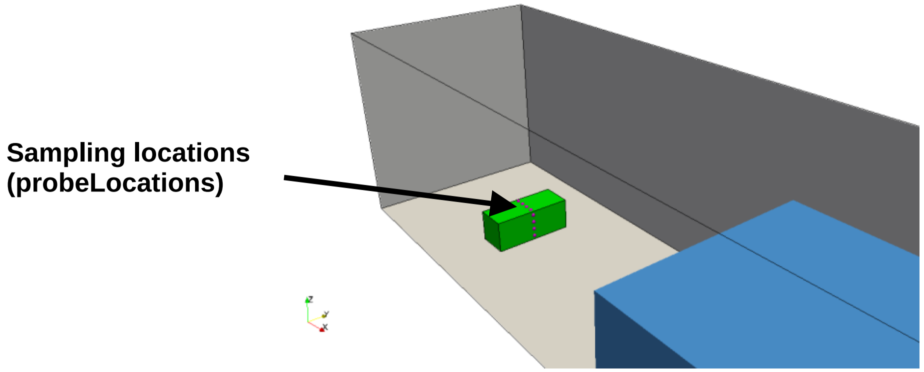
The `controlDict` dictionary

```
55      functions
56      {

193      probes1
194      {
195          type             probes;
196          functionObjectLibs ("libsampling.so");
197
198          pobeLocations
199          (
200              (0.82450002 0 0.021)
201              (0.82450002 0 0.061)
202              (0.82450002 0 0.101)
203              (0.82450002 0 0.141)
204              (0.8035 0 0.161)
205              (0.7635 0 0.161)
206              (0.7235 0 0.161)
207              (0.6835 0 0.161)
208          );
209
210         fields
211         (
212             p p_rgh
213         );
214
215          writeControl   timeStep;
216          writeInterval  1;
217     }

230     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 193-217 we define the **probes1 functionObject** which sample the selected fields (line 212) at the selected locations (lines 200-207).
- This sampling is done on-the-fly. All the information sample by this **functionObject** is saved in the directory **`./postProcessing/probes1`**
- As we are sampling starting from time 0, the sampled data will be located in the directory:

  `postProcessing/probes1/0`

- Feel free to open the files located in the directory **`postProcessing/probes1/0`** using your favorite text editor.



3D Dam break – Free surface flow

The `controlDict` dictionary

```
55      functions
56      {

221      yplus
222      {
223          type             yPlus;
224          functionObjectLibs ("libfieldFunctionObjects.so ");
225           enabled true;
226           writeControl outputTime;
227      }

230     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 221-227 we define the **yplus functionObject** which computes the y+ value.
- This quantity is related to turbulence modeling.
- This **functionObject** will save the yplus field in the solution directories with the same saving frequency as the solution (line 226).
- It will also save the minimum, maximum and mean values of y+ in the directory:

  `postProcessing/yplus`

3D Dam break – Free surface flow

The *fvSchemes* dictionary

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(U)         cellLimited Gauss linear 1;
}

divSchemes
{
    div(rhoPhi,U)  Gauss linearUpwindV grad(U);

    div(phi,alpha)  Gauss interfaceCompression vanLeer 1;

    div(phi,k) Gauss upwind;
    div(phi,epsilon) Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}
```

- In this case, for time discretization (**ddtSchemes**) we are using the **Euler** method.
- For gradient discretization (**gradSchemes**) we are using the **Gauss linear** as the default method and slope limiters (**cellLimited**) for the velocity gradient or **grad(U)**.
- For the discretization of the convective terms (**divSchemes**) we are using **linearUpwindV** interpolation method for the term **div(rhoPhi,U)**.
- For the term **div(phi,alpha)** we are using **interfaceCompression vanLeer** interpolation scheme.
  - This is an interface compression corrected scheme used to maintain sharp interfaces in VOF simulations.
  - The coefficient defines the degree of compression, where 1 is suitable for most VOF applications.
- For the terms **div(phi,k)** and **div(phi,epsilon)** we are using upwind (these terms are related to the turbulence modeling).
- **For the term div(((rho*nuEff)*dev2(T(grad(U)))))** we are using **linear** interpolation (this term is related to the turbulence modeling).
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear corrected** method
- In overall, this method is second order accurate but a little bit diffusive. Remember, at the end of the day we want a solution that is second order accurate.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
17    solvers
18    {
19        "alpha.water.*"
20        {
21            nAlphaCorr      2;
22            nAlphaSubCycles 1;
23            cAlpha          1;
24
25            MULESCorr       yes;
26            nLimiterIter    10;
27
35            solver          PBiCGStab;
36            smoother        DILU;
37            tolerance       1e-8;
38            relTol          0;
39        }
40
41        "(pcorr|pcorrFinal)"
42        {
43            solver          PCG;
44            preconditioner  DIC;
45            tolerance       1e-8;
46            relTol          0;
47        }
48
49        p_rgh
50        {
51            solver          PCG;
52            preconditioner  DIC;
53            tolerance       1e-06;
54            relTol          0.01;
55            minIter         2;
90        }
```

- To solve the volume fraction or **alpha.water** (lines 19-32) we are using the **smoothSolver** method.
- In line 25 we turn on the semi-implicit method MULES. The keyword **nLimiterIter** controls the number of MULES iterations over the limiter.
- To have more stability it is possible to increase the number of loops and corrections used to solve **alpha.water** (lines 21-22).
- The keyword **cAlpha** (line 23) controls the sharpness of the interface (1 is usually fine for most cases).
- In lines 41-47 we setup the solver for **pcorr** and **pcorrFinal** (pressure correction).
- In this case **pcorr** is solved only one time at the beginning of the computation.
- In lines 49-90 we setup the solver for **p_rgh**.
- The keyword **minIter 2** (line 55), means that the linear solver will do at least two iteration.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
92         p_rghFinal
93         {
94             $p_rgh;
96             relTol          0;
97             minIter         1;
98         }
99
100        "(U|UFinal)"
101        {
109            solver          PBiCGStab;
110            Preconditioner  DILU;
111            tolerance       1e-08;
112            relTol          0;
114        }
115
116         "(k|epsilon).*"
117         {
125             solver          PBiCGStab;
126             Preconditioner  DILU;
127             tolerance       1e-08;
128             relTol          0;
130         }
132    }
```

- In lines 92-98 we setup the solver for **p_rghFinal**. This corresponds to the last iteration in the loop (we can use a tighter convergence criteria to get more accuracy without increasing the computational cost)
- In lines 100-114 we setup the solvers for **U** and **UFInal**.
- In lines 116-130 we setup the solvers for the turbulent quantities, namely, **k** and **epsilon**.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
133
134    PIMPLE
135    {
136        Consistent          yes;
137        momentumPredictor    yes;
139        nOuterCorrectors     1;
140        nCorrectors          3;
141        nNonOrthogonalCorrectors 1;
142    }
143
144    relaxationFactors
145    {
146        fields
147        {
148            ".*" 0.9;
149        }
150        equations
151        {
152            ".*" 0.9;
153        }
154    }
155
```

- In lines 134-142 we setup the entries related to the pressure-velocity coupling method used (**PIMPLE** in this case). Setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- To gain more stability we can increase the number of correctors (lines 139-140), however this will increase the computational cost.
- In line 136, we use the consistent formulation of the **SIMPLE** method (used in the **PIMPLE** method outer iterations).
- In lines 144-154 we setup the under-relaxation factors related to the **PIMPLE** method outer iterations.
    - By using under-relaxation, we ensure diagonal equality.
    - Be careful not use too low values as you will lose time accuracy.
    - If you want to disable under-relaxation, comment out these lines.
- The option **momentumPredictor** (line 137), is recommended for highly convective flows.

3D Dam break – Free surface flow

The **`system`** directory

- In the **`system`** directory you will find the following optional dictionary files:
  - *`decomposeParDict`*
  - *`setFieldsDict`*

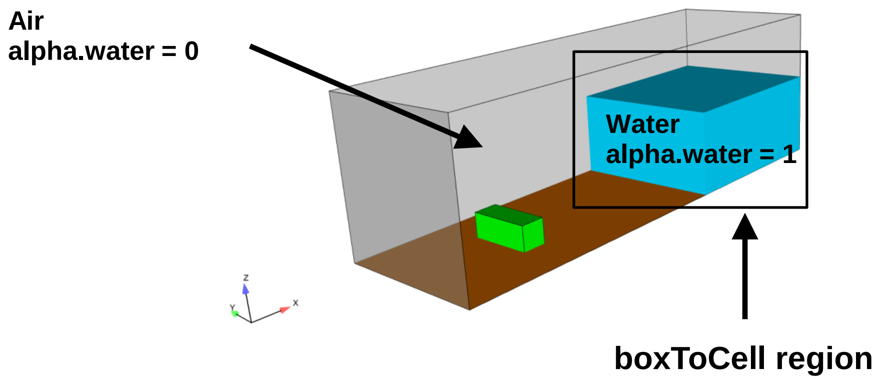
- *`decomposeParDict`* is read by the utility `decomposePar`. This dictionary file contains information related to the mesh partitioning. This is used when running in parallel.
- *`setFieldsDict`* is read by the utility `setFields`. This utility set values on selected cells/faces.

3D Dam break – Free surface flow

The *setFieldsDict* dictionary

```
17    defaultFieldValues
18    (
19        volScalarFieldValue alpha.water 0
20    );
21
22    regions
23    (
24        boxToCell
25        {
26            box (1.992 -10 0) (5 10 0.55);
27            fieldValues
28            (
29                volScalarFieldValue alpha.water 1
30            );
31        }
32    );
```

- This dictionary file is located in the directory **system**.
- In lines 17-20 we set the default value to be 0 in the whole domain (no water).
- In lines 22-32, we initialize a rectangular region (**box**) containing water (**alpha.water 1**).
- In this case, `setFields` will look for the dictionary file *alpha.water* and it will overwrite the original values according to the regions defined in *setFieldsDict*.
- We initialize the water phase because is the primary phase in the dictionary *transportProperties*.
- If you are interested in initializing the vector field **U**, you can proceed as follows **volVectorFieldValue U (0 0 0)**



3D Dam break – Free surface flow

The *decomposeParDict* dictionary

- This dictionary file is located in the directory **system**.
- This dictionary is used to decompose the domain in order to run in parallel.
- The keyword **numberOfSubdomains** (line 17) is used to set the number of cores we want to use in the parallel simulation.
- In this dictionary we also set the decomposition method (line 19).
- Most of the times the scotch method is fine.
- In this case we set the **numberOfSubdomains** to 4, therefore we will run in parallel using 4 cores.

```
17    numberOfSubdomains 4;
18
19    method scotch;
20
```

- When you run in parallel, the solution is saved in the directories **`processorN`**, where **`N`** stands for processor number. In this case you will find the following directories with the decomposed mesh and solution: **`processor0`**, **`processor1`**, **`processor2`**, and **`processor3`**.

3D Dam break – Free surface flow

Running the case

- Let us first generate the mesh.
- To generate the mesh will use snappyHexMesh (sHM), do not worry we will talk about sHM tomorrow.

```
1.  $> foamCleanTutorials
2.  $> rm –rf 0
3.  $> blockMesh
4.  $> surfaceFeatures
5.  $> snappyHexMesh -overwrite
6.  $> createPatch -dict system/createPatchDict.0 -overwrite
7.  $> createPatch -dict system/createPatchDict.1 -overwrite
8.  $> checkMesh
9.  $> paraFoam
```

3D Dam break – Free surface flow

Running the case

- Let us run the simulation in parallel using the solver `interFoam`.
- We will talk more about running in parallel tomorrow
- To run the case, type in the terminal:

```
1.  $> rm -rf 0
2.  $> cp -r 0_org 0
3.  $> setFields
4.  $> paraFoam
5.  $> decomposePar
6.  $> mpirun -np 4 interFoam -parallel | tee log.interFoam
7.  $> reconstructPar
8.  $> paraFoam
```

3D Dam break – Free surface flow

Running the case

- In steps 1-2 we copy the information of the backup directory **`0_org`** into the directory **`0`**. We do this because in the next step the utility `setFields` will overwrite the file *`0/alpha.water`*, so it is a good idea to keep a backup.
- In step 3 we initialize the solution using the utility `setFields`. This utility reads the dictionary *`setFieldsDict`* located in the **`system`** directory.
- In step 4 we visualize the initialization using paraFoam.
- In step 5 we use the utility `decomposePar` to do the domain decomposition needed to run in parallel.
- In step 6 we run the simulation in parallel. Notice that np means number of processors and the value used should be the same number as the one you set in the dictionary *`decomposeParDict`*.
- If you want to run in serial, type in the terminal: `interFoam | tee log`
- In step 7 we reconstruct the parallel solution. This step is only needed if you are running in parallel.
- Finally, in step 8 we visualize the solution.

3D Dam break – Free surface flow

- To plot the sampled data using gnuplot you can proceed as follows. To enter to the gnuplot prompt type in the terminal:

```
$> gnuplot
```

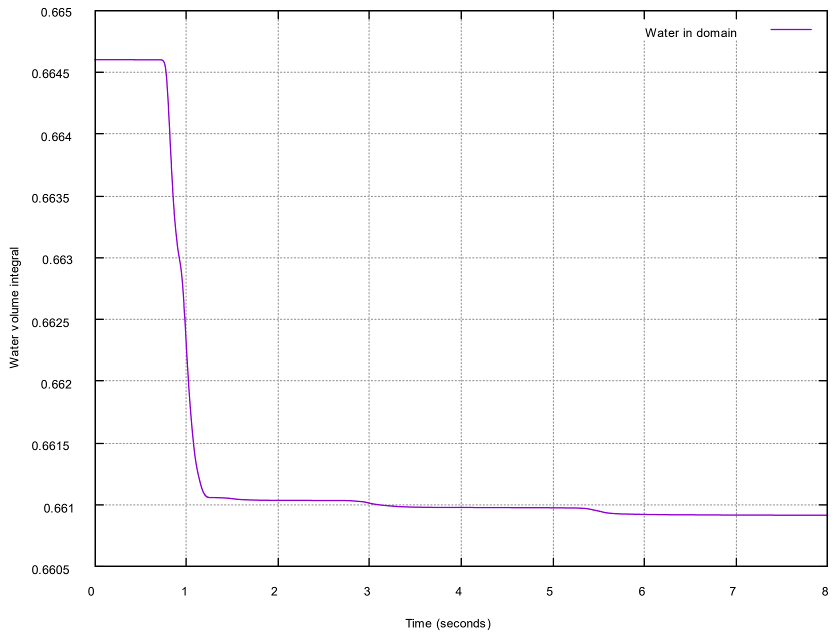
- Now that we are inside the gnuplot prompt, we can type,

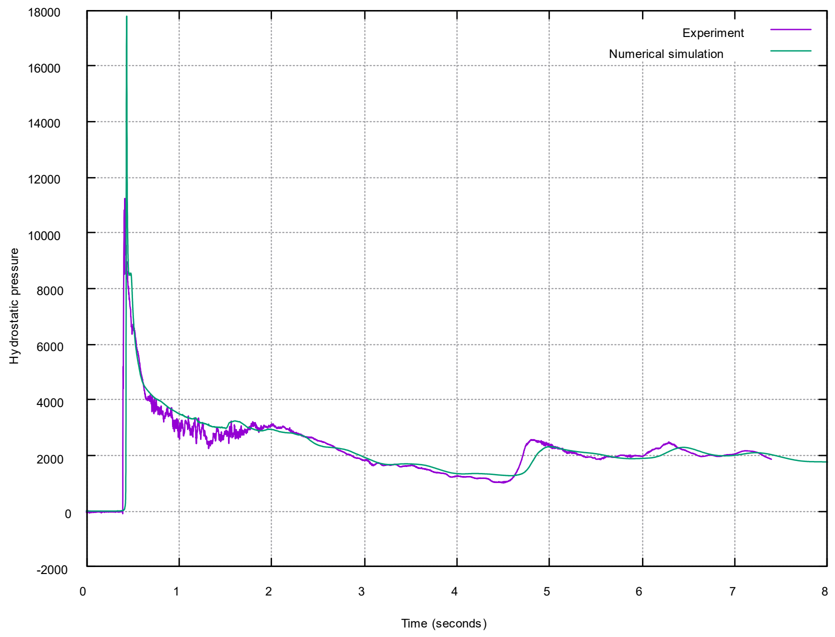
```
set xlabel 'Time (seconds)'

set ylabel 'Water volume integral'

gnuplot> plot 'postProcessing/water_in_domain/0/volRegion.dat' u 1:2 w l title
'Water in domain'
set xlabel 'Time (seconds)'

set ylabel 'Pressure'

plot 'SPHERIC_Test2/case.txt' u 1:2 w l title 'Experiment',
'postProcessing/probes1/0/p' u 1:2 w l title 'Numerical simulation'
gnuplot> exit
```

To exit gnuplot

3D Dam break – Free surface flow

- The output of steps 3 and 6 is the following:





alpha.water vs. time

p vs. time (at probe 0)

3D Dam break – Free surface flow

The output screen

```
Courant Number mean: 0.0099001831 max: 0.50908228
Interface Courant Number mean: 0.0012838336 max: 0.05362054
deltaT = 0.00061195165
Time = 0.41265658

PIMPLE: iteration 1
smoothSolver:  Solving for alpha.water, Initial residual = 0.00035163885, Final residual = 9.3476388e-11, No Iterations 2
Phase-1 volume fraction = 0.20706923  Min(alpha.water) = -9.1300674e-12  Max(alpha.water) = 1.0000113
MULES: Correcting alpha.water
MULES: Correcting alpha.water
MULES: Correcting alpha.water
Phase-1 volume fraction = 0.20706923  Min(alpha.water) = -1.2354076e-07  Max(alpha.water) = 1.0000113
DILUPBiCGStab:  Solving for Ux, Initial residual = 0.00057936556, Final residual = 2.3207684e-09, No Iterations 1
DILUPBiCGStab:  Solving for Uy, Initial residual = 0.0021990412, Final residual = 7.228845e-09, No Iterations 1
DILUPBiCGStab:  Solving for Uz, Initial residual = 0.00041048425, Final residual = 3.946807e-10, No Iterations 1
DICPCG:  Solving for p_rgh, Initial residual = 0.0013260985, Final residual = 1.2556023e-05, No Iterations 4
DICPCG:  Solving for p_rgh, Initial residual = 1.4873252e-05, Final residual = 8.7706547e-07, No Iterations 13
time step continuity errors : sum local = 2.166836e-08, global = -4.8300033e-11, cumulative = -5.8278026e-05
DICPCG:  Solving for p_rgh, Initial residual = 1.6925332e-05, Final residual = 8.9811533e-07, No Iterations 9
DICPCG:  Solving for p_rgh, Initial residual = 1.1731393e-06, Final residual = 4.991128e-07, No Iterations 1
time step continuity errors : sum local = 1.2328745e-08, global = -3.6165262e-09, cumulative = -5.8281643e-05
DICPCG:  Solving for p_rgh, Initial residual = 8.2834963e-07, Final residual = 4.6047958e-07, No Iterations 1
DICPCG:  Solving for p_rgh, Initial residual = 4.6053278e-07, Final residual = 4.65519e-07, No Iterations 1
time step continuity errors : sum local = 1.1498949e-08, global = -3.1908629e-09, cumulative = -5.8284834e-05
DILUPBiCGStab:  Solving for epsilon, Initial residual = 0.001169828, Final residual = 9.2601488e-11, No Iterations 2
DILUPBiCGStab:  Solving for k, Initial residual = 0.0014561556, Final residual = 9.4651262e-11, No Iterations 2
ExecutionTime = 23.21 s  ClockTime = 24 s

fieldMinMax minmaxdomain write:
    min(p) = -9.8942827 in cell 5509 at location (2.490155 0.025000016 1) on processor 2
    max(p) = 4703.3656 in cell 1485 at location (3.1948336 -0.425 0) on processor 2
    min(p_rgh) = -7.9025882 in cell 1241 at location (0.82088765 -0.20846334 0.043756428) on processor 1
    max(p_rgh) = 4831.247 in cell 3285 at location (3.1948341 -0.475 0.42499986) on processor 2
    min(U) = (-0.96505264 -0.019641482 -0.052664083) in cell 2 at location (2.1879167 -0.42500042 0.024999822) on processor 2
    max(U) = (0.32541708 0.29383224 2.7117589) in cell 5246 at location (0.8884354 0.087713417 0.16296979) on processor 1
    min(alpha.water) = -1.2354076e-07 in cell 2653 at location (0.84202094 -0.10628417 0.0062556498) on processor 1
    max(alpha.water) = 1.0000113 in cell 224 at location (2.6411358 -0.42500003 0.074999874) on processor 2
    min(k) = 0.0041733636 in cell 2510 at location (0.65789113 -0.0062500875 0.0062360099) on processor 1
    max(k) = 0.83402261 in cell 6589 at location (1.2803306 -0.025028634 0.17499623) on processor 1
    min(epsilon) = 0.018352121 in cell 2510 at location (0.65789113 -0.0062500875 0.0062360099) on processor 1
    max(epsilon) = 11.712212 in cell 1933 at location (0.83147515 -0.19630576 0.068753535) on processor 1

volFieldValue water_in_domain write:
    volIntegrate() of alpha.water = 0.66459985
```

