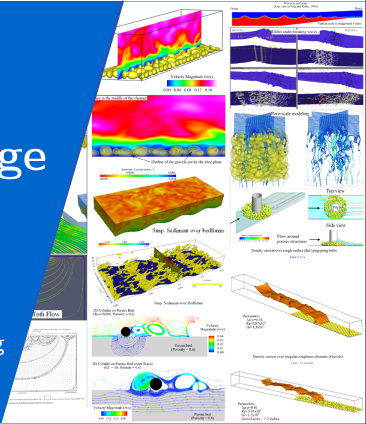
CE597D Special Topics, Fall, 2014

Chapter 2, Part 3: Structure and basic usage of OpenFOAM®

Xiaofeng Liu, Ph.D., P.E.
Assistant Professor
Department of Civil and Environmental Engineering
Pennsylvania State University
xliu@engr.psu.edu


Velocity Magnitude (m/s)
Eddies under breaking waves
Pore-scale modeling
Outline of the gravels cut by the slice plane
Susp. Sediment over bedforms
Top view
Side view
Flow around porous structures
Toth Flow
2D Cylinder on Porous Bed
Porous bed (Porosity = 0.6)
2D Cylinder on Porous Bed under Waves
Density current over irregular roughness elements (Gravels)

Outline

Level of users

- Beginner level:
  - A brief introduction of OpenFOAM®
  - Demonstration of OpenFOAM® usages
  - Installation of OpenFOAM®
  - Hands-on exercises for the basics
  - Detailed walk-through of the code
  - ...
- Intermediate and advanced levels (later):
  - Demonstration of OpenFOAM® development
    - Solver
    - Boundary conditions
    - Tools
  - Hands-on exercise
  - Discussion of specific applications

What we have done so far:

- *A brief introduction of OpenFOAM®*
- *Sample applications of OpenFOAM®*
- Installation of OpenFOAM®
  - Brief introduction of Linux system
  - Installation of OpenFOAM®
  - Test the installation
  - Initial browsing of the OpenFOAM® code
- Structure and basic usage of OpenFOAM®
- Detailed walk-through of the code

Basic usage and Hands-on exercise

In this lecture, we will do:

- Basic structure of OpenFOAM® : code and simulation cases
- Explain the basic usage of OpenFOAM® (as an entry-level user)
  1. Lid-driven case
- Hands-on exercises

Basic structure of OpenFOAM

OpenFOAM® is highly organized through its directory structure:

- We can have a look at its directory structure by using the `tree` command

```
$ foam  (Note: this alias command takes to your OpenFOAM instal
$ tree -C -d -L 1
.
|-- applications
|-- bin
|-- doc
|-- etc
|-- platforms
|-- src
|-- tutorials
|-- wmake
```

- The meaning and function of each sub-directory are apparent from the names.
- You can navigate into each of the sub-directory and use the `tree` command recursively and inspect the content.
- Inside this top level directory, there are also other files, including `Allwmake`, which you will use to compile the whole OpenFOAM® package

Basic structure of OpenFOAM

The applications directory:

-
  ```
  $ cd applications  (Note: or use the app alias command)
  $ tree -C -d -L 1
  .
  |-- Allwmake
  |-- solvers
  |-- test
  |-- utilities
  ```
- The three sub-directories contains three different categories
  - `solvers`: Each is designed to solve a set of PDEs, such as `icoFoam`.
  - `utilities`: perform pre- and post-processing tasks, such as mesh generation, parallel decomposition, etc.
  - `test`: A lot of test programs for specific functions of OpenFOAM® , such as `volField` tests field reading and manipulation.