Flow courant number

Interface courant number. When solving multiphase flows, is always desirable to keep the interface courant number less than 1.

alpha.water residuals

nAlphaSubCycles 1
Only one loop

nAlphaCorr 3

3 pressure correctors and no non-orthogonal corrections

Tighter tolerance (p_rghFinal) is only applied to this iteration (the final one)

Turbulence variables residuals

Minimum and maximum values of field variables

Volume integral functionObject

3D Dam break – Free surface flow

Post-processing multiphase flows in paraFoam

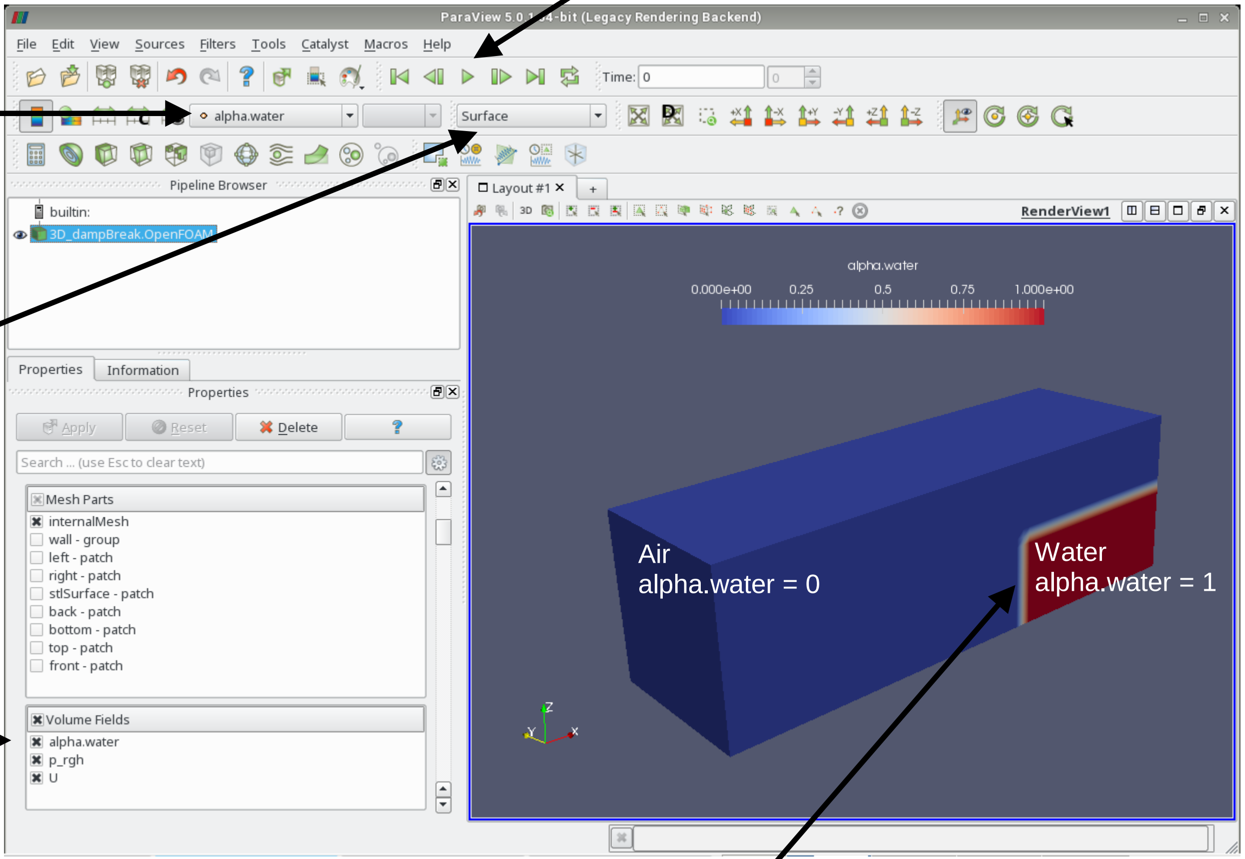
- To visualize the volume fraction, proceed as follows,

4. To animate the solution, press `Play` in the VCR Controls

2. Select **alpha.water** in the Active Variable drop-down menu

3. Select `Surface` in the Representation drop-down menu

1. In the Properties tab select **alpha.water** in Volume Fields



Interface
alpha.water = 0.5

3D Dam break – Free surface flow

Post-processing multiphase flows in paraFoam

- To visualize a surface representing the interface, proceed as follows,

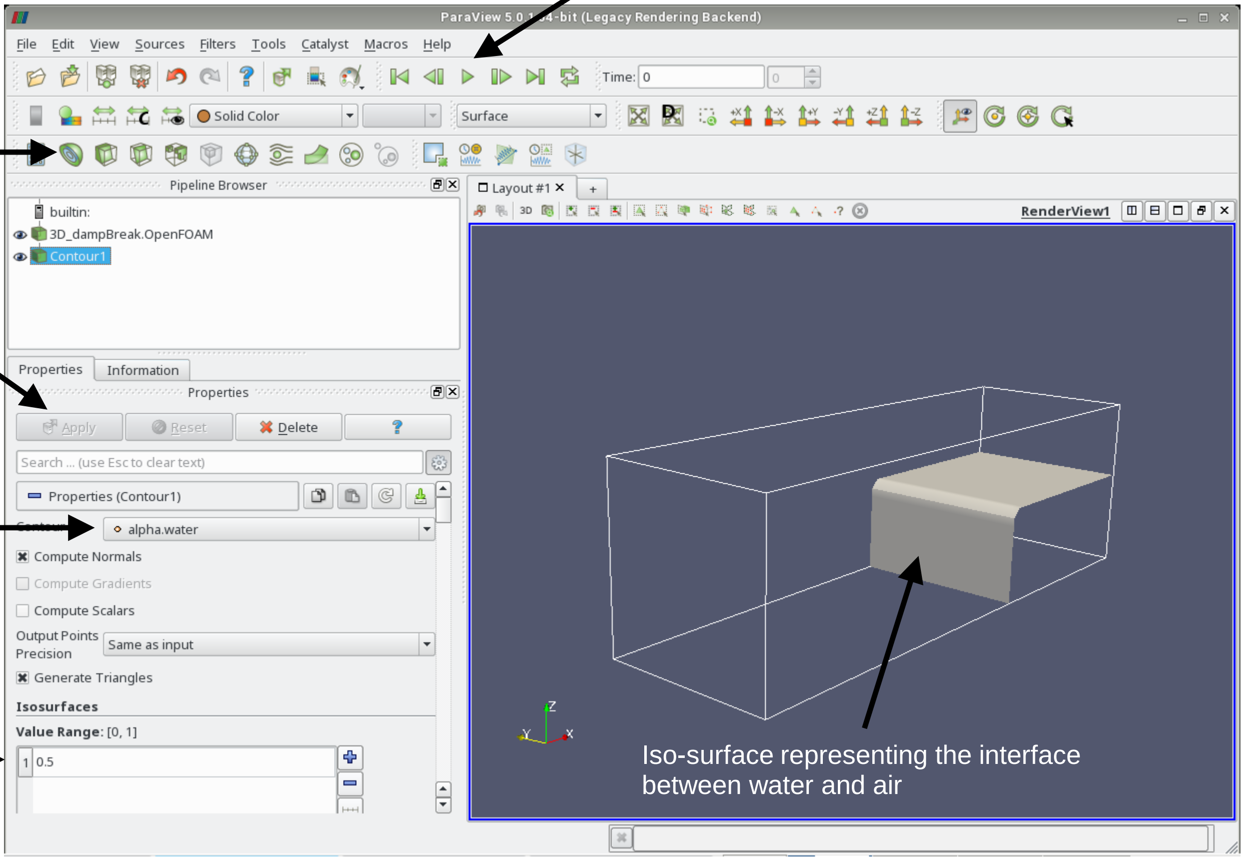
5. To animate the solution, press `Play` in the VCR Controls

1. Select the filter `Contour`

4. Press apply

2. Select **alpha.water** or the field you want to use to plot the iso-surface (it has to be a scalar)

3. Enter the value 0.5 which corresponds to the interface between water and air


ParaView 5.0.1 64-bit (Legacy Rendering Backend)
File Edit View Sources Filters Tools Catalyst Macros Help
Time: 0
Solid Color
Surface
Pipeline Browser
Layout #1
RenderView1
Properties Information
Apply Reset Delete
Search ... (use Esc to clear text)

builtin:
3D_dampBreak.OpenFOAM
Contour1

Properties (Contour1)

Contour By: alpha.water
[✓] Compute Normals
[ ] Compute Gradients
[ ] Compute Scalars
Output Points Precision: Same as input
[✓] Generate Triangles

Isosurfaces
Value Range: [0, 1]
1 0.5

Iso-surface representing the interface between water and air

3D Dam break – Free surface flow

Post-processing multiphase flows in paraFoam

- To visualize all the cells representing the water fraction, proceed as follows,

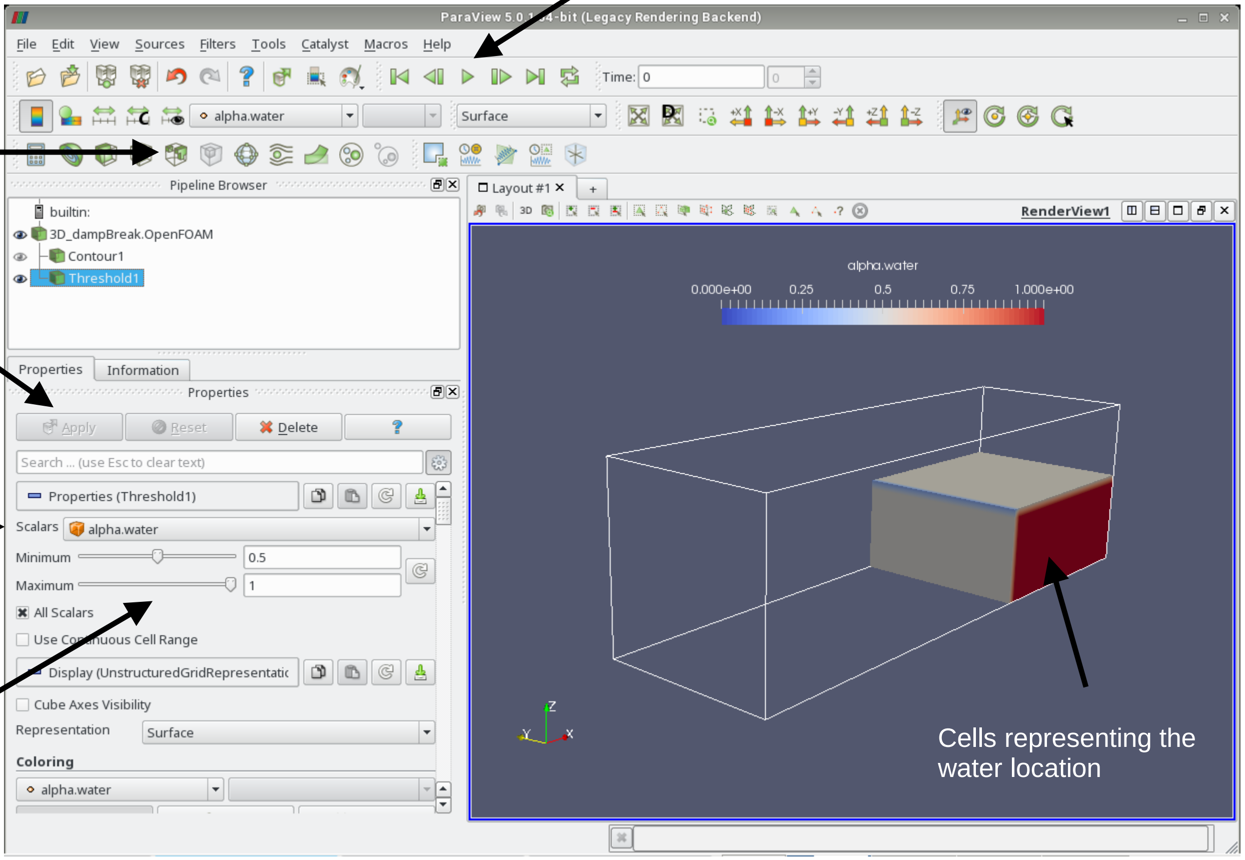
5. To animate the solution, press `Play` in the VCR Controls

1. Select the filter `Threshold`

4. Press apply

2. Select **alpha.water** or the field you want to use to visualize the cells (it has to be a scalar)

3. Select the range you want to visualize. To visualize the water select `Minimum` 0.5 and `Maximum` 1.


ParaView 5.0.1 64-bit (Legacy Rendering Backend)
File Edit View Sources Filters Tools Catalyst Macros Help
Time: 0
Pipeline Browser
builtin:
3D_dampBreak.OpenFOAM
Contour1
Threshold1
Properties Information
Apply Reset Delete
Search ... (use Esc to clear text)
Properties (Threshold1)
Display (UnstructuredGridRepresentation)
Layout #1
RenderView1
alpha.water
0.000e+00 0.25 0.5 0.75 1.000e+00

Scalars: alpha.water
Minimum: 0.5
Maximum: 1
☒ All Scalars
☐ Use Continuous Cell Range
☐ Cube Axes Visibility
Representation: Surface
Coloring: alpha.water

Cells representing the water location

3D Dam break – Free surface flow

Exercises

- Instead of using the boundary condition **totalPressure** and **pressureInletOutletVelocity** for the patch **top**, try use **zeroGradient**. Do you get the same results? Any comments?
  (Hint: this combination of boundary conditions might give you an error, if so, read carefully the screen and try to find a fix, you can start by looking at the file *`fvSolution`*)
- Instead of using the boundary condition **fixedFluxPressure** for the walls, try to use **zeroGradient**. Do you get the same results? Any comments?
- Run the simulation in a close domain. Does the volume integral of **alpha.water** remains the same? Why the value is not constant when the domain is open?
- Use **the utility** `postprocess` to measure the average pressure on the obstacle.
  (Hint: use the utility `postProcess` with patchAverage, take a look at module 5)
- Look for a **functionObject** to measure the average pressure on the obstacle (something similar to the method used in the previous question).
- How many initialization methods are there available in the dictionary *`setFieldsDict`*?
  (Hint: use the banana method)
- Run the simulation using **Gauss upwind** instead of **Gauss vanLeer** or **Gauss interfaceCompression vanLeer 1** for the term **div(phi,alpha)** (`fvSchemes`). Do you get the same quantitative results?

3D Dam break – Free surface flow

Exercises

- Run a numerical experiment for **cAlpha** equal to **0**, **1**, and **2**. Do you see any difference in the solution? What about computing time?
- Use the solver **GAMG** instead of using the solver **PCG** for the variable **p_rgh**. Do you see any difference on the solution or computing time?
- Increase the number of **nOuterCorrector** to 2 and study the output screen. What difference do you see?
- Turn off the MULES corrector (**MULESCorr**). Do you see any difference on the solution or computing time?
- If you set the gravity vector to (0 0 0), what do you think will happen?
- Try to break the solver and identify the cause of the error. You are free to try any kind of setup.

Roadmap

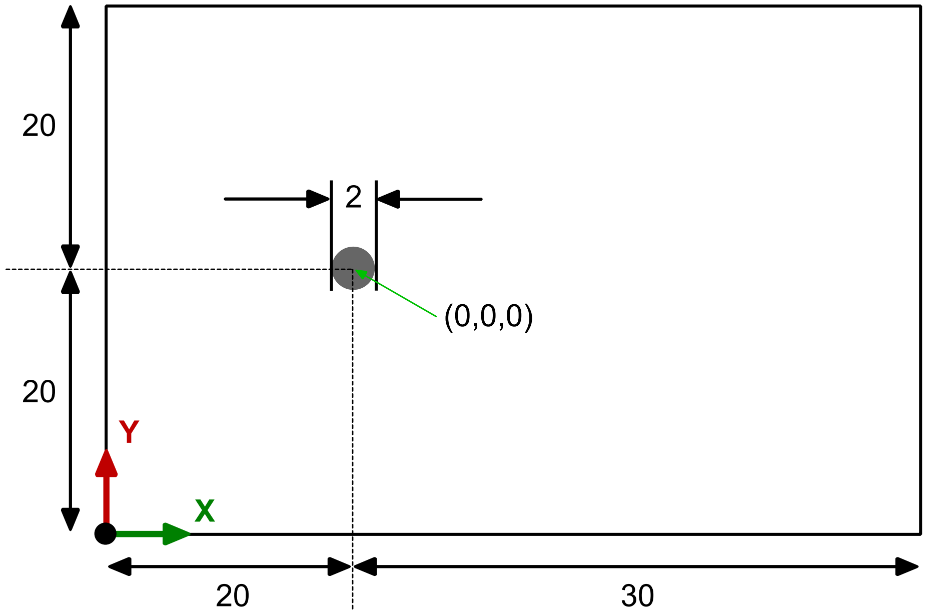
1. ~~OpenFOAM® brief overview~~
2. ~~OpenFOAM® directory organization~~
3. ~~Directory structure of an application/utility~~
4. ~~Applications/utilities in OpenFOAM®~~
5. ~~Directory structure of an OpenFOAM® case~~
6. ~~Running my first OpenFOAM® case setup blindfold~~
7. ~~A deeper view to my first OpenFOAM® case setup~~
8. ~~3D Dam break – Free surface flow~~
9. **Flow past a cylinder – From laminar to turbulent flow**

Flow past a cylinder – From laminar to turbulent flow

- At this point we all have a rough idea of what is going on with all these dictionary files.
- Unless it is strictly necessary, from now on we will not go into details about the dictionaries and files we are using.
- Remember, if you are using the lab computers, do not forget to load the environment variables.