Basic structure of OpenFOAM

The `src` directory (where most of things happen):

-
  ```
  $ src  (Note: use the src alias command)
  $ tree -C -d -L 1
  .
  |-- Allwmake
  |-- finiteVolume
  |-- OpenFOAM
  |-- turbulenceModels
  ...
  35 directories, 1 file
  ```
- Each sub-directory contains a different fundamental part of the OpenFOAM® platform.
- Most important ones related to CFD and FVM:
  - `OpenFOAM`: fundamental data structures, algorithms, dimensions, etc.
  - `finteVolume`: machinery related to FVM discretization, such as field, mesh, numerical schemes, etc.
  - `turbulence`: all kinds of turbulence models (RANS and LES)

Basic structure of OpenFOAM

The `src` directory (where most of things happen):

- In particular, `src/finiteVolume/cfdTools/general/include/fvCFD.H` is a header file which is included in most of OpenFOAM® code

```
$ cat finiteVolume/cfdTools/general/include/fvCFD.H
#include "parRun.H"

#include "Time.H"
#include "fvMesh.H"
#include "fvc.H"
#include "fvMatrices.H"
#include "fvm.H"
#include "uniformDimensionedFields.H"
#include "OSspecific.H"
#include "argList.H"
#include "timeSelector.H"
```

- It loads many of the important header files for things like time, mesh, matrix, and some operating system specific settings.
- In many code which does not include `fvCFD.H`, a subsect of it is usually included.

Basic structure of OpenFOAM

The `bin` directory:

- It contains pre-defined scripts which are for functions such as submit OpenFOAM® job, analyze job log, and visualization using `ParaView` (`paraFoam`).

```
$ foam
$ cd bin
$ ls
engridFoam        foamNew            foamSystemCheck
finddep           foamNewCase        foamTags
findEmptyMake     foamNewSource      foamUpdateCaseFileHeader
...
```

- This directory is automatically added to the `PATH` environmental variable (if OpenFOAM® installed and configured correctly). So the scripts can be called anywhere in a terminal window.

Basic structure of OpenFOAM

The `doc` directory:

- It contains the documentation.
- Two good references:
  - `UserGuide.pdf`: How to use OpenFOAM®
  - `ProgrammersGuide.pdf`: How to program in OpenFOAM®
- `Doxgen` folder: You can use Doxgen documentation.
  - By default, it is not compiled. So you should compile yourself if you want to use Doxgen on your own computer
  - Or you can just go to http://www.openfoam.org/docs/cpp/

Basic structure of OpenFOAM

The `etc` directory:

- It contains the configuration files, such as `bashrc` which can be sourced to setup and use the current version of OpenFOAM® .

```
$  tree -C -L 1
.
|-- bashrc
|-- caseDicts
|-- cellModels
|-- codeTemplates
|-- config
|-- controlDict
|-- cshrc
|-- thermoData
```

- It also contains the following things which affect the functionality and behavior of OpenFOAM® :
  - `cellModels` file: definition of cells (hex, tet, etc.)
  - `config` folder: definition of alias, `ParaView` setup, `scotch` setup, etc.
  - `codeTemplates`: Some templates for writing C++ classes in OpenFOAM® style.

Basic structure of OpenFOAM

The `tutorials` directory:

- It contains the tutorials shipped with OpenFOAM® .

```
$  tree -C -L 1
Allclean  combustion       electromagnetics  lagrangian
Allrun    compressible     financial         mesh
Alltest   discreteMethods  heatTransfer      multiphase
basic     DNS              incompressible    resources
```

- The tutorial directory is organized to roughly reflect the available solvers in the `solvers` directory of OpenFOAM® .
- You should have made a copy of this folder to your own directory.
- It is not suggested to make any changes here; do it in your own direcotry instead.

Basic structure of OpenFOAM

The `platforms` directory:

- It contains binaries of the applications (solvers, tools, etc.) in the `bin` subdirectory and libraries in the `lib` subdirectory.
- For example, it might contain the following:

```
$ tree -C -L 2
.
|-- linux64Gcc47DPDebug
|    |-- bin
|    |-- lib
|-- linux64Gcc47DPOpt
     |-- bin
     |-- lib
```

- In this case, it actually contains two versions of the OpenFOAM® compilation in two different directories:
  - `Debug`: the debug version, runs slow but with a lot of diagonostic information on errors and warnings.
  - `Opt`: the optimized version, runs fast with less output if errors occur.