Flow past a cylinder – From laminar to turbulent flow

Flow around a cylinder – 10 < Re < 2 000 000
Incompressible and compressible flow



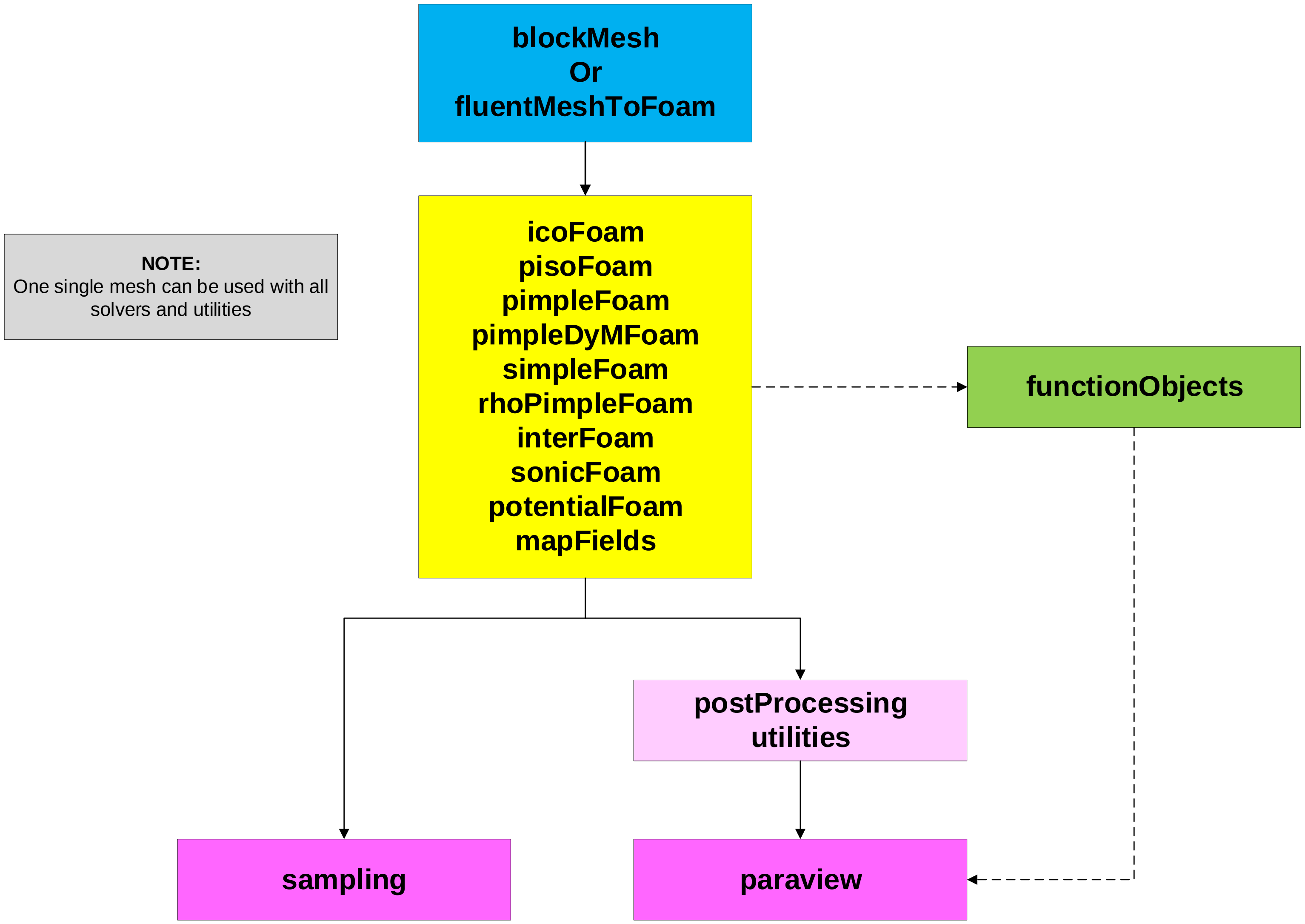
All the dimensions are in meters

Physical and numerical side of the problem:

- In this case we are going to solve the flow around a cylinder. We are going to use incompressible and compressible solvers, in laminar and turbulent regime.
- Therefore, the governing equations of the problem are the incompressible/compressible laminar/turbulent Navier-Stokes equations.
- We are going to work in a 2D domain.
- Depending on the Reynolds number, the flow can be steady or unsteady.
- This problem has a lot of validation data.

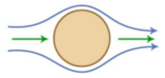
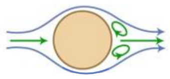
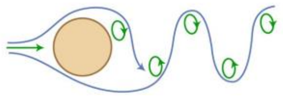
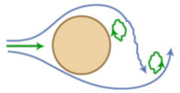
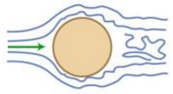
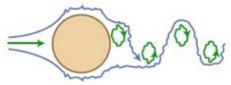
Flow past a cylinder – From laminar to turbulent flow

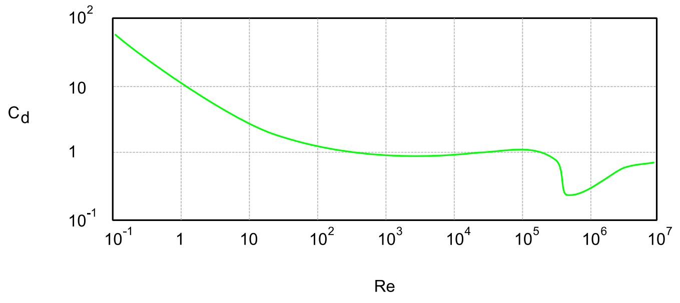
Workflow of the case



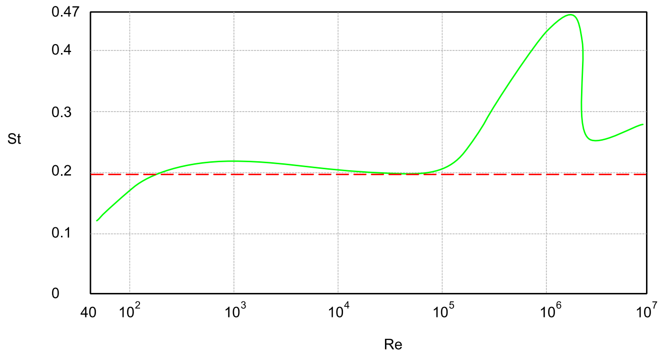
Flow past a cylinder – From laminar to turbulent flow

Vortex shedding behind a cylinder

	Creeping flow (no separation) Steady flow	$Re < 5$
	A pair of stable vortices in the wake Steady flow	$5 < Re < 40 - 46$
	Laminar vortex street (Von Karman street) Unsteady flow	$40 - 46 < Re < 150$
	Laminar boundary layer up to the separation point, turbulent wake Unsteady flow	$150 < Re < 300$ Transition to turbulence $300 < Re < 3 \times 10^5$
	Boundary layer transition to turbulent Unsteady flow	$3 \times 10^5 < Re < 3 \times 10^6$
	Turbulent vortex street, but the wake is narrower than in the laminar case Unsteady flow	$3 \times 10^6 > Re$



Drag coefficient



Strouhal number

Flow past a cylinder – From laminar to turbulent flow

Some experimental (E) and numerical (N) results of the flow past a circular cylinder at various Reynolds numbers

Reference	c_d – Re = 20	L_{rb} – Re = 20	c_d – Re = 40	L_{rb} – Re = 40
[1] Tritton (E)	2.22	–	1.48	–
[2] Cuntanceau and Bouard (E)	–	0.73	–	1.89
[3] Russel and Wang (N)	2.13	0.94	1.60	2.29
[4] Calhoun and Wang (N)	2.19	0.91	1.62	2.18
[5] Ye et al. (N)	2.03	0.92	1.52	2.27
[6] Fornbern (N)	2.00	0.92	1.50	2.24
[7] Guerrero (N)	2.20	0.92	1.62	2.21

L_{rb} = length of recirculation bubble, **c_d** = drag coefficient, **Re** = Reynolds number,

[1] D. Tritton. Experiments on the flow past a circular cylinder at low Reynolds numbers. Journal of Fluid Mechanics, 6:547-567, 1959.
[2] M. Cuntanceau and R. Bouard. Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 1. Steady flow. Journal of Fluid Mechanics, 79:257-272, 1973.
[3] D. Rusell and Z. Wang. A cartesian grid method for modeling multiple moving objects in 2D incompressible viscous flow. Journal of Computational Physics, 191:177-205, 2003.
[4] D. Calhoun and Z. Wang. A cartesian grid method for solving the two-dimensional streamfunction-vorticity equations in irregular regions. Journal of Computational Physics. 176:231-275, 2002.
[5] T. Ye, R. Mittal, H. Udaykumar, and W. Shyy. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. Journal of Computational Physics, 156:209-240, 1999.
[6] B. Fornberg. A numerical study of steady viscous flow past a circular cylinder. Journal of Fluid Mechanics, 98:819-855, 1980.
[7] J. Guerrero. Numerical simulation of the unsteady aerodynamics of flapping flight. PhD Thesis, University of Genoa, 2009.

Flow past a cylinder – From laminar to turbulent flow

Some experimental (E) and numerical (N) results of the flow past a circular cylinder at various Reynolds numbers

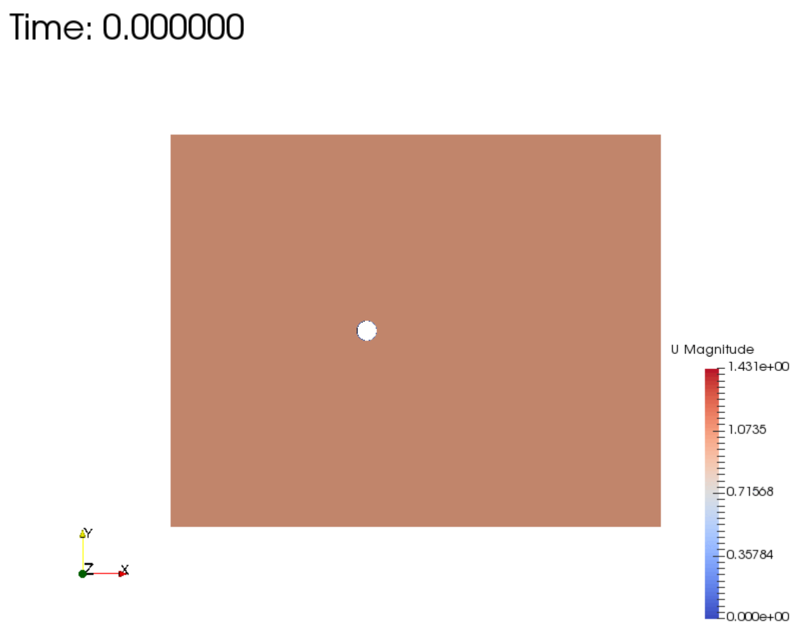
Reference	c_d – Re = 100	c_l – Re = 100	c_d – Re = 200	c_l – Re = 200
[1] Russel and Wang (N)	1.38 ± 0.007	± 0.322	1.29 ± 0.022	± 0.50
[2] Calhoun and Wang (N)	1.35 ± 0.014	± 0.30	1.17 ± 0.058	± 0.67
[3] Braza et al. (N)	1.386± 0.015	± 0.25	1.40 ± 0.05	± 0.75
[4] Choi et al. (N)	1.34 ± 0.011	± 0.315	1.36 ± 0.048	± 0.64
[5] Liu et al. (N)	1.35 ± 0.012	± 0.339	1.31 ± 0.049	± 0.69
[6] Guerrero (N)	1.38 ± 0.012	± 0.333	1.408 ± 0.048	± 0.725

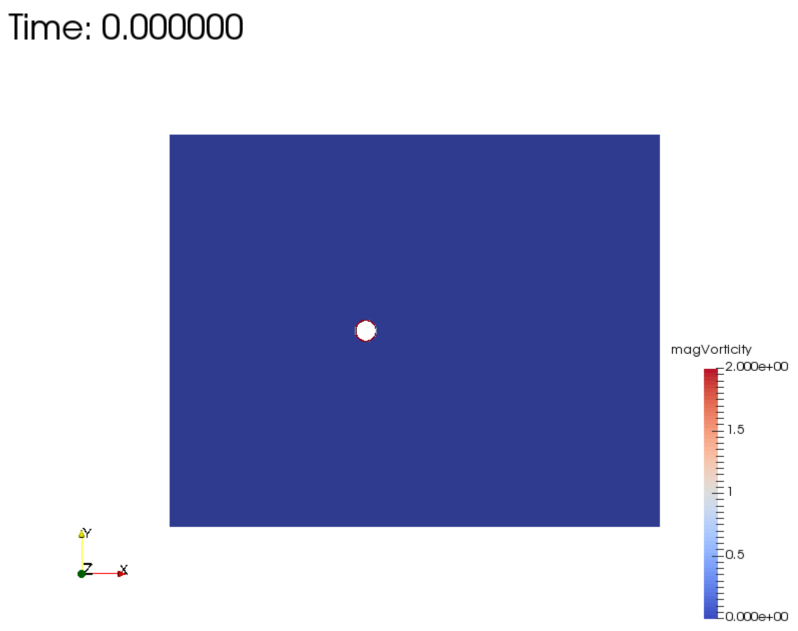
$\mathbf{c_l}$ = lift coefficient, $\mathbf{c_d}$ = drag coefficient, **Re** = Reynolds number

[1] D. Rusell and Z. Wang. A cartesian grid method for modeling multiple moving objects in 2D incompressible viscous flow. Journal of Computational Physics, 191:177-205, 2003.
[2] D. Calhoun and Z. Wang. A cartesian grid method for solving the two-dimensional streamfunction-vorticity equations in irregular regions. Journal of Computational Physics. 176:231-275, 2002.
[3] M. Braza, P. Chassaing, and H. Hinh. Numerical study and physical analysis of the pressure and velocity fields in the near wake of a circular cylinder. Journal of Fluid Mechanics, 165:79-130, 1986.
[4] J. Choi, R. Oberoi, J. Edwards, an J. Rosati. An immersed boundary method for complex incompressible flows. Journal of Computational Physics, 224:757-784, 2007.
[5] C. Liu, X. Zheng, and C. Sung. Preconditioned multigrid methods for unsteady incompressible flows. Journal of Computational Physics, 139:33-57, 1998.
[6] J. Guerrero. Numerical Simulation of the unsteady aerodynamics of flapping flight. PhD Thesis, University of Genoa, 2009.

Flow past a cylinder – From laminar to turbulent flow

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





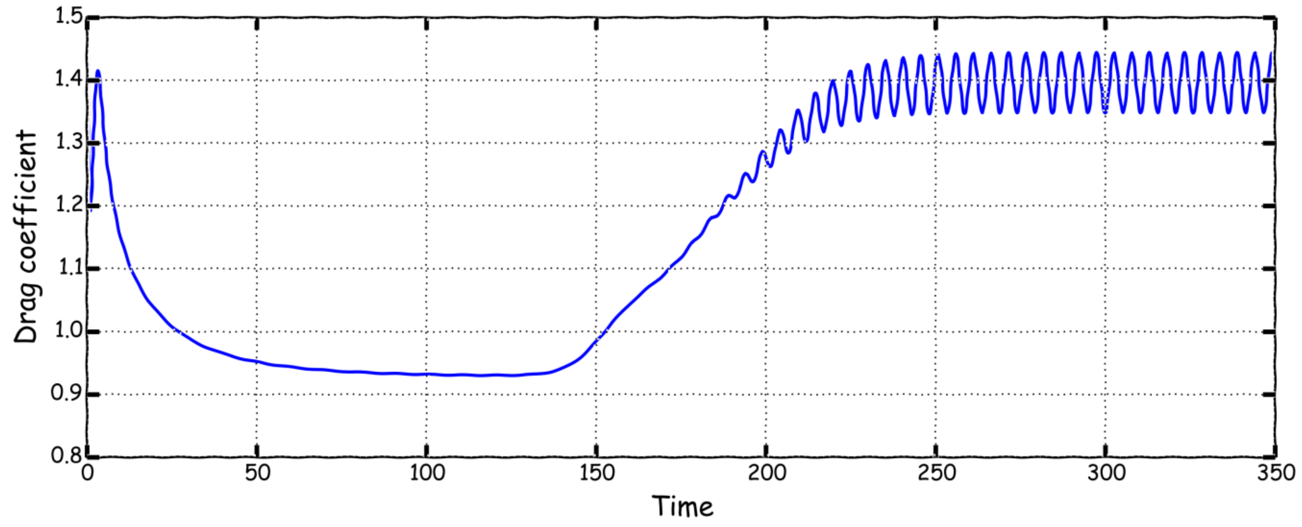
Instantaneous velocity magnitude field
www.wolfdynamics.com/wiki/cylinder_vortex_shedding/movvmag.gif

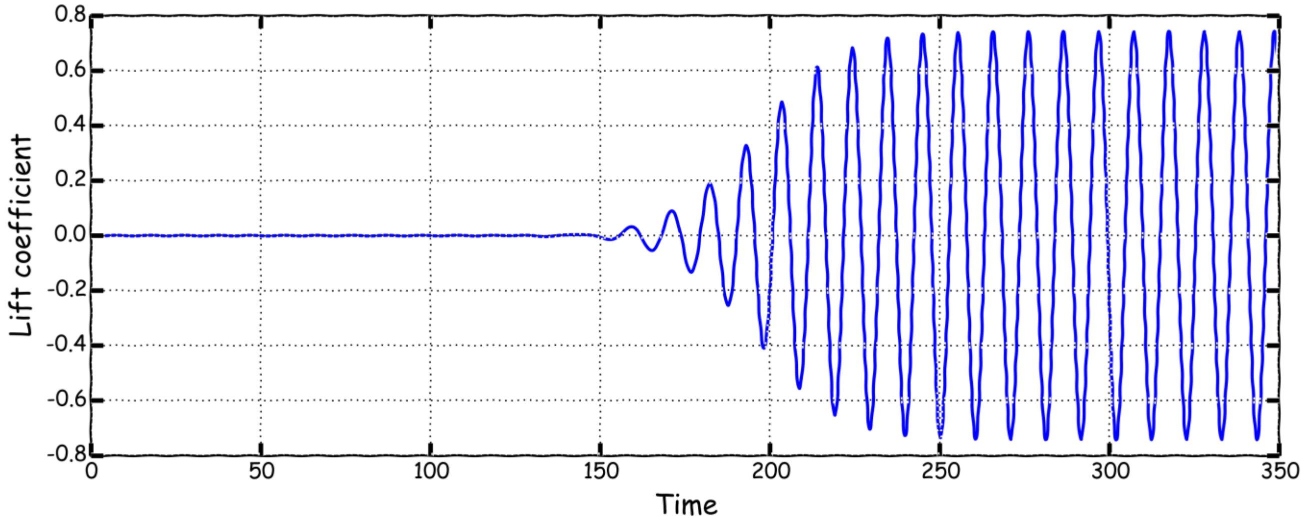
Instantaneous vorticity magnitude field
www.wolfdynamics.com/wiki/cylinder_vortex_shedding/movvort.gif

Incompressible flow – Reynolds 200

Flow past a cylinder – From laminar to turbulent flow

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





Incompressible flow – Reynolds 200

Flow past a cylinder – From laminar to turbulent flow

- Let us run this case. Go to the directory:

```
$PTOFC/101OF/vortex_shedding
```

- In the case directory, you will find a few scripts with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on.
- These scripts can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver`
- These scripts are human-readable, and we highly recommend you open them, get familiar with the steps, and type the commands in the terminal. In this way, you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, run the cases automatically using these scripts.
- In the case directory, you will also find the `README.FIRST` file. In this file, you will find some additional comments.

Flow past a cylinder – From laminar to turbulent flow

What are we going to do?

- We will use this case to learn how to use different solvers and utilities.
- Remember, different solvers have different input dictionaries.
- We will learn how to convert the mesh from a third-party software.
- We will learn how to use `setFields` to initialize the flow field and accelerate the convergence.
- We will learn how to map a solution from a coarse mesh to a fine mesh.
- We will learn how to setup a compressible solver.
- We will learn how to setup a turbulence case.
- We will use gnuplot to plot and compute the mean values of the lift and drag coefficients.
- We will visualize unsteady data.

Flow past a cylinder – From laminar to turbulent flow

Running the case

- Let us first convert the mesh from a third-party format (Fluent format).
- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c2`**
- In the terminal window type:

```
1.  $> foamCleanTutorials
2.  $> fluent3DMeshToFoam ../../../meshes_and_geometries/vortex_shedding/ascii.msh
3.  $> checkMesh
4.  $> paraFoam
```

- In step 2, we convert the mesh from Fluent format to OpenFOAM® format. Have in mind that the Fluent mesh must be in ascii format.
- If we try to open the mesh using `paraFoam` (step 4), it will crash. Can you tell what is the problem by just reading the screen?

Flow past a cylinder – From laminar to turbulent flow

Running the case

- To avoid this problem, type in the terminal,

1.
```
$> paraFoam -builtin
```

- Basically, the problem is related to the names and type of the patches in the file *boundary* and the boundary conditions (*U*, *p*). Notice that OpenFOAM® is telling you what and where is the error.