Basic structure of OpenFOAM

The `platforms` directory:

- How it works?
  - For example `linux64Gcc47DPDebug` corresponding to the concatenation of the following environmental variables
  - `WM_ARCH` + `WM_ARCH_OPTION` + `WM_COMPILER` + `WM_PRECISION_OPTION` + `WM_COMPILE_OPTION`
    - `WM_ARCH` = linux (set in `etc/config/settings.sh`)
    - `WM_ARCH_OPTION` = 64 (set in `etc/bashrc`)
    - `WM_COMPILER` = Gcc47 (set in `etc/bashrc`)
    - `WM_PRECISION_OPTION` = DP (set in `etc/bashrc`)
    - `WM_COMPILE_OPTION` = Debug (set in `etc/bashrc`)
- So when you compile OpenFOAM® , the compiler will use the values of the environment variables you set in the configuration files and put the resulting binaries and libraries into the proper place.
- When you run simulations, the environmental variables (`PATH`, `LD_LIBRARY_PATH`, etc.) are also setup so you are using the right version of the binaries and libraries.
  - `PATH`: modified in `etc/config/settings.sh`
  - `LD_LIBRARY_PATH`: modified mainly in `etc/config/settings.sh`

Basic structure of OpenFOAM

The `wmake` directory:

- OpenFOAM® use a special version of `make` to organize the source code compilation: `wmake`
  http://www.openwatcom.org/index.php/Using_wmake
- `wmake` is similar to GNU make. It aims to ease the management of large programming projects
- It uses macros to maintain the updating and dependents
- In this folder, pre-defined scripts and configuration files are stored. They will be used by `wmake` during compilation.
- Some often used commands in this directory: `wmake`, `wclean`, `wcleanAll`
- `wmake`'s understanding of the compiler is through the rules defined in the directory `rules`
  - It defines things like rules, compilation switches, parallel compilation
  - If you add a new compiler, you should modify accordingly.
- A decent explanation for OpenFOAM® :
  http://www.openfoam.org/docs/user/compiling-applications.php

Basic structure of OpenFOAM

- So far we have looked at the OpenFOAM® directory structure itself
- On a typical installation, you will have your own directory. At the end of an installation, you should have done:

```
$ mkdir -p $FOAM_RUN
```

- Do things in your own directory. For example, I have organized my own directory roughly corresponding to the OpenFOAM® directory structure:

```
$ cd $WM_PROJECT_USER_DIR
$ tree -C -L 2
.
|-- applications
|   |-- lib
|   |-- solvers
|   |-- tools
|   |-- utilities
|-- platforms
|   |-- linux64Gcc47DPDebug
|   |-- linux64Gcc47DPOpt
|-- run
|   |-- tutorials
```

Basic structure of OpenFOAM

Some user specific environmental variables:

- `WM_PROJECT_USER_DIR`: defined in `etc/bashrc` file, for example

  ```
  export WM_PROJECT_USER_DIR=$HOME/OpenFOAM/$WM_PROJECT/
          $USER-$WM_PROJECT_VERSION
  ```

  It defines the location of a user's own directory.

- `FOAM_USER_APPBIN` and `FOAM_USER_APPBIN`:
  - The `bin` and `lib` location for a specific user.
  - When you compile your own solver or library, you should instruct `wmake` to put the resulting files there so it does not mess up with the original OpenFOAM® installation.
  - For example, I wrote a solver with the following `Make/files` for `wmake`

    ```
    ibPisoFoam.C

    EXE = $(FOAM_USER_APPBIN)/ibPisoFoam
    ```

  - More on this later

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- Remember to always work in your own directory

```
$ run
$ cd tutorials/incompressible/icoFoam/cavity
$ ls
```