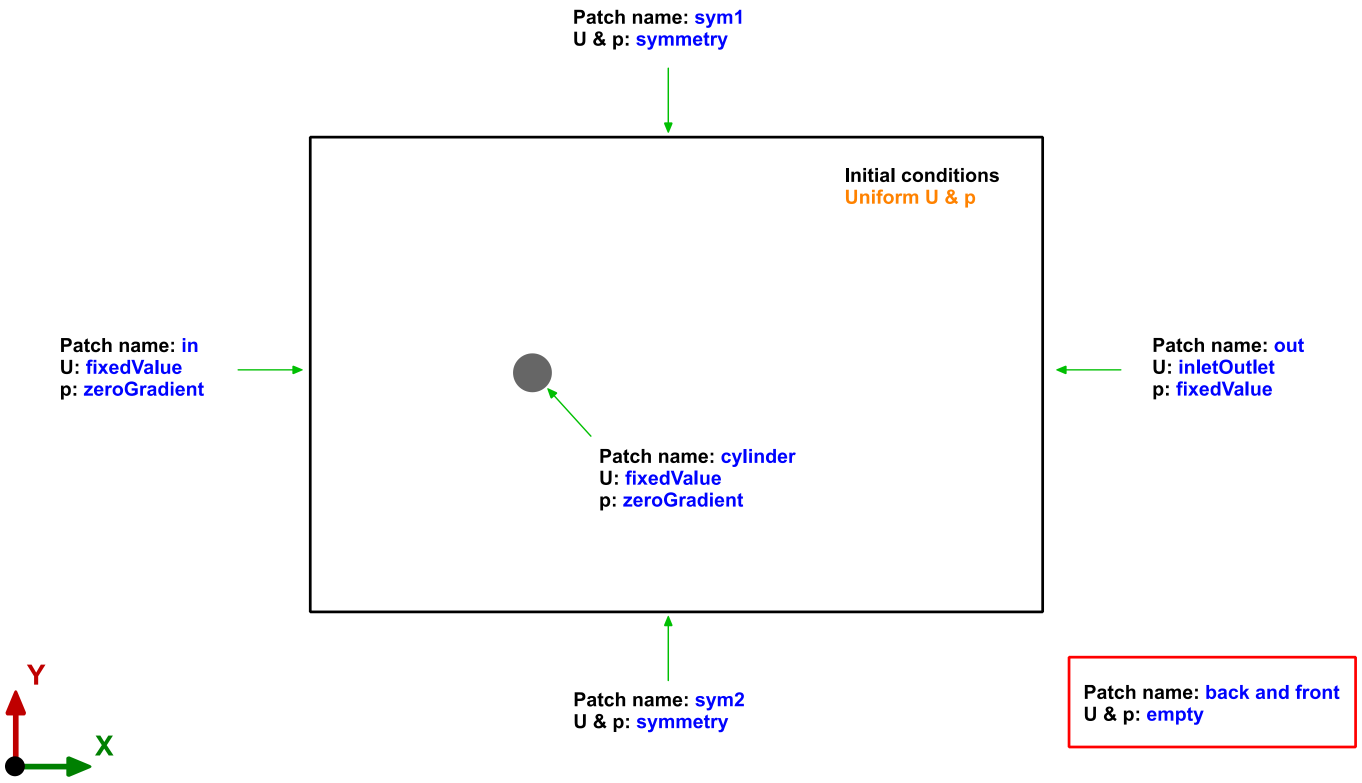
```
Created temporary 'c2.OpenFOAM'


--> FOAM FATAL IO ERROR:

    patch type 'patch' not constraint type 'empty'
    for patch front of field p in file "/home/joegi/my_cases_course/8/101OF/vortex_shedding/c2/0/p"

file: /home/joegi/my_cases_course/8/101OF/vortex_shedding/c2/0/p.boundaryField.front from line 60 to line 60.

    From function Foam::emptyFvPatchField<Type>::emptyFvPatchField(const Foam::fvPatch&, const
Foam::DimensionedField<Type, Foam::volMesh>&, const Foam::dictionary&) [with Type = double]
    in file fields/fvPatchFields/constraint/empty/emptyFvPatchField.C at line 80.

FOAM exiting
```

$\leftarrow$ **What**

$\leftarrow$ **Where**

Flow past a cylinder – From laminar to turbulent flow

- Remember, when converting meshes the **name** and **type** of the patches are not always set as you would like, so it is always a good idea to take a look at the file *`boundary`* and modify it according to your needs.
- Let us modify the *`boundary`* dictionary file.
- In this case, we want to setup the following **numerical type** boundary conditions.

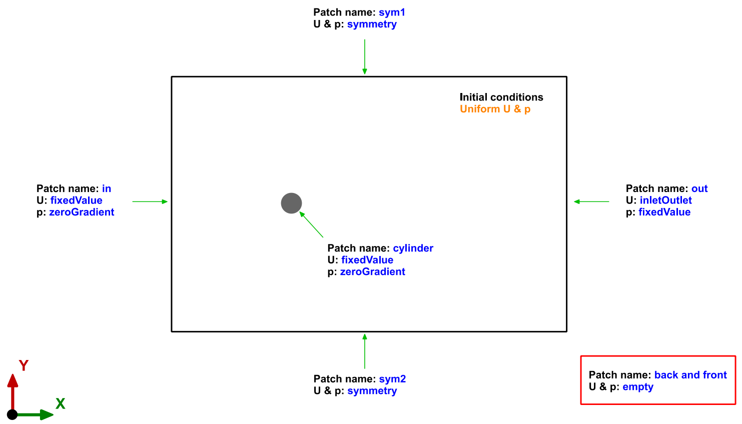


Flow past a cylinder – From laminar to turbulent flow

The *boundary* dictionary file

```
7
(
    out
    {
        type            patch;
        nFaces          80;
        startFace       18180;
    }
    sym1
    {
        type            symmetry;
        inGroups        1(symmetry);
        nFaces          100;
        startFace       18260;
    }
    sym2
    {
        type            symmetry;
        inGroups        1(symmetry);
        nFaces          100;
        startFace       18360;
    }
    in
    {
        type            patch;
        nFaces          80;
        startFace       18460;
    }
```

- This dictionary is located in the **`constant/polyMesh`** directory.
- This file is automatically created when converting or generating the mesh.
- To get a visual reference of the patches, you can visualize the mesh with paraFoam/paraview.
- The type of the **out** patch is OK.
- The type of the **sym1** patch is OK.
- The type of the **sym2** patch is OK.
- The type of the **in** patch is OK.

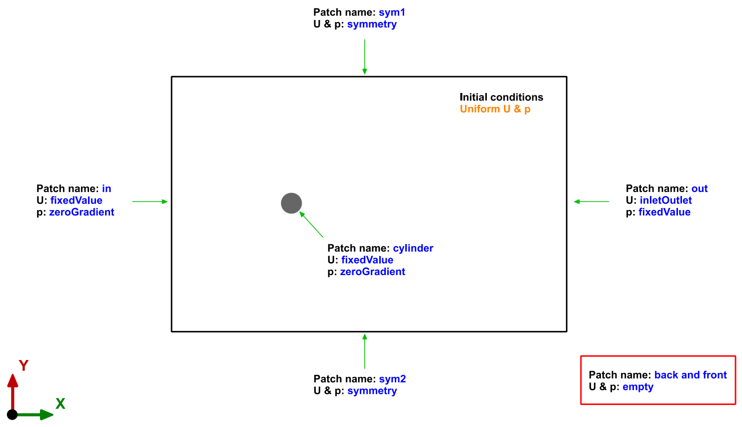


Flow past a cylinder – From laminar to turbulent flow

The *boundary* dictionary file

```
        cylinder
        {
            type            wall;
            inGroups        1(wall);
            nFaces          80;
            startFace       18540;
        }
        back
        {
            type            patch;
            nFaces          9200;
            startFace       18620;
        }
        front
        {
            type            patch;
            nFaces          9200;
            startFace       27820;
        }
    )
```

- The type of the **cylinder** patch is OK.
- The type of the **back** patch is **NOT OK**. Remember, this is a 2D simulation, therefore the type should be **empty**.
- The type of the **front** patch is **NOT OK**. Remember, this is a 2D simulation, therefore the type should be **empty**.
- Remember, we assign the **numerical type** boundary conditions (numerical values), in the field files found in the directory ***0***



Flow past a cylinder – From laminar to turbulent flow

- One of the most difficult things to understand in OpenFOAM is how to setup boundary conditions.
- There are too many and can be used in many combinations.
- To simplify things, it is better to think only in Dirichlet and Neumann boundary conditions. Therefore, we only need to deal with two types of boundary conditions (mathematically speaking).
- In the table below, we list the most frequent boundary conditions.
- Remember, you can always browse the source code or use the utility `foamInfo` to get more information about the boundary conditions.

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

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

wall	**type fixedValue;**
	value uniform (0 0 0);
		zeroGradient

Flow past a cylinder – From laminar to turbulent flow

- One of the most difficult things to understand in OpenFOAM is how to setup boundary conditions.
- There are too many and can be used in many combinations.
- To simplify things, it is better to think only in Dirichlet and Neumann boundary conditions. Therefore, we only need to deal with two types of boundary conditions (mathematically speaking).
- In the table below, we list the most frequent boundary conditions.
- Remember, you can always browse the source code or use the utility `foamInfo` to get more information about the boundary conditions.

`constant/polyMesh/boundary`	*`0/U - 0/p (IC/BC)`*
patch	**calculated** **fixedValue** **flowRateInletVelocity** **freestream** **inletOutlet** **slip** **supersonicFreeStream** **totalPressure** **zeroGradient**

Flow past a cylinder – From laminar to turbulent flow

- And when dealing with turbulence modeling, these are the most often used boundary conditions.
- To use these boundary conditions (wall functions) and to be able to compute y+, the primitive patch (the patch type defined in the `boundary` dictionary), must be of type **wall**.
- That is, these boundary conditions only apply to walls (moving or fixed).
- We will talk more about this when dealing with turbulence modeling.

Field	Wall functions – High RE	Resolved BL – Low RE
nut	**nut(–)WallFunction*** or **nutUSpaldingWallFunction**** (with 0 or a small number)	**nutUSpaldingWallFunction**, nutkWallFunction, nutUWallFunction, nutLowReWallFunction** or **fixedValue***** (with 0 or a small number)
k, q, R	**kqRWallFunction**** (with inlet value or a small number) $k_{wall} = k$	**kqRWallFunction**** or **kLowReWallFunction** (with inlet value, 0, or a small number) or **fixedValue***** (with 0 or a small number)
epsilon	**epsilonWallFunction** (with inlet value) $\epsilon_{wall} = \epsilon$	**epsilonWallFunction** (with inlet value) or **zeroGradient***** or **fixedValue***** (with 0 or a small number)
omega	**omegaWallFunction**** (with a large number) $\omega_{wall} = 10\frac{6\nu}{\beta y^2} \qquad \beta = 0.075$	**omegaWallFunction**** or **fixedValue***** (both with a large number)
nuTilda	–	**fixedValue** (one to ten times the molecular viscosity, a small number, or 0)

* nutUWallFunction or nutkWallfunction

** Recommended options for y^+ insensitive treatment (continuous wall functions)

*** Will disable wall functions. The equations will be integrated down to the viscous sublayer with no damping or corrections.

Flow past a cylinder – From laminar to turbulent flow

- At this point, check that the **name** and **type** of the **base type** boundary conditions and **numerical type** boundary conditions are consistent. If everything is ok, you are ready to go.
- Do not forget to explore the rest of the dictionary files, namely:
  - *`0/p`* (**p** is defined as relative pressure)
  - *`0/U`*
  - `constant/`*`transportProperties`*
  - *`system/controlDict`*
  - *`system/fvSchemes`*
  - *`system/fvSolution`*
- Reminder:
  - The diameter of the cylinder is 2.0 m.
  - And we are targeting for a Re = 200.

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

Flow past a cylinder – From laminar to turbulent flow

Running the case

- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c2`**
- In the folder **`c1`** you will find the same setup, but to generate the mesh we use `blockMesh` (the mesh is identical).
- To run this case, in the terminal window type:

1.
```
$> renumberMesh -overwrite
```
2.
```
$> icoFoam | tee log.icofoam
```
3.
```
$> pyFoamPlotWatcher.py log.icofoam
```
You will need to launch this script in a different terminal

4.
```
$> gnuplot scripts0/plot_coeffs
```
You will need to launch this script in a different terminal

5.
```
$> paraFoam
```

Flow past a cylinder – From laminar to turbulent flow

Running the case

- In step 1 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers. This is inexpensive (even for large meshes), therefore is highly recommended to always do it.
- In step 2 we run the simulation and save the log file. Notice that we are sending the job to background.
- In step 3 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
- In step 4 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.
- The force coefficients are computed using **functionObjects**.
- After the simulation is over, we use `paraFoam` to visualize the results. Remember to use the VCR Controls to animate the solution.
- In the folder **`c1`** you will find the same setup, but to generate the mesh we use `blockMesh` (the mesh is identical).

Flow past a cylinder – From laminar to turbulent flow

- At this point try to use the following utilities. In the terminal type:
  - `$> postProcess -func vorticity -noZero`
    This utility will compute and write the vorticity field. The –`noZero` option means do not compute the vorticity field for the solution in the directory **`0`**. If you do not add the `-noZero` option, it will compute and write the vorticity field for all the saved solutions, including **`0`**
  - `$> postprocess -func 'grad(U)' -latestTime`
    This utility will compute and write the velocity gradient or `grad(U)` in the whole domain (including at the walls). The `-latestTime` option means compute the velocity gradient only for the last saved solution.
  - `$> postprocess -func 'grad(p)'`
    This utility will compute and write the pressure gradient or `grad(U)` in the whole domain (including at the walls).
  - `$> foamToVTK -time 50:300`
    This utility will convert the saved solution from OpenFOAM® format to VTK format. The `-time 50:300` option means convert the solution to VTK format only for the time directories **`50`** to **`300`**

These last three will give you and error message, try to fix it.

  - `$> postProcess -func 'div(U)'`
    This utility will compute and write the divergence of the velocity field or `grad(U)` in the whole domain (including at the walls).
  - `$> pisoFoam -postProcess -func CourantNo`
    This utility will compute and write the Courant number. This utility needs to access the solver database for the physical properties and additional quantities; therefore, we need to tell what solver we are using. As the solver `icoFoam` does not accept the option `-postProcess`, we can use the solver `pisoFoam` instead. Remember, `icoFoam` is a fully laminar solver and `pisoFoam` is a laminar/turbulent solver.
  - `$> pisoFoam -postProcess -func wallShearStress`
    This utility will compute and write the wall shear stresses at the walls. As no arguments are given, it will save the wall shear stresses for all time-steps.

Flow past a cylinder – From laminar to turbulent flow

Non-uniform field initialization

- In the previous case, it took about 150 seconds of simulation time to onset the instability.
- If you are not interested in the initial transient or if you want to speed-up the computation, you can add a perturbation in order to trigger the onset of the instability.
- Let us use the utility `setFields` to initialize a non-uniform flow.
- This case is already setup in the directory ,

  `$PTOFC/101OF/vortex_shedding/c3`

- As you saw in the previous example, `icoFoam` is a very basic solver that does not have access to all the advanced modeling or postprocessing capabilities that comes with OpenFOAM®.
- Therefore, instead of using `icoFoam` we will use `pisoFoam` (or `pimpleFoam`) from now on.
- To run the solver `pisoFoam` (or `pimpleFoam`) starting from the directory structure of an `icoFoam` case, you will need to add the followings modifications:
  - Add the file *`momentumTransport`* in the directory **`constant`**.
  - Add the **transportModel** to be used in the file *`constant/transportProperties`*.
  - Add the entry **div((nuEff*dev2(T(grad(U))))) Gauss linear;** to the dictionary *`system/fvSchemes`* in the section **divSchemes** (this entry is related to the Reynodls stresses).

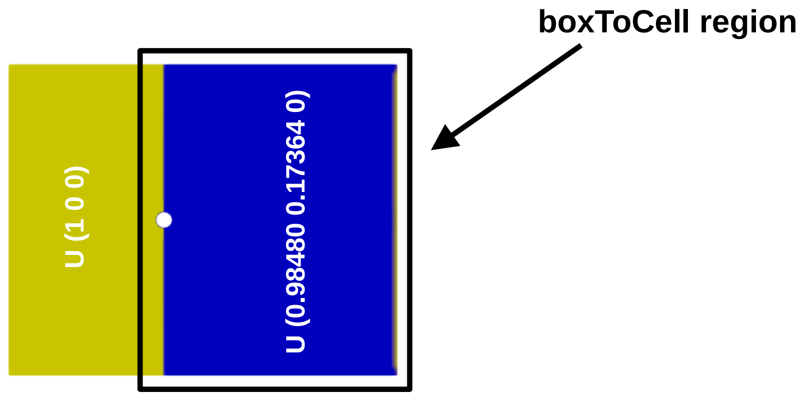
Flow past a cylinder – From laminar to turbulent flow

- Let us run the same case but using a non-uniform field

The `setFieldsDict` dictionary

```
17    defaultFieldValues
18    (
19        volVectorFieldValue U (1 0 0)
20    );
21
22    regions
23    (
24        boxToCell
25        {
26            box (0 -100 -100) (100 100 100);
27            fieldValues
28            (
29                volVectorFieldValue U (0.98480 0.17364 0)
30            );
31        }
32    );
```

- This dictionary file is located in the directory **system**.
- In lines 17-20 we set the default value of the velocity vector to be **(0 0 0)** in the whole domain.
- In lines 24-31, we initialize a rectangular region (**box**) just behind the cylinder with a velocity vector equal to **(0.98480 0.17364 0)**
- In this case, `setFields` will look for the dictionary file *U* and it will overwrite the original values according to the regions defined in `setFieldsDict`.



Flow past a cylinder – From laminar to turbulent flow

- Let us run the same case but using a non-uniform field.
- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c3`**
- Feel free to use the Fluent mesh or the mesh generated with `blockMesh`. Hereafter, we will use `blockMesh`.
- To run this case, in the terminal window type:

1.
```
$> foamCleanTutorials
```
2.
```
$> blockMesh
```
3.
```
$> rm -rf 0 > /dev/null 2>&1
```
4.
```
$> cp -r 0_org/ 0
```
5.
```
$> setFields
```
6.
```
$> renumberMesh -overwrite
```
7.
```
$> pisoFoam | tee log.solver
```
8.
```
$> pyFoamPlotWatcher.py log.pisofoam
```
You will need to launch this script in a different terminal

9.
```
$> gnuplot scripts0/plot_coeffs
```
You will need to launch this script in a different terminal

10.
```
$> paraFoam
```

Flow past a cylinder – From laminar to turbulent flow

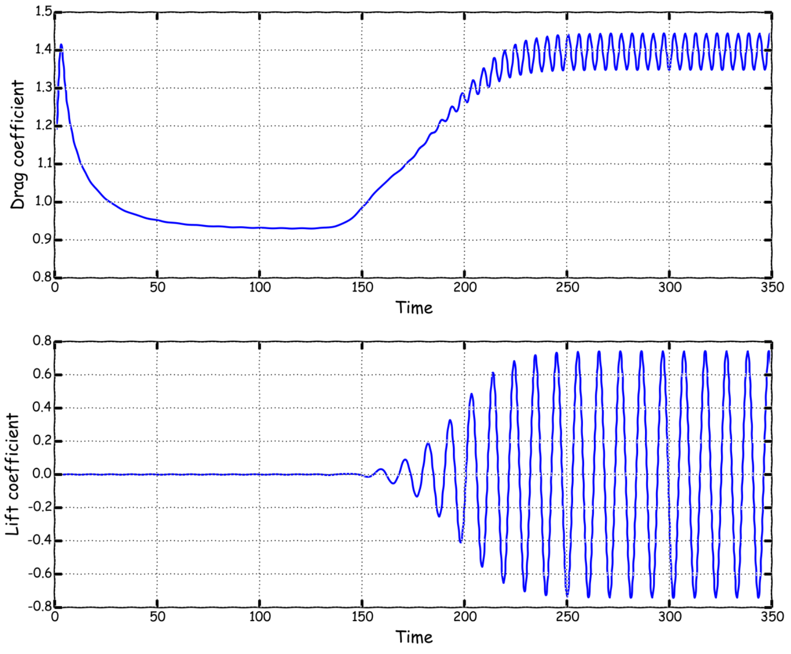
Running the case – Non-uniform field initialization

- In step 2 we generate the mesh using `blockMesh`. The **name** and **type** of the patches are already set in the dictionary `blockMeshDict` so there is no need to modify the `boundary` file.
- In step 4 we copy the original files to the directory **`0`**. We do this to keep a backup of the original files as the file *`0/U`* will be overwritten when using `setFields`.
- In step 5 we initialize the solution using `setFields`.
- In step 6 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.
- In step 7 we run the simulation and save the log file. Notice that we are sending the job to background.
- In step 8 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
- In step 9 we use the gnuplot script `scripts0/plot_coeffs` to plot the lift and drag coefficients on-the-fly. Besides monitoring the residuals, is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.

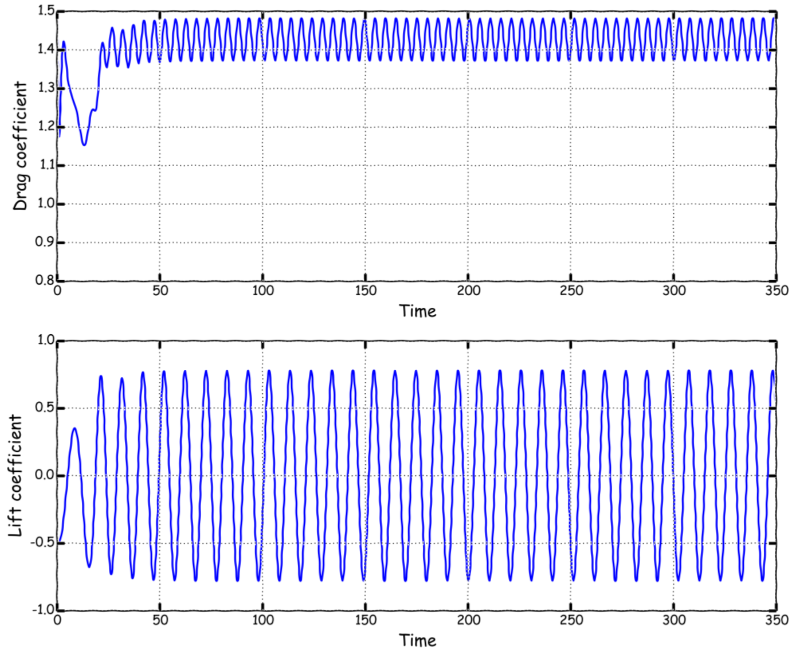
Flow past a cylinder – From laminar to turbulent flow

Does non-uniform field initialization make a difference?

- A picture is worth a thousand words. No need to tell you yes, even if the solutions are slightly different.
- This bring us to the next subject, for how long should we run the simulation?



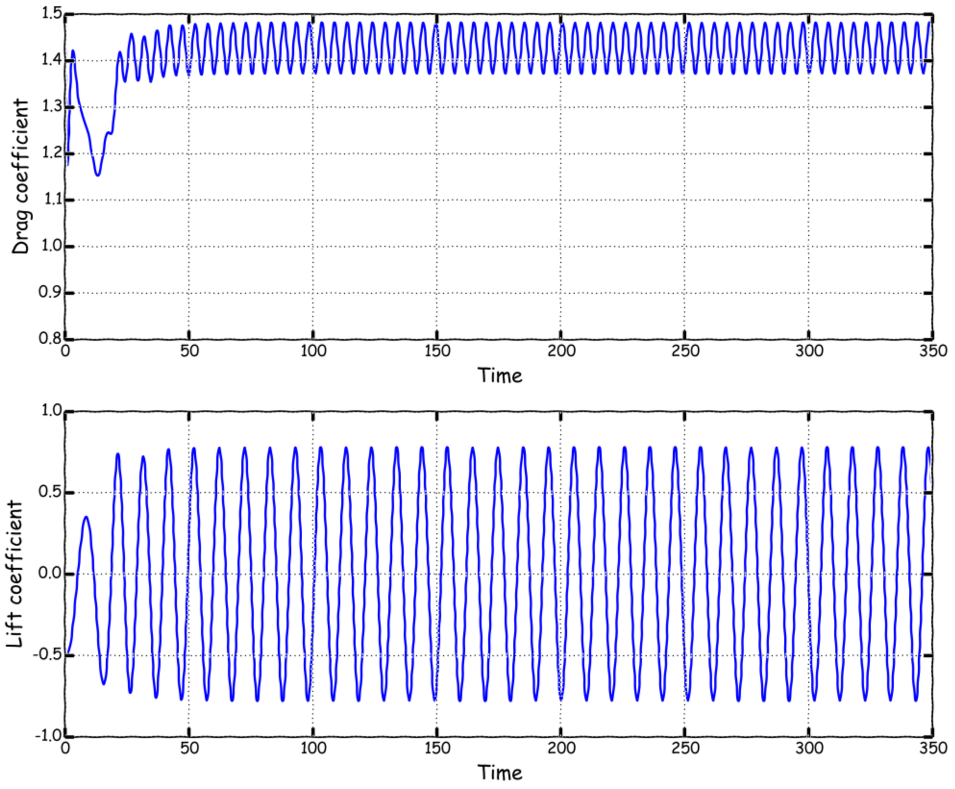
No field initialization



With field initialization

Flow past a cylinder – From laminar to turbulent flow

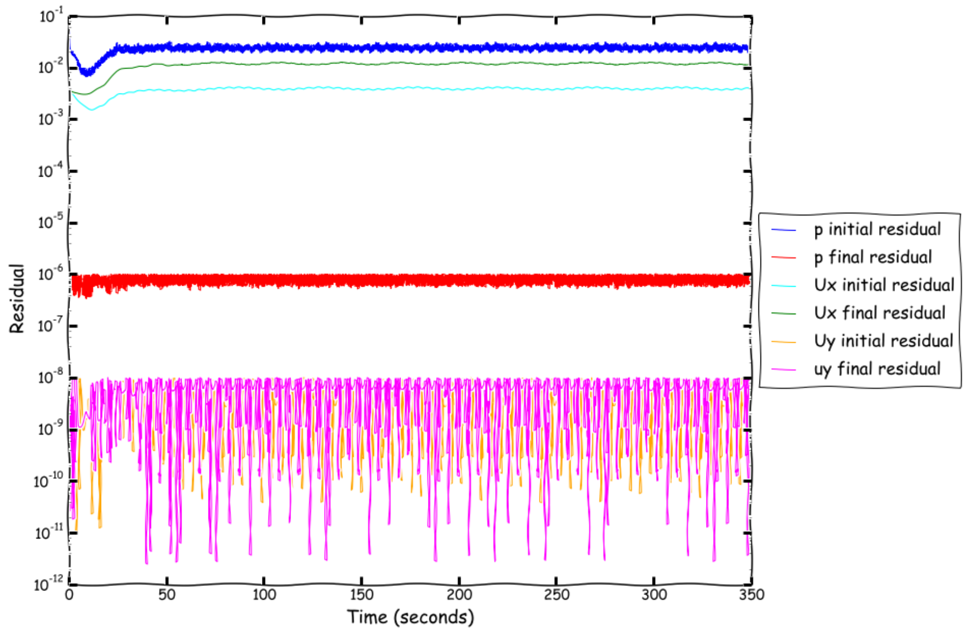
For how long should run the simulation?



- This is the difficult part when dealing with unsteady flows.
- Usually, you run the simulation until the behavior of a quantity of interest does not oscillates or it becomes periodic.
- In this case we can say that after the 50 seconds mark the solution becomes periodic, therefore there is no need to run up to 350 seconds (unless you want to gather a lot of statistics).
- We can stop the simulation at 150 seconds (or maybe less), and do the average of the quantities between 100 and 150 seconds.

Flow past a cylinder – From laminar to turbulent flow

What about the residuals?



- Residuals are telling you a lot, but they are difficult to interpret.
- In this case the fact that the initial residuals are increasing after about 10 seconds, does not mean that the solution is diverging. This is in indication that something is happening (in this case the onset of the instability).
- Remember, the residuals should always drop to the tolerance criteria set in the *fvSolution* dictionary (final residuals). If they do not drop to the desired tolerance, we are talking about unconverged time-steps.
- Things that are not clear from the residuals:
  - For how long should we run the simulation?
  - Is the solution converging to the right value?

Flow past a cylinder – From laminar to turbulent flow

How to compute force coefficients

```
51      functions
52      {

178         forceCoeffs_object
179         {
188             type forceCoeffs;
189             functionObjectLibs ("libforces.so");
191             patches (cylinder);

193             pName p;
194             Uname U;
195             rhoName rhoInf;
196             rhoInf 1.0;

198             //// Dump to file
199             log true;

201             CofR (0.0 0 0);
202             liftDir (0 1 0);
202             dragDir (1 0 0);
204             pitchAxis (0 0 1);
205             magUInf 1.0;
206             lRef 1.0;
207             Aref 2.0;

209             outputControl   timeStep;
210             outputInterval  1;
211         }

237     };
```

- To compute the force coefficients, we use **functionObjects**.
- Remember, **functionObjects** are defined at the end of the `controlDict` dictionary file.
- In line 178 we give a name to the **functionObject**.
- In line 191 we define the patch where we want to compute the forces.
- In lines 195-196 we define the reference density value.
- In line 201 we define the center of rotation (for moments).
- In line 202 we define the lift force axis.
- In line 203 we define the drag force axis.
- In line 204 we define the axis of rotation for moment computation.
- In line 206 we give the reference length (for computing the moments)
- In line 207 we give the reference area (in this case the frontal area).
- The output of this **functionObject** is saved in the file `forceCoeffs.dat` located in the directory **`forceCoeffs_object/0/`**

Flow past a cylinder – From laminar to turbulent flow

Can we compute basic statistics of the force coefficients using gnuplot?

- Yes, we can. Enter the gnuplot prompt and type:

1.
```
gnuplot> stats ‘postProcessing/forceCoeffs_object/0/forceCoeffs.dat’ u 3
```
This will compute the basic statistics of all the rows in the file forceCoeffs.dat (we are sampling column 3 in the input file)

2.
```
gnuplot> stats ‘postProcessing/forceCoeffs_object/0/forceCoeffs.dat’ every ::3000::7000 u 3
```
This will compute the basic statistics of rows 3000 to 7000 in the file forceCoeffs.dat (we are sampling column 3 in the input file)

3.
```
gnuplot> plot ‘postProcessing/forceCoeffs_object/0/forceCoeffs.dat’ u 3 w l
```
This will plot column 3 against the row number (iteration number)

4.
```
gnuplot> exit
```
To exit gnuplot

- Remember the force coefficients information is saved in the file `forceCoeffs.dat` located in the directory **`postProcessing/forceCoeffs_object/0`**

Flow past a cylinder – From laminar to turbulent flow

On the solution accuracy

```
ddtSchemes
{
  default          backward;
}

gradSchemes
{
    default            cellLimited leastSquares 1;
}

divSchemes
{
    default          none;
    div(phi,U)       Gauss linearUpwindV default;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default          Gauss linear limited 1;
}

interpolationSchemes
{
    default          linear;
}

snGradSchemes
{
    default          limited 1;
}
```

- At the end of the day, we want a solution that is second order accurate.
- We define the discretization schemes (and therefore the accuracy) in the dictionary `fvSchemes`.
- In this case, for time discretization (**ddtSchemes**) we are using the **backward** method.
- For gradient discretization (**gradSchemes**) we are using the **leastSquares** method with slope limiters (**cellLimited**) for all terms (**default** option).
- Sometimes adding a gradient limiter to the pressure gradient or **grad(p)** can be too diffusive, so it is better not to use gradient limiters for **grad(p)**, e.g., **grad(p) leastSquares**.
- For the discretization of the convective terms (**divSchemes**) we are using **linearUpwindV** interpolation method for the term **div(rho,U)**.
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear limited 1** method
- In overall, this method is second order accurate (this is what we want).

Flow past a cylinder – From laminar to turbulent flow

On the solution tolerance and linear solvers

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration on;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 100;
        mergeLevels     1;
    }

    pFinal
    {
        $p;
        relTol         0;
    }

    U
    {
        solver         PBiCGStab;
        preconditioner DILU;
        tolerance      1e-08;
        relTol         0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 2;
}
```

- We define the solution tolerance and linear solvers in the dictionary *fvSolution*.
- To solve the pressure (**p**) we are using the **GAMG** method with an absolute **tolerance** of 1e-6 and a relative tolerance **relTol** of 0.01.
- The entry **pFinal** refers to the final correction of the **PISO** loop. It is possible to use a tighter convergence criteria only in the last iteration.
- To solve **U,** we are using the solver **PBiCGStab** and the **DILU** preconditioner**,** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0 (the solver will stop iterating when it meets any of the conditions).
- Solving for the velocity is relatively inexpensive, whereas solving for the pressure is expensive.
- The **PISO** sub-dictionary contains entries related to the pressure-velocity coupling (in this case the **PISO** method). Hereafter we are doing two **PISO** correctors (**nCorrectors**) and two non-orthogonal corrections (**nNonOrthogonalCorrectors**).

Flow past a cylinder – From laminar to turbulent flow

On the runtime parameters

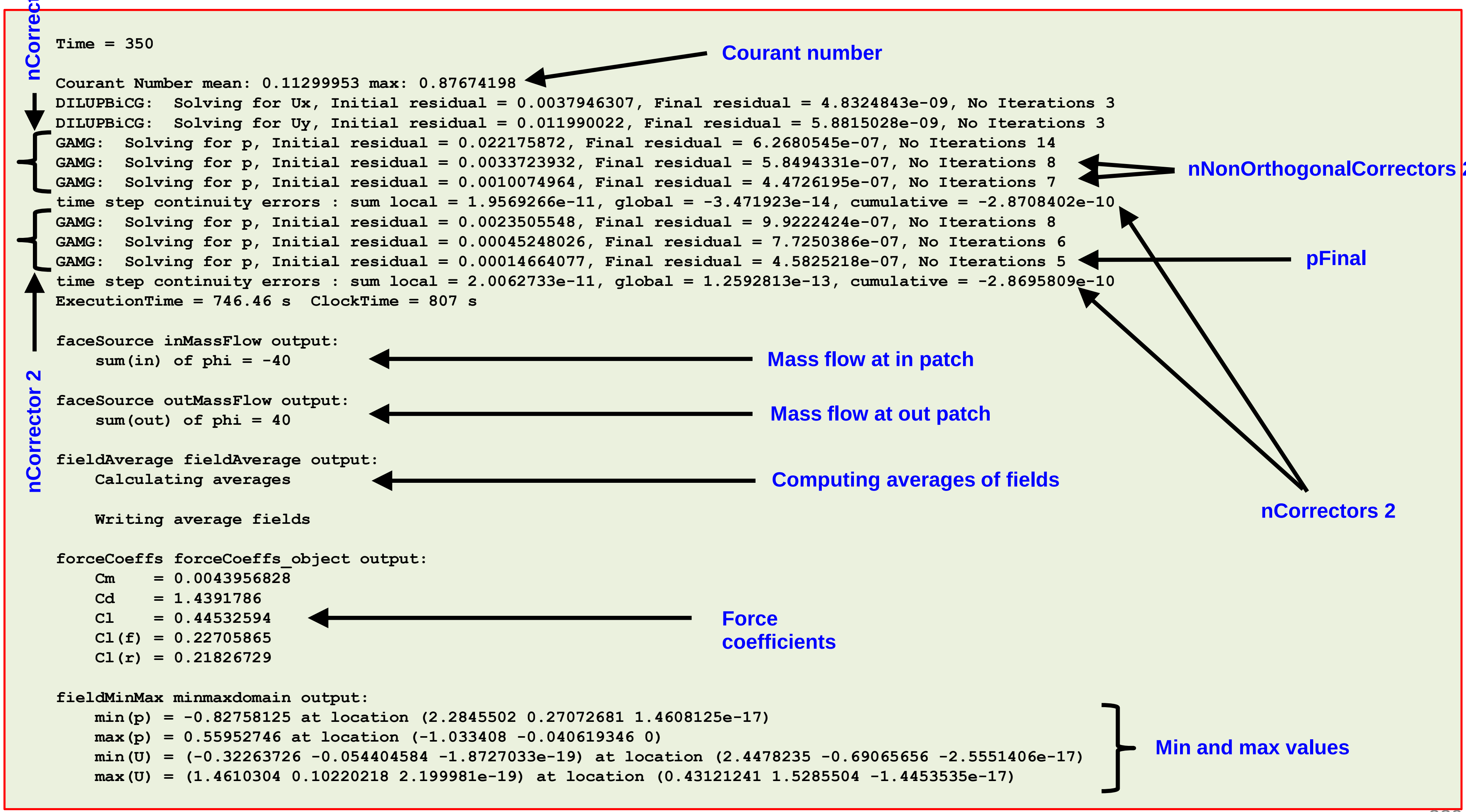
```
application     pisoFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         350;

deltaT          0.05;

writeControl    runTime;

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

- This case starts from the latest saved solution (**startFrom**).
- In this case as there are no saved solutions, it will start from 0 (**startTime**).
- It will run up to 350 seconds (**endTime**).
- The time-step of the simulation is 0.05 seconds (**deltaT**). The time-step has been chosen in such a way that the Courant number is less than 1
- It will write the solution every 1 second (**writeInterval**) of simulation time (**runTime**).
- It will keep all the solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**).
- The write precision is 8 digits (**writePrecision**).
- And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.

Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the `pisoFoam` solver.



```
Time = 350

Courant Number mean: 0.11299953 max: 0.87674198
DILUPBiCG:  Solving for Ux, Initial residual = 0.0037946307, Final residual = 4.8324843e-09, No Iterations 3
DILUPBiCG:  Solving for Uy, Initial residual = 0.011990022, Final residual = 5.8815028e-09, No Iterations 3
GAMG:  Solving for p, Initial residual = 0.022175872, Final residual = 6.2680545e-07, No Iterations 14
GAMG:  Solving for p, Initial residual = 0.0033723932, Final residual = 5.8494331e-07, No Iterations 8
GAMG:  Solving for p, Initial residual = 0.0010074964, Final residual = 4.4726195e-07, No Iterations 7
time step continuity errors : sum local = 1.9569266e-11, global = -3.471923e-14, cumulative = -2.8708402e-10
GAMG:  Solving for p, Initial residual = 0.0023505548, Final residual = 9.9222424e-07, No Iterations 8
GAMG:  Solving for p, Initial residual = 0.00045248026, Final residual = 7.7250386e-07, No Iterations 6
GAMG:  Solving for p, Initial residual = 0.00014664077, Final residual = 4.5825218e-07, No Iterations 5
time step continuity errors : sum local = 2.0062733e-11, global = 1.2592813e-13, cumulative = -2.8695809e-10
ExecutionTime = 746.46 s  ClockTime = 807 s

faceSource inMassFlow output:
    sum(in) of phi = -40

faceSource outMassFlow output:
    sum(out) of phi = 40

fieldAverage fieldAverage output:
    Calculating averages

    Writing average fields

forceCoeffs forceCoeffs_object output:
    Cm    = 0.0043956828
    Cd    = 1.4391786
    Cl    = 0.44532594
    Cl(f) = 0.22705865
    Cl(r) = 0.21826729

fieldMinMax minmaxdomain output:
    min(p) = -0.82758125 at location (2.2845502 0.27072681 1.4608125e-17)
    max(p) = 0.55952746 at location (-1.033408 -0.040619346 0)
    min(U) = (-0.32263726 -0.054404584 -1.8727033e-19) at location (2.4478235 -0.69065656 -2.5551406e-17)
    max(U) = (1.4610304 0.10220218 2.199981e-19) at location (0.43121241 1.5285504 -1.4453535e-17)
```