- Three steps in running a CFD simulation
  1. pre-processing: type `blockMesh` to generate the mesh. Other setups are already done and will be examined later. You can also type `checkMesh` to see information abou the mesh, such as mesh size, domain size, and some mesh quality metrics.
  2. run: type `icoFoam` to run the simulation.
  3. post-processing: We will use `ParaView` directly.

```
$ touch case.foam     (Note: create an empty file with
                       extension ‘‘foam’’)
$ paraview case.foam&
```

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- Now we can have a look at the directory structure of a typical OpenFOAM® simulation case.

```
$  tree -C -L 2
.
|-- 0
|   |-- p
|   |-- U
|-- 0.1
|   |-- p
|   |-- U
...
|-- constant
|   |-- polyMesh
|   |-- transportProperties
|-- system
    |-- controlDict
    |-- fvSchemes
    |-- fvSolution
```

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- 0 and subsequent time directories: contain solution variable field files, in this case pressure `p` and velocity `U`.
- Examine the content and format of field file. For example:

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
 }
```

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- In the subsequent time directories, the field variables have nonuniform values.
- As such, the `internalField` is a list with the nonuniform values at cell centers.
- If the boundary field is also nonuniform, a list with the values at the face centers of the patch will present.
- There is also a `phi` field file. It is the face flux $\mathbf{U} \cdot \mathbf{S}$.
- There is also a sub-directory `uniform` which contains information such as time step size and index.
- If the mesh is changing during the simulation, each time directory will contain a `polyMesh` sub-directory for the new mesh at this time step.

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- The `constant` directory contains model specifications and mesh definition
- In this case, we have a file named `transportProperties` and a directory `polyMesh`.
- The file `transportProperties` is very simple. It only defines the kinematic viscosity.

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

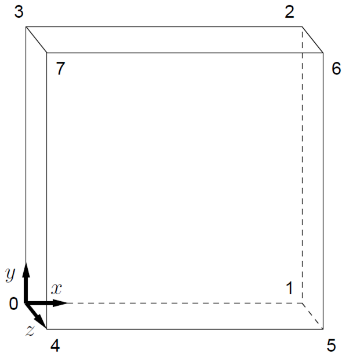
- For more complicated models, you will see more files in the directory, such as `turbulenceProperties`, `RASProperties`, and `LESProperties`.

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- The constant/polyMesh directory contains the definition of the mesh.
- Since we use blockMesh, there is a definition file named blockMeshDict.
- The basic idea of using blockMesh is to define vertices, lines, patches, and volumetric blocks. The first part of the file contains

```
convertToMeters 0.1;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

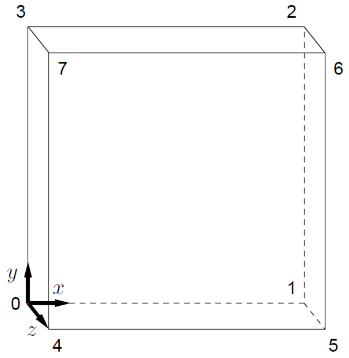


Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- The second part defines a block (with eight vertices)

```
blocks
(
    hex (0 1 2 3 4 5 6 7)
    (20 20 1) simpleGrading
    (1 1 1)
);
```



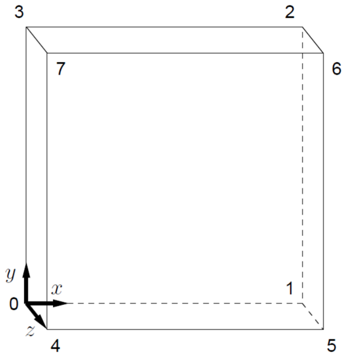
- The order of the vertices is important (right-hand system)
- `(20 20 1)` is the number of cells in each direction
- `simpleGrading (1 1 1)`: grading of the cell size in each direction

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- The next part defines boundary patches

```
boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```



Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- Each patch has a name, type, and a list of faces
- For example, a patch named `fixedWalls` has a type of `wall` with three faces defined in a list.