Flow past a cylinder – From laminar to turbulent flow

Let us use a potential solver to find a quick solution

- In this case we are going to use the potential solver `potentialFoam` (remember potential solvers are inviscid, irrotational and incompressible).
- This solver is super fast, and it can be used to find a solution to be used as initial conditions (non-uniform field) for an incompressible solver.
- A good initial condition will accelerate and improve the convergence rate.
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c4
```

- Do not forget to explore the dictionary files.
- The following dictionaries are different
    - *system/fvSchemes*
    - *system/fvSolution*

Try to spot the differences.

Flow past a cylinder – From laminar to turbulent flow

Running the case – Let us use a potential solver to find a quick solution

- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c4`**
- Feel free to use the Fluent mesh or the mesh generated with `blockMesh`. In this case we will use `blockMesh`.
- To run this case, in the terminal window type:

1. `$> foamCleanTutorials`
2. `$> blockMesh`
3. `$> rm -rf 0`
4. `$> cp -r 0_org 0`
5. `$> potentialFoam -noFunctionObjects -initialiseUBCs -writep -writePhi`
6. `$> paraFoam`

Flow past a cylinder – From laminar to turbulent flow

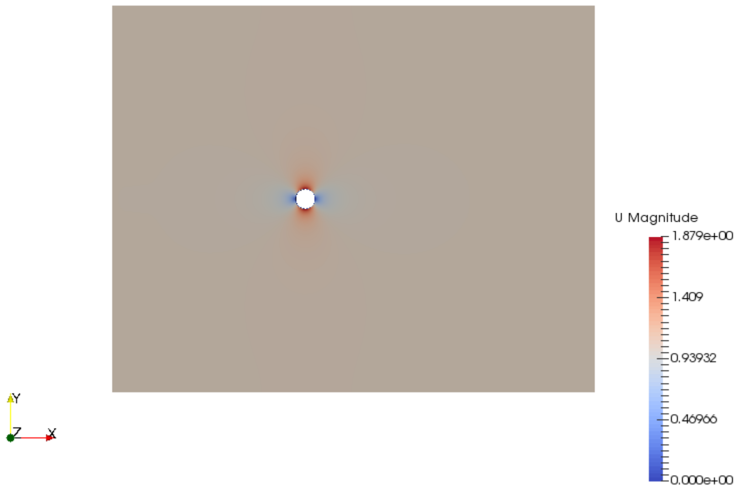
Running the case – Let us use a potential solver to find a quick solution

- In step 2 we generate the mesh using `blockMesh`. The **name** and **type** of the patches are already set in the dictionary `blockMeshDict` so there is no need to modify the *boundary* file.
- In step 4 we copy the original files to the directory **0**. We do this to keep a backup of the original files as they will be overwritten by the solver `potentialFoam`.
- In step 5 we run the solver. We use the option `–noFunctionObjects` to avoid conflicts with the **functionobjects**. The options `–writep` and `–writePhi` will write the pressure field and fluxes respectively.
- At this point, if you want to use this solution as initial conditions for an incompressible solver, just copy the files *U* and *p* into the start directory of the incompressible case you are looking to run. Have in mind that the meshes need to be the same.
- Be careful with the **name** and **type** of the boundary conditions, they should be same between the potential case and incompressible case.

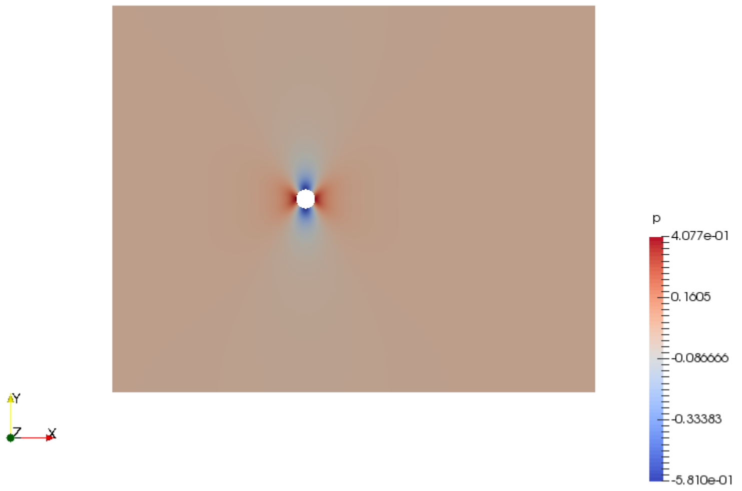
Flow past a cylinder – From laminar to turbulent flow

Potential solution

- Using a potential solution as initial conditions is much better than using a uniform flow. It will speed up the solution and it will give you more stability.
- Finding a solution using the potential solver is inexpensive.



Velocity field



Pressure field

Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the `potentialFoam` solver.
- The output of this solver is also a good indication of the sensitivity of the mesh quality to gradients computation. If you see that the number of iterations are dropping iteration after iteration, it means that the mesh is fine.
- If the number of iterations remain stalled, it means that the mesh is sensitive to gradients, so you should use non-orthogonal correction.
- In this case we have a good mesh.

```
Calculating potential flow
DICPCG:  Solving for Phi, Initial residual = 2.6622265e-05, Final residual = 8.4894837e-07, No Iterations 27
DICPCG:  Solving for Phi, Initial residual = 1.016986e-05, Final residual = 9.5168103e-07, No Iterations 9
DICPCG:  Solving for Phi, Initial residual = 4.0789046e-06, Final residual = 7.7788216e-07, No Iterations 5
DICPCG:  Solving for Phi, Initial residual = 1.8251249e-06, Final residual = 8.8483568e-07, No Iterations 1
DICPCG:  Solving for Phi, Initial residual = 1.1220074e-06, Final residual = 5.6696809e-07, No Iterations 1
DICPCG:  Solving for Phi, Initial residual = 7.1187246e-07, Final residual = 7.1187246e-07, No Iterations 0
Continuity error = 1.3827583e-06
Interpolated velocity error = 7.620206e-07

Calculating approximate pressure field
DICPCG:  Solving for p, Initial residual = 0.0036907012, Final residual = 9.7025397e-07, No Iterations 89
DICPCG:  Solving for p, Initial residual = 0.0007470416, Final residual = 9.9942495e-07, No Iterations 85
DICPCG:  Solving for p, Initial residual = 0.00022829496, Final residual = 8.6107759e-07, No Iterations 36
DICPCG:  Solving for p, Initial residual = 7.9622793e-05, Final residual = 8.4360883e-07, No Iterations 31
DICPCG:  Solving for p, Initial residual = 2.8883108e-05, Final residual = 8.7152873e-07, No Iterations 25
DICPCG:  Solving for p, Initial residual = 1.151539e-05, Final residual = 9.7057871e-07, No Iterations 9
ExecutionTime = 0.17 s  ClockTime = 0 s

End
```

Velocity computation

Initial approximation

nNonOrthogonalCorrectors 5

Pressure computation

Flow past a cylinder – From laminar to turbulent flow

Let us map a solution from a coarse mesh to a finer mesh

- It is also possible to map the solution from a coarse mesh to a finer mesh (and all the way around).
- For instance, you can compute a full Navier-Stokes solution in a coarse mesh (fast solution), and then map it to a finer mesh.
- Let us map the solution from the potential solver to a finer mesh (if you want you can map the solution obtained using `pisoFoam` or `icoFoam`). To do this we will use the utility `mapFields`.
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c6
```

Flow past a cylinder – From laminar to turbulent flow

Running the case – Let us map a solution from a coarse mesh to a finer mesh

- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c6`**
- To generate the mesh, use `blockMesh` (remember this mesh is finer).
- To run this case, in the terminal window type:

```
1.  $> foamCleanTutorials
2.  $> blockMesh
3.  $> rm -rf 0
4.  $> cp -r 0_org 0
5.  $> mapfields ../c4 -consistent -noFunctionObjects -mapMethod cellPointInterpolate -sourceTime 0
6.  $> paraFoam
```

- To run step 5 you need to have a solution in the directory **`../c4`**

Flow past a cylinder – From laminar to turbulent flow

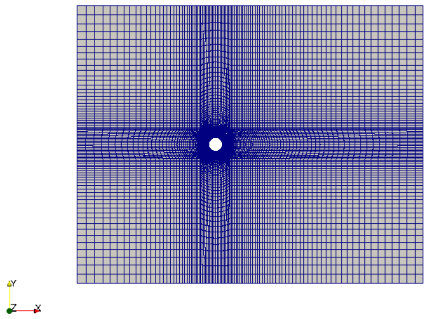
Running the case – Let us map a solution from a coarse mesh to a finer mesh

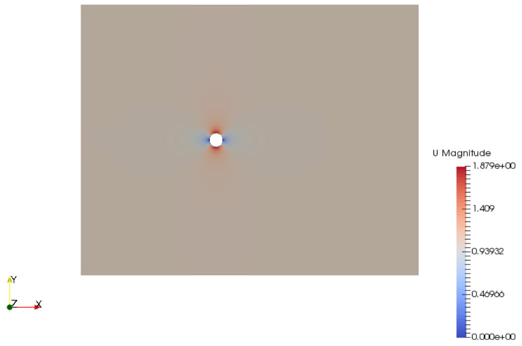
- In step 2 we generate a finer mesh using `blockMesh`. The **name** and **type** of the patches are already set in the dictionary `blockMeshDict` so there is no need to modify the `boundary` file.
- In step 4 we copy the original files to the directory **`0`**. We do this to keep a backup of the original files as they will be overwritten by the utility `mapFields`.
- In step 5 we use the utility `mapFields` with the following options:
  - We copy the solution from the directory `../c4`
  - The options `–consistent` is used when the domains and BCs are the same.
  - The option `–noFunctionObjects` is used to avoid conflicts with the **functionObjects**.
  - The option `–mapMethod cellPointInterpolate` defines the interpolation method.
  - The option `–sourceTime 0` defines the time from which we want to interpolate the solution.

Flow past a cylinder – From laminar to turbulent flow

The meshes and the mapped fields

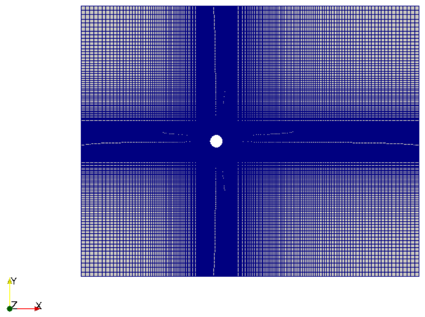
Coarse mesh

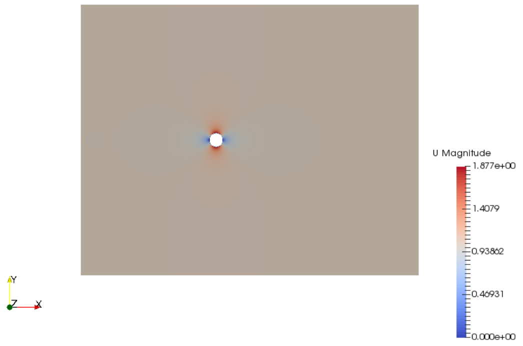




$\xrightarrow{\texttt{mapFields}}$

Fine mesh

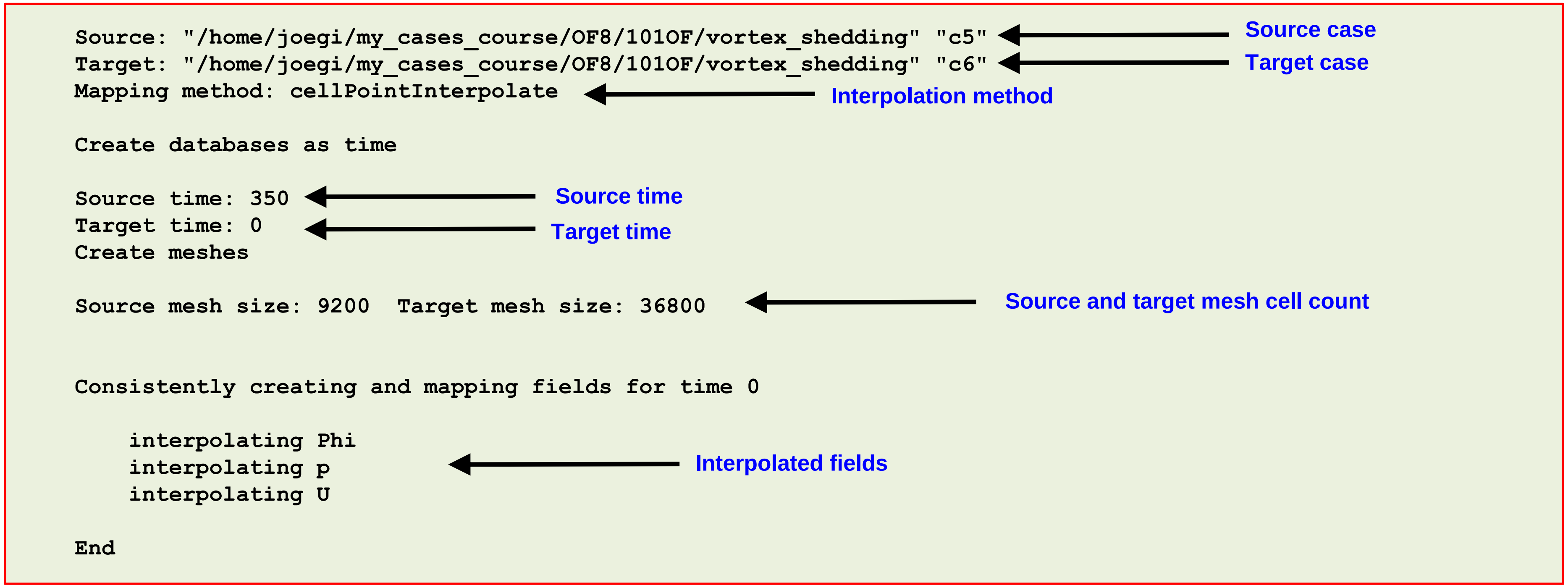




Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the `mapFields` utility.
- The utility `mapFields`, will try to interpolate all fields in the source directory.
- You can control the target time via the **startFrom** and **startTime** keywords in the *`controlDict`* dictionary file.



```
Source: "/home/joegi/my_cases_course/OF8/101OF/vortex_shedding" "c5"
Target: "/home/joegi/my_cases_course/OF8/101OF/vortex_shedding" "c6"
Mapping method: cellPointInterpolate

Create databases as time

Source time: 350
Target time: 0
Create meshes

Source mesh size: 9200  Target mesh size: 36800


Consistently creating and mapping fields for time 0

    interpolating Phi
    interpolating p
    interpolating U

End
```

- Finally, after mapping the solution, you can run the solver in the usual way. The solver will use the mapped solution as initial conditions.

Flow past a cylinder – From laminar to turbulent flow

Setting a turbulent case

- So far, we have used laminar incompressible solvers.
- Let us do a turbulent simulation.
- When doing turbulent simulations, we need to choose the turbulence model, define the boundary and initial conditions for the turbulent quantities, and modify the `fvSchemes` and `fvSolution` dictionaries to take account for the new variables we are solving (the transported turbulent quantities).
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c14
```

Flow past a cylinder – From laminar to turbulent flow

- The following dictionaries remain unchanged
  - *system/blockMeshDict*
  - *constant/polyMesh/boundary*
  - *0/p*
  - *0/U*

- The following dictionaries need to be adapted for the turbulence case
  - *constant/transportProperties*
  - *system/controlDict*
  - *system/fvSchemes*
  - *system/fvSolution*

- The following dictionaries need to be adapted for the turbulence case
  - *constant/momentumTransport*

Flow past a cylinder – From laminar to turbulent flow

The *transportProperties* dictionary file

- This dictionary file is located in the directory **constant**.
- In this file we set the transport model and the kinematic viscosity (**nu**).

```
transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 0.0002;
```

- Reminder:
  - The diameter of the cylinder is 2.0 m.
  - And we are targeting for a Re = 10000.

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

Flow past a cylinder – From laminar to turbulent flow

The *momentumTransport* dictionary file

- This dictionary file is located in the directory **`constant`**.
- In this dictionary file we select what model we would like to use (laminar or turbulent).
- In this case we are interested in modeling turbulence, therefore the dictionary is as follows

```
simulationType  RAS;                 RANS type simulation

RAS                                  RANS sub-dictionary
{
    RASModel        kOmegaSST;       RANS model to use

    turbulence      on;              Turn on/off turbulence.  Runtime modifiable

    printCoeffs     on;              Print coefficients at the beginning
}
```

- If you want to know the models available use the banana method.

Flow past a cylinder – From laminar to turbulent flow

The `controlDict` dictionary

```
17      application       pimpleFoam;
18
20      startFrom         latestTime;
21
22      startTime         0;
23
24      stopAt            endTime;
25
26      endTime           500;
27
28      deltaT            0.001;
29
30      writeControl      runTime;
31
32      writeInterval     1;
33
34      purgeWrite        0;
35
36      writeFormat       ascii;
37
38      writePrecision    8;
39
40      writeCompression  off;
41
42      timeFormat        general;
43
44      timePrecision     6;
45
46      runTimeModifiable yes;
47
48      adjustTimeStep    yes;
49
50      maxCo             0.9;
51      maxDeltaT         0.1;
```

- This case will start from the last saved solution (**startFrom**). If there is no solution, the case will start from time 0 (**startTime**).
- It will run up to 500 seconds (**endTime**).
- The initial time-step of the simulation is 0.001 seconds (**deltaT**).
- It will write the solution every 1 second (**writeInterval**) of simulation time (**runTime**).
- It will keep all the solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**).
- The write precision is 8 digits (**writePrecision**).
- And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.
- In line 48 we turn on the option **adjustTimeStep**. This option will automatically adjust the time-step to achieve the maximum desired courant number **maxCo** (line 50).
- We also set a maximum time-step **maxDeltaT** in line 51.
- Remember, the first time-step of the simulation is done using the value set in line 28 and then it is automatically scaled to achieve the desired maximum values (lines 50-51).
- The feature **adjustTimeStep** is only present in the **PIMPLE** family solvers, but it can be added to any solver by modifying the source code.

Flow past a cylinder – From laminar to turbulent flow

The *fvSchemes* dictionary

```
ddtSchemes
{
    default         CrankNicolson 0.7;
}
gradSchemes
{
    default           cellLimited leastSquares 1;
    grad(U)           cellLimited Gauss linear 1;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwindV grad(U);
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div(phi,k)          Gauss linearUpwind default;
    div(phi,omega)      Gauss linearUpwind default;
}
laplacianSchemes
{
     default         Gauss linear limited 1;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         limited 1;
}
wallDist
{
    method meshWave;
}
```

- In this case, for time discretization (**ddtSchemes**) we are using the blended **CrankNicolson** method. The blending coefficient goes from 0 to 1, where 0 is equivalent to the **Euler** method and 1 is a pure **Crank Nicolson**.
- For gradient discretization (**gradSchemes**) we are using as default option the **leastSquares** method. For **grad(U)** we are using **Gauss linear** with slope limiters (**cellLimited**). You can define different methods for every term in the governing equations, for example, you can define a different method for **grad(p)**.
- For the discretization of the convective terms (**divSchemes**) we are using **linearUpwindV** interpolation method with slope limiters for the term **div(phi,U)**.
- For the terms **div(phi,k)** and **div(phi,omega)** we are using **linearUpwind** interpolation method with no slope limiters. These terms are related to the turbulence modeling.
- For the term **div((nuEff*dev2(T(grad(U)))))** we are using **linear** interpolation (this term is related to turbulence modeling).
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear limited 1** method.
- To compute the distance to the wall and normals to the wall, we use the method **meshWave**. This only applies when using wall functions (turbulence modeling).
- This method is second order accurate.

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
    solvers
    {
        p
        {
            solver           GAMG;
            tolerance        1e-6;
            relTol           0.001;
            smoother         GaussSeidel;
            nPreSweeps       0;
            nPostSweeps      2;
            cacheAgglomeration on;
            agglomerator     faceAreaPair;
            nCellsInCoarsestLevel 100;
            mergeLevels      1;
            minIter         2;
        }

        pFinal
        {
            solver          PCG;
            preconditioner  DIC;
            tolerance       1e-06;
            relTol          0;
            minIter         3;
        }

        U
        {
            solver          PBiCGStab;
            preconditioner  DILU;
            tolerance       1e-08;
            relTol          0;
            minIter         3;
        }
```

- To solve the pressure (**p**) we are using the **GAMG** method, with an absolute **tolerance** of 1e-6 and a relative tolerance **relTol** of 0.001. Notice that we are fixing the number of minimum iterations (**minIter**).
- To solve the final pressure correction (**pFinal**) we are using the **PCG** method with the **DIC** preconditioner, with an absolute **tolerance** of 1e-6 and a relative tolerance **relTol** of 0.
- Notice that we can use different methods between **p** and **pFinal**. In this case we are using a tighter tolerance for the last iteration.
- We are also fixing the number of minimum iterations (**minIter**). This entry is optional.
- To solve **U** we are using the solver **PBiCGStab** with the **DILU** preconditioner, an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
    solvers
    {

        UFinal
        {
            solver          PBiCGStab;
            preconditioner  DILU;
            tolerance       1e-08;
            relTol          0;
            minIter          3;
        }

        omega
        {
            solver          PBiCGStab;
            preconditioner  DILU;
            tolerance       1e-08;
            relTol          0;
            minIter          3;
        }

        omegaFinal
        {
            solver          PBiCGStab;
            preconditioner  DILU;
            tolerance       1e-08;
            relTol          0;
            minIter          3;
        }

        k
        {
            solver          PBiCGStab;
            preconditioner  DILU;
            tolerance       1e-08;
            relTol          0;
            minIter          3;
        }
```

- To solve **UFinal** we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).
- To solve **omega** and **omegaFinal** we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).
- To solve **k,** we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
113         kFinal
114         {
115             solver          PBiCGStab;
116             preconditioner  DILU;
117             tolerance       1e-08;
118             relTol          0;
119             minIter          3;
120         }
121     }
122
123     PIMPLE
124     {
125          momentumPreditor   yes;
126          consistent         yes;

130           nOuterCorrectors 1;
132           nCorrectors 3;
133          nNonOrthogonalCorrectors 1;

137     }


157     relaxationFactors
158     {
159         fields
160         {
161             ".*"            0.9;
162         }
163         equations
164         {
165             ".*"            0.9;
166         }
167     }
```

- To solve **kFinal** we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0. Notice that we are fixing the number of minimum iterations (**minIter**).
- In lines 123-137 we setup the entries related to the pressure-velocity coupling method used (**PIMPLE** in this case). Setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- To gain more stability we are using 1 outer correctors (**nOuterCorrectors**), 3 inner correctors or **PISO** correctors (**nCorrectors**), and 1 correction due to non-orthogonality (**nNonOrthogonalCorrectors**).
- Remember, adding corrections increase the computational cost.
- In lines 157-167 we setup the under-relaxation factors used during the outer corrections of the **PIMPLE** method.
    - The values defined correspond to the industry standard of the **SIMPLE** method.
    - By using under-relaxation, we ensure diagonal equality.
    - Be careful not use too low values as you will lose time accuracy.
    - If you want to disable under-relaxation, comment out these lines.

Flow past a cylinder – From laminar to turbulent flow

- The following dictionaries are new
    - *0/k*
    - *0/omega*
    - *0/nut*

These are the field variables related to the closure equations of the turbulent model.

- As we are going to use the $\kappa - \omega \; SST$ model, we need to define the initial conditions and boundaries conditions.
- To define the IC/BC we will use the free stream values of κ and ω
- In the following site, you can find a lot information about choosing initial and boundary conditions for the different turbulence models:
    - https://turbmodels.larc.nasa.gov/

Flow past a cylinder – From laminar to turbulent flow

$\kappa - \omega\ SST$ Turbulence model free-stream boundary conditions

- The initial value for the turbulent kinetic energy κ can be found as follows

$$\kappa = \frac{3}{2}(UI)^2$$

- The initial value for the specific turbulent dissipation rate ω can be found as follows

$$\omega = \frac{\rho\kappa}{\mu}\frac{\mu_t}{\mu}^{-1}$$

- Where $\frac{\mu_t}{\mu}$ is the viscosity ratio and $I = \frac{u'}{\bar{u}}$ is the turbulence intensity.

- If you are working with external aerodynamics or virtual wind tunnels, you can use the following reference values for the turbulence intensity and the viscosity ratio. They work most of the times, but it is a good idea to have some experimental data or a better initial estimate.

	Low	Medium	High
I	1.0 %	5.0 %	10.0 %
μ_t/μ	1	10	100

Flow past a cylinder – From laminar to turbulent flow

The file *0/k*

```
19    internalField   uniform 0.00015;
20
21    boundaryField
22    {
23        out
24        {
25            type            inletOutlet;
26            inletValue      uniform 0.00015;
27            value           uniform 0.00015;
28        }
29        sym1
30        {
31            type            symmetry;
32        }
33        sym2
34        {
35            type            symmetry;
36        }
37        in
38        {
39            type            fixedValue;
40            value           uniform 0.00015;
41        }
42        cylinder
43        {
44            type            kqRWallFunction;
45            value           uniform 0.00015;
46        }
47        back
48        {
49            type            empty;
50        }
51        front
52        {
53            type            empty;
54        }
55    }
```

- We are using uniform initial conditions (line 19).
- For the **in** patch, we are using a **fixedValue** boundary condition.
- For the **out** patch we are using an **inletOutlet** boundary condition (this boundary condition avoids backflow).
- For the **cylinder** patch (which is **base type wall**), we are using the **kqRWallFunction** boundary condition. This is a wall function; we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of **base type wall**.
- The rest of the patches are constrained.
- FYI, the inlet velocity is 1 and the turbulence intensity is equal to 1%.
- We will study with more details how to setup the boundary conditions when we deal with turbulence modeling in the advanced modules.

Flow past a cylinder – From laminar to turbulent flow

The file *0/omega*

```
19    internalField   uniform 0.075;
20
21    boundaryField
22    {
23        out
24        {
25            type            inletOutlet;
26            inletValue      uniform 0.075;
27            value           uniform 0.075;
28        }
29        sym1
30        {
31            type            symmetry;
32        }
33        sym2
34        {
35            type            symmetry;
36        }
37        in
38        {
39            type            fixedValue;
40            value           uniform 0.075;
41        }
42        cylinder
43        {
44            type            omegaWallFunction;
45            Cmu             0.09;
46            kappa           0.41;
47            E               9.8;
48            beta1           0.075;
49            value           uniform 0.075;
50        }
51        back
52        {
53            type            empty;
54        }
55        front
56        {
57            type            empty;
58        }
59    }
```

- We are using uniform initial conditions (line 19).
- For the **in** patch, we are using a **fixedValue** boundary condition.
- For the **out** patch we are using an **inletOutlet** boundary condition (this boundary condition avoids backflow).
- For the **cylinder** patch (which is **base type wall**), we are using the **omegaWallFunction** boundary condition. This is a wall function; we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of **base type wall**.
- The rest of the patches are constrained.
- FYI, the inlet velocity is 1 and the eddy viscosity ratio is equal to 10.
- We will study with more details how to setup the boundary conditions when we deal with turbulence modeling in the advanced modules.

Flow past a cylinder – From laminar to turbulent flow

The file *0/nut*

```
19   internalField   uniform 0;
20
21   boundaryField
22   {
23       out
24       {
25           type            calculated;
26           value           uniform 0;
27       }
28       sym1
29       {
30           type            symmetry;
31       }
32       sym2
33       {
34           type            symmetry;
35       }
36       in
37       {
38           type            calculated;
39           value           uniform 0;
40       }
41       cylinder
42       {
43           type            nutkWallFunction;
44           Cmu             0.09;
45           kappa           0.41;
46           E               9.8;
47           value           uniform 0;
48       }
49       back
50       {
51           type            empty;
52       }
53       front
54       {
55           type            empty;
56       }
57   }
```

- We are using uniform initial conditions (line 19).
- For the **in** patch, we are using the **calculated** boundary condition (nut is computed from kappa and omega)
- For the **out** patch we are using the **calculated** boundary condition (nut is computed from kappa and omega)
- For the **cylinder** patch (which is **base type wall**), we are using the **nutkWallFunction** boundary condition. This is a wall function; we are going to talk about this when we deal with turbulence modeling. Remember, we can use wall functions only if the patch is of **base type wall**.
- The rest of the patches are constrained.
- Remember, the turbulent viscosity ν_t (nut) is equal to

$$\frac{\kappa}{\omega}$$

- We will study with more details how to setup the boundary conditions when we deal with turbulence modeling in the advanced modules.

Flow past a cylinder – From laminar to turbulent flow

Running the case – Setting a turbulent case

- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c14`**
- Feel free to use the Fluent mesh or the mesh generated with `blockMesh`. In this case we will use `blockMesh`.
- To run this case, in the terminal window type:

1.
```
$> foamCleanTutorials
```
2.
```
$> blockMesh
```
3.
```
$> renumberMesh -overwrite
```
4.
```
$> pimpleFoam | tee log.solver
```
   You will need to launch this script in a different terminal
5.
```
$> pyFoamPlotWatcher.py log.solver
```
   You will need to launch this script in a different terminal
6.
```
$> gnuplot scripts0/plot_coeffs
```
   You will need to launch this script in a different terminal
7.
```
$> pimpleFoam -postprocess -func yPlus -latestTime -noFunctionObjects
```
8.
```
$> paraFoam
```

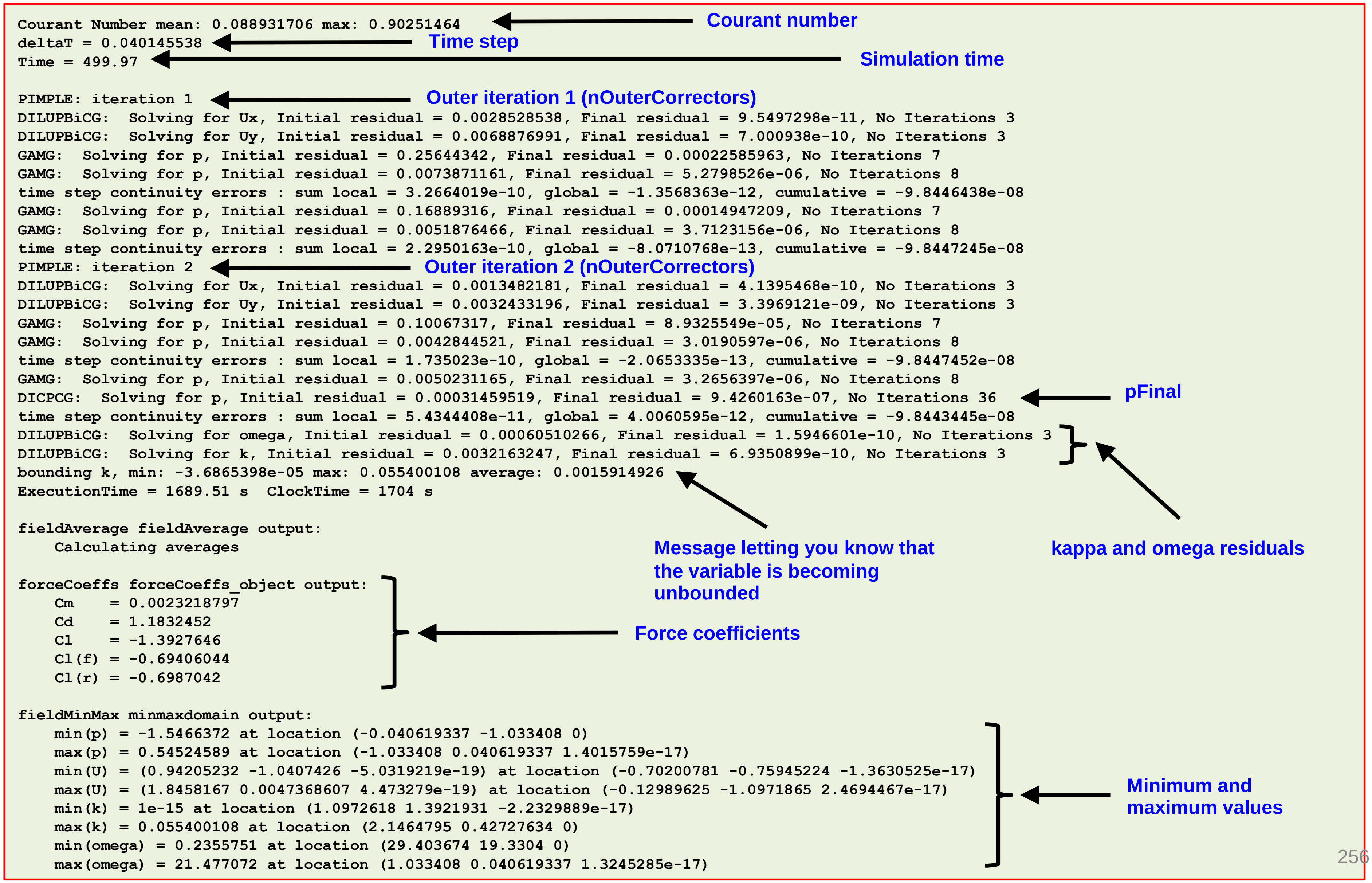
Flow past a cylinder – From laminar to turbulent flow

Running the case – Setting a turbulent case

- In step 3 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.
- In step 4 we run the simulation and save the log file. Notice that we are sending the job to background.
- In step 5 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
- In step 6 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.
- In step 7 we use the utility `postProcess` to compute the y^+ value of each saved solution (we are going to talk about y^+ when we deal with turbulence modeling).

Flow past a cylinder – From laminar to turbulent flow

`pimpleFoam` **output screen**



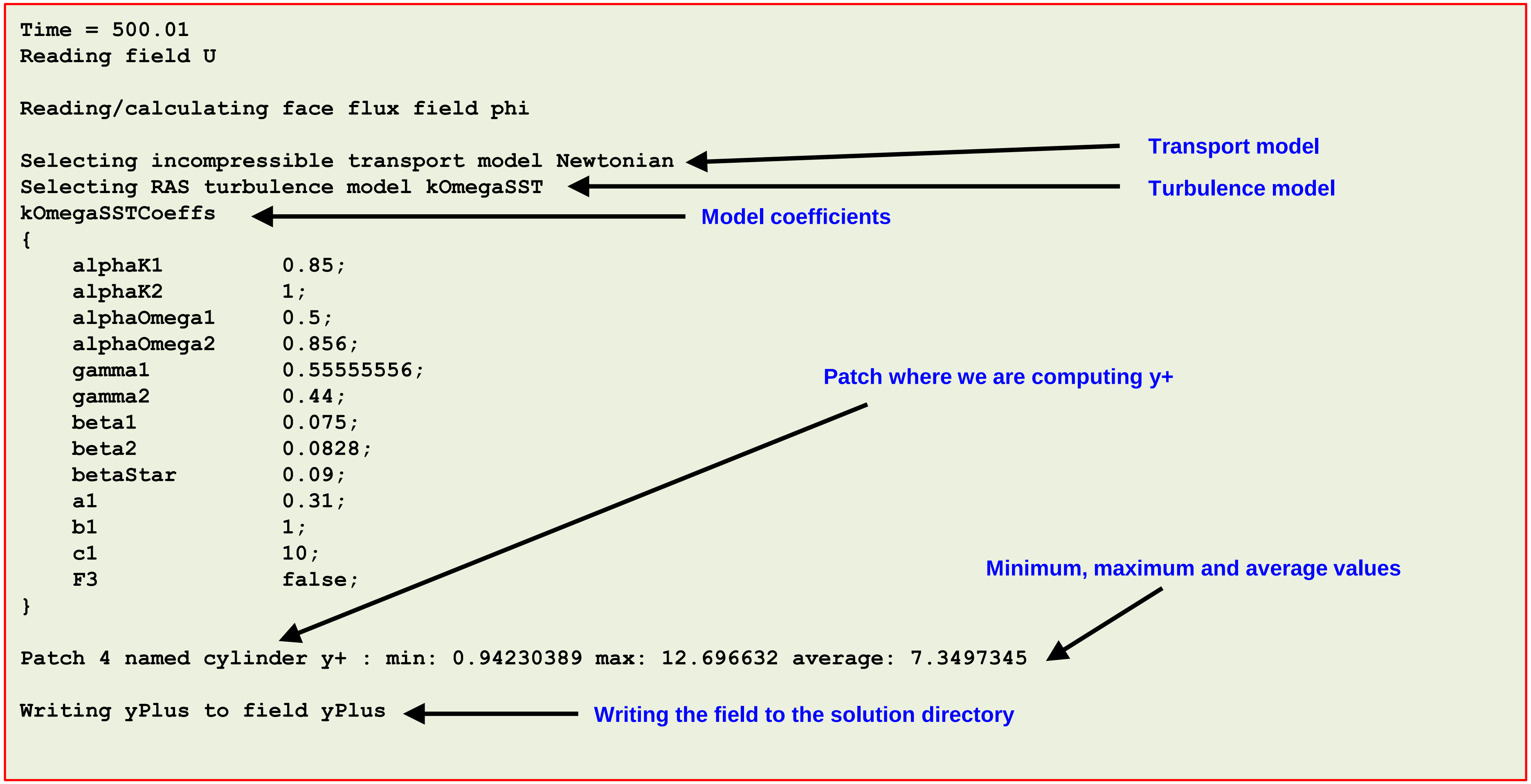
```
Courant Number mean: 0.088931706 max: 0.90251464
deltaT = 0.040145538
Time = 499.97

PIMPLE: iteration 1
DILUPBiCG:  Solving for Ux, Initial residual = 0.0028528538, Final residual = 9.5497298e-11, No Iterations 3
DILUPBiCG:  Solving for Uy, Initial residual = 0.0068876991, Final residual = 7.000938e-10, No Iterations 3
GAMG:  Solving for p, Initial residual = 0.25644342, Final residual = 0.00022585963, No Iterations 7
GAMG:  Solving for p, Initial residual = 0.0073871161, Final residual = 5.2798526e-06, No Iterations 8
time step continuity errors : sum local = 3.2664019e-10, global = -1.3568363e-12, cumulative = -9.8446438e-08
GAMG:  Solving for p, Initial residual = 0.16889316, Final residual = 0.00014947209, No Iterations 7
GAMG:  Solving for p, Initial residual = 0.0051876466, Final residual = 3.7123156e-06, No Iterations 8
time step continuity errors : sum local = 2.2950163e-10, global = -8.0710768e-13, cumulative = -9.8447245e-08
PIMPLE: iteration 2
DILUPBiCG:  Solving for Ux, Initial residual = 0.0013482181, Final residual = 4.1395468e-10, No Iterations 3
DILUPBiCG:  Solving for Uy, Initial residual = 0.0032433196, Final residual = 3.3969121e-09, No Iterations 3
GAMG:  Solving for p, Initial residual = 0.10067317, Final residual = 8.9325549e-05, No Iterations 7
GAMG:  Solving for p, Initial residual = 0.0042844521, Final residual = 3.0190597e-06, No Iterations 8
time step continuity errors : sum local = 1.735023e-10, global = -2.0653335e-13, cumulative = -9.8447452e-08
GAMG:  Solving for p, Initial residual = 0.0050231165, Final residual = 3.2656397e-06, No Iterations 8
DICPCG:  Solving for p, Initial residual = 0.00031459519, Final residual = 9.4260163e-07, No Iterations 36
time step continuity errors : sum local = 5.4344408e-11, global = 4.0060595e-12, cumulative = -9.8443445e-08
DILUPBiCG:  Solving for omega, Initial residual = 0.00060510266, Final residual = 1.5946601e-10, No Iterations 3
DILUPBiCG:  Solving for k, Initial residual = 0.0032163247, Final residual = 6.9350899e-10, No Iterations 3
bounding k, min: -3.6865398e-05 max: 0.055400108 average: 0.0015914926
ExecutionTime = 1689.51 s  ClockTime = 1704 s

fieldAverage fieldAverage output:
    Calculating averages

forceCoeffs forceCoeffs_object output:
    Cm    = 0.0023218797
    Cd    = 1.1832452
    Cl    = -1.3927646
    Cl(f) = -0.69406044
    Cl(r) = -0.6987042

fieldMinMax minmaxdomain output:
    min(p) = -1.5466372 at location (-0.040619337 -1.033408 0)
    max(p) = 0.54524589 at location (-1.033408 0.040619337 1.4015759e-17)
    min(U) = (0.94205232 -1.0407426 -5.0319219e-19) at location (-0.70200781 -0.75945224 -1.3630525e-17)
    max(U) = (1.8458167 0.0047368607 4.473279e-19) at location (-0.12989625 -1.0971865 2.4694467e-17)
    min(k) = 1e-15 at location (1.0972618 1.3921931 -2.2329889e-17)
    max(k) = 0.055400108 at location (2.1464795 0.42727634 0)
    min(omega) = 0.2355751 at location (29.403674 19.3304 0)
    max(omega) = 21.477072 at location (1.033408 0.040619337 1.3245285e-17)
```

Courant number
Time step
Simulation time
Outer iteration 1 (nOuterCorrectors)
Outer iteration 2 (nOuterCorrectors)
pFinal
Message letting you know that the variable is becoming unbounded
kappa and omega residuals
Force coefficients
Minimum and maximum values

Flow past a cylinder – From laminar to turbulent flow

The output screen

- This is the output screen of the `yPlus` utility.



```
Time = 500.01
Reading field U

Reading/calculating face flux field phi

Selecting incompressible transport model Newtonian
Selecting RAS turbulence model kOmegaSST
kOmegaSSTCoeffs
{
    alphaK1         0.85;
    alphaK2         1;
    alphaOmega1     0.5;
    alphaOmega2     0.856;
    gamma1          0.55555556;
    gamma2          0.44;
    beta1           0.075;
    beta2           0.0828;
    betaStar        0.09;
    a1              0.31;
    b1              1;
    c1              10;
    F3              false;
}

Patch 4 named cylinder y+ : min: 0.94230389 max: 12.696632 average: 7.3497345

Writing yPlus to field yPlus
```

Transport model
Turbulence model
Model coefficients
Patch where we are computing y+
Minimum, maximum and average values
Writing the field to the solution directory

Flow past a cylinder – From laminar to turbulent flow

Using a compressible solver

- So far, we have only used incompressible solvers.
- Let us use the compressible solver `rhoPimpleFoam`, which is a,

  Transient solver for laminar or turbulent flow of compressible fluids for HVAC and similar applications. Uses the flexible PIMPLE (PISO-SIMPLE) solution for time-resolved and pseudo-transient simulations.

- When working with compressible solver we need to define the thermodynamical properties of the working fluid and the temperature field (we are also solving the energy equation).
- Also remember, compressible solvers use absolute pressure. Conversely, incompressible solvers use relative pressure.
- This case is already setup in the directory

```
$PTOFC/101OF/vortex_shedding/c24
```

Flow past a cylinder – From laminar to turbulent flow

- The following dictionaries remain unchanged
    - *`system/blockMeshDict`*
    - *`constant/polyMesh/boundary`*

- Reminder:
    - The diameter of the cylinder is 0.002 m.
    - The working fluid is air at 20° Celsius and at a sea level.
    - And we are targeting for a Re = 20000.

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

Flow past a cylinder – From laminar to turbulent flow

The **`constant`** directory

- In this directory, we will find the following compulsory dictionary files:

    - *`thermophysicalProperties`*
    - *`momentumTransport`*

- *`thermophysicalProperties`* contains the definition of the physical properties of the working fluid.
- *`momentumTransport`* contains the definition of the turbulence model to use.

Flow past a cylinder – From laminar to turbulent flow

The *thermophysicalProperties* dictionary file

```
16   thermoType
17   {
18       type            hePsiThermo;
19       mixture         pureMixture;
21       transport       sutherland;
22       thermo          hConst;
23       equationOfState perfectGas;
24       specie          specie;
25       energy          sensibleEnthalpy;
26   }

28   mixture
29   {
30       specie
31       {
32           nMoles      1;
33           molWeight   28.9;
34       }
35       thermodynamics
36       {
37           Cp          1005;
38           Hf          0;
39       }
40       transport
41       {
47             As          1.4792e-06;
48             Ts          116;
49        }
50   }
```

- This dictionary file is located in the directory **constant**. Thermophysical models are concerned with energy, heat and physical properties.
- In the sub-dictionary **thermoType** (lines 16-26), we define the thermophysical models.
- The **transport** modeling concerns evaluating dynamic viscosity (line 21). In this case the viscosity is computed using the Sutherland model.
- The thermodynamic models (**thermo**) are concerned with evaluating the specific heat Cp (line 22). In this case Cp is constant
- The **equationOfState** keyword (line 23) concerns to the equation of state of the working fluid. In this case

$$\rho = \frac{p}{RT}$$

- The form of the energy equation to be used in the solution is specified in line 25 (**energy**). In this case we are using enthalpy (**sensibleEnthalpy**).

Flow past a cylinder – From laminar to turbulent flow

The *thermophysicalProperties* dictionary file

```
16    thermoType
17    {
18        type            hePsiThermo;
19        mixture         pureMixture;
21        transport       sutherland;
22        thermo          hConst;
23        equationOfState perfectGas;
24        specie          specie;
25        energy          sensibleEnthalpy;
26    }

28    mixture
29    {
30        specie
31        {
32            nMoles      1;
33            molWeight   28.9;
34        }
35        thermodynamics
36        {
37            Cp          1005;
38            Hf          0;
39        }
40        transport
41        {
47             As          1.4792e-06;
48             Ts          116;
49         }
50    }
```

- In the sub-dictionary **mixture** (lines 28-50), we define the thermophysical properties of the working fluid.
- In this case, we are defining the properties for air at 20° Celsius and at a sea level.
- The constants As and Ts (lines 47-48), are related to the Sutherland model.

Flow past a cylinder – From laminar to turbulent flow

The *momentumTransport* dictionary file

- This dictionary file is located in the directory **constant**.
- In this dictionary file we select what model we would like to use (laminar or turbulent).
- In this case we are interested in modeling turbulence, therefore the dictionary is as follows

```
simulationType  RAS;                      RANS type simulation

RAS                       RANS sub-dictionary
{
    RASModel        kOmegaSST;            RANS model to use

    turbulence      on;                   Turn on/off turbulence.  Runtime modifiable

    printCoeffs     on;                   Print coefficients at the beginning
}
```

- If you want to know the models available use the banana method.

Flow past a cylinder – From laminar to turbulent flow

The `0` directory

- In this directory, we will find the dictionary files that contain the boundary and initial conditions for all the primitive variables.
- As we are solving the compressible laminar Navier-Stokes equations, we will find the following field files:

• *`p`*	(pressure)
• *`T`*	(temperature)
• *`U`*	(velocity field)
• *`k`*	(turbulent kinetic energy)
• *`omega`*	(specific turbulent dissipation rate)
• *`nut`*	(turbulent viscosity)
• *`alphat`*	(turbulent thermal diffusivity)

Flow past a cylinder – From laminar to turbulent flow

The file *0/p*

```
17    dimensions       [1 -1 -2 0 0 0 0];
18
19    internalField    uniform 101325;
20
21    boundaryField
22    {
23        in
24        {
25            type             zeroGradient;
26        }
28        out
29        {
30            type             fixedValue;
31            value            uniform 101325;
32        }
34        cylinder
35        {
36            type             zeroGradient;
37        }
39        sym1
40        {
41            type             symmetry;
42        }
44        sym2
45        {
46            type             symmetry;
47        }
49        back
50        {
51            type             empty;
52        }
54        front
55        {
56            type             empty;
57        }
58    }
```

- This file contains the boundary and initial conditions for the scalar field pressure (**p**). **We are working with absolute pressure.**
- Contrary to incompressible flows where we defined relative pressure, this is the absolute pressure.
- Also, pay attention to the units (line 17). The pressure is defined in Pascal.
- We are using uniform initial conditions (line 19).
- For the **in** patch, we are using a **zeroGradient** boundary condition.
- For the **outlet** patch we are using a **fixedValue** boundary condition.
- For the **cylinder** patch we are using a **zeroGradient** boundary condition.
- The rest of the patches are constrained.

Flow past a cylinder – From laminar to turbulent flow

The file *0/T*

```
17    dimensions       [0 0 0 -1 0 0 0];
18
19    internalField   uniform 293.15;
20
21    boundaryField
22    {
23        in
24        {
25            type              fixedValue;
26            value             $internalField;
27        }
29        out
30        {
31            type              inletOutlet;
32            value             $internalField;
33            inletValue        $internalField;
34        }
36        cylinder
37        {
38            type              zeroGradient;
39        }
41        sym1
42        {
43            type              symmetry;
44        }
46        sym2
47        {
48            type              symmetry;
49        }
51        back
52        {
53            type              empty;
54        }
56        front
57        {
58            type              empty;
59        }
60    }
```

- This file contains the boundary and initial conditions for the scalar field temperature (**T**).
- Also, pay attention to the units (line 17). The temperature is defined in Kelvin.
- We are using uniform initial conditions (line 19).
- For the **in** patch, we are using a **fixedValue** boundary condition.
- For the **out** patch we are using an **inletOutlet** boundary condition (in case of backflow).
- For the **cylinder** patch we are using a **zeroGradient** boundary condition.
- The rest of the patches are constrained.

Flow past a cylinder – From laminar to turbulent flow

The file *0/U*

```
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (150 0 0);
20
21  boundaryField
22  {
23      in
24      {
25          type            fixedValue;
26          value           uniform (150 0 0);
27      }
29      out
30      {
31          type            inletOutlet;
32          phi             phi;
33          inletValue      uniform (0 0 0);
34          value           uniform (0 0 0);
35      }
37      cylinder
38      {
39          type            fixedValue;
40          value           uniform (0 0 0);
41      }
43      sym1
44      {
45          type            symmetry;
46      }
48      sym2
49      {
50          type            symmetry;
51      }
53      back
54      {
55          type            empty;
56      }
58      front
59      {
60          type            empty;
61      }
62  }
```

- This file contains the boundary and initial conditions for the dimensional vector field **U**.
- We are using uniform initial conditions and the numerical value is **(150 0 0)** (keyword **internalField** in line 19).
- For the **in** patch, we are using a **fixedValue** boundary condition.
- For the **out** patch we are using an **inletOutlet** boundary condition (in case of backflow).
- For the **cylinder** patch we are using a **zeroGradient** boundary condition.
- The rest of the patches are constrained.

Flow past a cylinder – From laminar to turbulent flow

The **`system`** directory

- 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.

Flow past a cylinder – From laminar to turbulent flow

The *controlDict* dictionary

```
17      application      rhoPimpleFoam;
18
19      startFrom        startTime;
20      //startFrom        latestTime;
21
22      startTime        0;
23
24      stopAt           endTime;
25      //stopAt   writeNow;
26
27      endTime          0.01;
28
29      deltaT           0.000001;
30
31      writeControl     adjustableRunTime;
32
33      writeInterval    0.00005;
34
35      purgeWrite       100;
36
37      writeFormat      ascii;
38
39      writePrecision   10;
40
41      writeCompression off;
42
43      timeFormat       general;
44
45      timePrecision    6;
46
47      runTimeModifiable true;
48
49      adjustTimeStep   yes;
50      maxCo            10;
51      maxDeltaT        0.001;
```

- This case will start from the last saved solution (**startFrom**). If there is no solution, the case will start from time 0 (**startTime**).
- It will run up to 0.01 seconds (**endTime**).
- The initial time-step of the simulation is 0.000001 seconds (**deltaT**).
- It will write the solution every 0.00005 seconds (**writeInterval**) of simulation time (**adjustableRunTime**). The option **adjustableRunTime** will adjust the time-step to save the solution at the precise intervals. This may add some oscillations in the solution as the CFL is changing.
- It will keep the latest 100 solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**).
- And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.
- In line 49 we turn on the option **adjustTimeStep**. This option will automatically adjust the time-step to achieve the maximum desired courant number (line 50).
- We also set a maximum time-step in line 51.
- Remember, the first time-step of the simulation is done using the value set in line 29 and then it is automatically scaled to achieve the desired maximum values (lines 50-51).
- The feature **adjustTimeStep** is only present in the **PIMPLE** family solvers, but it can be added to any solver by modifying the source code.

Flow past a cylinder – From laminar to turbulent flow

The `controlDict` dictionary

```
55     functions
56     {

178    forceCoeffs_object
179    {
188     type forceCoeffs;
189     functionObjectLibs ("libforces.so");
190     patches (cylinder);
191
192     pName p;
193     Uname U;
194
195     //rho rhoInf;
196     rhoInf 1.205;
197
198     //// Dump to file
199     log true;
200
201     CofR (0.0 0 0);
202     liftDir (0 1 0);
203     dragDir (1 0 0);
204     pitchAxis (0 0 1);
205     magUInf 150;
206     lRef 0.002;
207     Aref 0.000002;
208
209      writeControl   timeStep;
210      writeInterval  1;
211      }


312    };
```

- As usual, at the bottom of the `controlDict` dictionary file we define the **functionObjects** (lines 55-312).
- Of special interest is the **functionObject forceCoeffs_object**.
- As we changed the domain dimensions and the inlet velocity, we need to update the reference values (lines 205-207).
- It is also important to update the reference density (line 196).

Flow past a cylinder – From laminar to turbulent flow

The *fvSchemes* dictionary

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default           cellLimited leastSquares 1;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind default;

    div(phi,K)      Gauss linear;
    div(phi,h)      Gauss linear;

    div(phi,omega)  Gauss upwind;
    div(phi,k)      Gauss upwind;

    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear limited 1;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         limited 1;
}
```

- In this case, for time discretization (**ddtSchemes**) we are using the **Euler** method.
- For gradient discretization (**gradSchemes**) we are using the **leastSquares** method.
- For the discretization of the convective terms (**divSchemes**) we are using **linearUpwind** interpolation with no slope limiters for the term **div(phi,U)**.
- For the terms **div(phi,K)** (kinetic energy) and **div(phi,h)** (enthalpy) we are using **linear** interpolation method with no slope limiters.
- For the terms **div(phi,omega)** and **div(phi,k)** (turbulent quantities) we are using **upwind i**nterpolation method.
- For the term **div(((rho*nuEff)*dev2(T(grad(U)))))** we are using **linear** interpolation (this term is related to the turbulence modeling).
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear limited 1** method.
- This method is second order accurate.

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0.01;
        minIter          2;
    }
    pFinal
    {
        $p;
        relTol          0;
        minIter          2;
    }
    "U.*"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-08;
        relTol          0;
        minIter          2;
    }
    "(h|hFinal)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-08;
        relTol          0;
        minIter          2;
    }

}
```

- To solve the pressure (**p**) we are using the **PCG** method with an absolute **tolerance** of 1e-6 and a relative tolerance **relTol** of 0.01.
- The entry **pFinal** refers to the final correction of the **PISO** loop. Notice that we are using macro expansion (**$p)** to copy the entries from the sub-dictionary **p**.
- To solve **U** and **UFinal** (**U.***) we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0.
- To solve **hFinal** (enthalpy) we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0.
- Be careful with the enthalpy, it might cause oscillations.

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
solvers
{

    "(omega|k).*"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-08;
        relTol          0;
        minIter          2;
    }

    "rho.*"
    {
        solver         diagonal;
    }

}
```

- To solve the turbulent quantities **omega** and **k** we are using the solver **PBiCGStab** with an absolute **tolerance** of 1e-8 and a relative tolerance **relTol** of 0.
- To solve **rho** and **rhoFinal** (**rho.***) we are using the **diagonal** solver (remember rho is found from the equation of state, so this is a back-substitution).
- FYI, solving for the velocity is relatively inexpensive, whereas solving for the pressure is expensive.

Flow past a cylinder – From laminar to turbulent flow

The *fvSolution* dictionary

```
97
98      PIMPLE
99      {
100         momentumPredictor yes;
101          consistent yes;

103         nOuterCorrectors 2;
104         nCorrectors     2;
105         nNonOrthogonalCorrectors 1;

123      }

125     relaxationFactors
126     {
127         fields
128         {
129             ".*"            0.9;
130         }
131         equations
132         {
133             ".*"            0.9;
134         }
135     }
```

- The **PIMPLE** sub-dictionary contains entries related to the pressure-velocity coupling (in this case the **PIMPLE** method).
- Setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- Hereafter we are doing 2 **PISO** correctors (**nCorrectors**) and 1 non-orthogonal corrections (**nNonOrthogonalCorrectors**).
- If we increase the number of **nCorrectors** and **nNonOrthogonalCorrectors** we gain more stability but at a higher computational cost.
- The choice of the number of corrections is driven by the quality of the mesh and the physics involve.
- You need to do at least one **PISO** loop (**nCorrectors**).
- In lines 125-135 we setup the under-relaxation factors used during the outer corrections of the **PIMPLE** method.
    - The values defined correspond to the industry standard of the **SIMPLE** method.
    - By using under-relaxation, we ensure diagonal equality.
    - Be careful not use too low values as you will lose time accuracy.
    - If you want to disable under-relaxation, comment out these lines.

Flow past a cylinder – From laminar to turbulent flow

Running the case – Using a compressible solver

- You will find this tutorial in the directory **`$PTOFC/101OF/vortex_shedding/c24`**
- Feel free to use the Fluent mesh or the mesh generated with `blockMesh`. In this case we will use `blockMesh`.
- To run this case, in the terminal window type:

1.
```
$> foamCleanTutorials
```
2.
```
$> blockMesh
```
3.
```
$> transformPoints “scale=(0.001 0.001 0.001)”
```
4.
```
$> renumberMesh -overwrite
```
5.
```
$> rhoPimpleFoam | tee log
```
6.
```
$> pyFoamPlotWatcher.py log
```
You will need to launch this script in a different terminal

7.
```
$> gnuplot scripts0/plot_coeffs
```
You will need to launch this script in a different terminal

8.
```
$> rhoPimpleFoam -postProcess -func MachNo
```
9.
```
$> paraFoam
```

Flow past a cylinder – From laminar to turbulent flow

Running the case – Using a compressible solver

- In step 3 we scale the mesh.
- In step 4 we use the utility `renumberMesh` to make the linear system more diagonal dominant, this will speed-up the linear solvers.
- In step 5 we run the simulation and save the log file. Notice that we are sending the job to background.
- In step 6 we use `pyFoamPlotWatcher.py` to plot the residuals on-the-fly. As the job is running in background, we can launch this utility in the same terminal tab.
- In step 7 we use the gnuplot script `scripts0/plot_coeffs` to plot the force coefficients on-the-fly. Besides monitoring the residuals, is always a good idea to monitor a quantity of interest. Feel free to take a look at the script and to reuse it.
- In step 8 we use the utility `MachNo` to compute the Mach number.

Flow past a cylinder – From laminar to turbulent flow

- In the directory **`$PTOFC/101OF/vortex_shedding`**, you will find 28 variations of the cylinder case involving different solvers and models.
- There are no exercises in this section, just play around with the different cases.

Flow past a cylinder – From laminar to turbulent flow

- This is what you will find in each directory,
  - c1 = blockMesh – icoFoam – Unsteady solver – Re = 200.
  - c2 = fluentMeshToFoam – icoFoam – Unsteady solver – Re = 200.
  - c3 = blockMesh – pisoFoam – Unsteady solver – Field initialization – Re = 200.
  - c4 = blockMesh – potentialFoam – Re = 200.
  - c5 = blockMesh – mapFields – pisoFoam – Unsteady solver – original mesh – Re = 200.
  - c6 = blockMesh – mapFields – pisoFoam – Unsteady solver – Finer mesh – Re = 200.
  - c7 = blockMesh – pimpleFoam – Unsteady solver – Re = 200 – No turbulent model.
  - c8 = blockMesh – pisoFoam – Unsteady solver – Re = 200 – No turbulent model.
  - c9 = blockMesh – pisoFoam – Unsteady solver – Re = 200 – k-Omega SST turbulent model.
  - c10 = blockMesh – simpleFoam – Steady solver – Re = 200 – k-Omega SST turbulent model.
  - c11 = blockMesh – simpleFoam – Steady solver – Re = 40 – No turbulent model.
  - c12 = blockMesh – pisoFoam – Unsteady solver – Re = 40 – No turbulent model.
  - c14 = blockMesh – pimpleFoam – Unsteady solver – Re = 10000 – K-Omega SST turbulence model with wall functions.
  - c15 = blockMesh – pimpleFoam – Unsteady solver – Re = 100000 – K-Omega SST turbulence model with wall functions
  - c16 = blockMesh – simpleFoam – Steady solver – Re = 100000 – K-Omega SST turbulence model no wall functions.
  - c17 = blockMesh – simpleFoam – Steady solver – Re = 100000 – K-Omega SST turbulent model with wall functions.
  - c18 = blockMesh – pisoFoam – Unsteady solver – Re = 100000, LES Smagorinsky turbulent model.

Flow past a cylinder – From laminar to turbulent flow

- This is what you will find in each directory,
  - c19 = blockMesh – pimpleFoam – Unsteady solver – Re = 1000000 – Spalart Allmaras turbulent model no wall functions.
  - c20 = blockMesh – rhoPimpleFoam – Unsteady solver – Mach = 2.0 – Compressible – Laminar.
  - c21 = blockMesh – rhoPimpleFoam –Unsteady solver – Mach = 2.0 – Unsteady solver – Compressible – K-Omega SST turbulent model with wall functions.
  - c22 = blockMesh – rhoSimpleFoam – Mach = 2.0 – Steady solver – Compressible – K-Omega SST turbulent model with wall functions.
  - c23 = blockMesh – pimpleFoam – Unsteady solver – Re = 200 – No turbulent model – Source terms (momentum)
  - c24 = blockMesh – rhoPimpleFoam – Unsteady solver – Re = 20000 – Turbulent, compressible
  - c25 = blockMesh – rhoPimpleFoam – Unsteady solver – Re = 200 – Laminar, compressible, isothermal
  - c26 = blockMesh – pimpleFoam – Unsteady solver – Re = 200 – Laminar, moving cylinder (oscillating).
  - c27 = blockMesh – pimpleFoam/pimpleFoam – Unsteady solver – Re = 200 – Laminar, rotating cylinder using AMI patches.
  - c28 = blockMesh – interFoam – Unsteady solver – Laminar, multiphase, free surface.