```
fixedWalls
{
    type wall;
    faces
    (
        (0 4 7 3)
        (2 6 5 1)
        (1 5 4 0)
    );
}
```

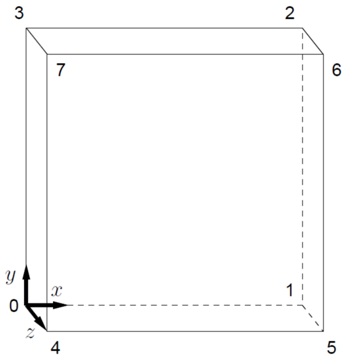
- The faces should be defined in a way that when the vertices are marched clock-wise when looking from inside of the block.

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- There is a special patch named `frontAndBack` with a type `empty`.
- The front and back faces of the block are `empty`, thus not contributing to the volume integration in finite volume method. As a result, the simulation is effectively a 2D case.
- Any idea how to make a 1D case?

```
frontAndBack
{
    type empty;
    faces
    (
        (0 3 2 1)
        (4 5 6 7)
    );
}
```



Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- After the mesh is generated with the blockMesh tool, there are several new files generated: boundary,faces,neighbour,owner, and points.
- boundary: defines the boundaries. Each field variable file (e.g., U) should define corresponding boundary conditions. This file defines the type of the boundary, how many faces in this boundary, and the start index in the global face list (in file faces).

```
movingWall
{
    type            wall;
    nFaces          20;
    startFace       760;
}
```

- points: defines the coordinates of all points in the mesh
- faces: defines all the faces (composed of point indices)
- owner and neighbour: defines the owner and neighbor of each face

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- `owner` and `neighbour`: defines the owner and neighbor of each face
  - Internal faces connect two cells:
  - Boundary faces only connect to one cell, i.e., owner cell; the neighbor cell is `-1`
- Why we need to define owner and neighbor?
  - One important place is to define the sign of flux. A positive flux goes from `owner` to `neighbor`
  - It is related to how the face normal is defined: from `owner` to `neighbor`.
  - For a boundary face, since there is no `neighbor`, the face normal always points outward.
- There is `cell` file: cell definitions are implicit and will be calculated upon the construction of a `fvMesh` object.
- You can find out how many cells are there in the mesh by check the line in the header of `neighbor` or `owner` files. For example,

```
note "nPoints: 882 nCells: 400 nFaces: 1640 nInternalFaces: 760
```

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- `system` directory contains the specifications for numerical discretization, linear solvers, run control, parallel computation, and other solution parameters.
- In this case, there are three files:
  - `controlDict`: starting and end time, time step size, whether dynamically adjust the time step size, output frequency, etc
  - `fvSchemes`: discretization schemes, such as time derivative, gradient, divergence, interpolation, etc.
  - `fvSolution`: specifications for the linear solvers. It also contains the specifications for the fluid dynamics solver.

Basic usage of OpenFOAM

Examination of the lid-driven cavity case:

- Parallel computation: This is a simple case and we only use it as a demo.
- To do parallel computation, you typically need to do three steps:
  1. decompose the case: it needs file `system/decomposeParDict` which defines how many subdivided domains and how to divide them.
  2. run in parallel
  3. (optional) recombine the results

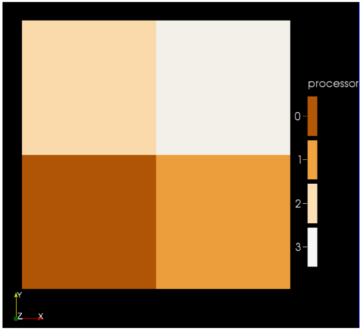


Figure: Run *decomposePare* with *cellDist* option and plot *cellDist* as category

Basic usage of OpenFOAM

- Most updated tutorials can be found in the `UserGuide.pdf`
- The best way to learn is by experimenting.
- Where to find help?
  - cfd-online forum
  - google: with openfoam as the first key word
  - ask around

Hands-on exercises

Two tutorials uploaded to ANGEL:

- `Extra_tutorial_on_OpenFOAM_basic_usage.pdf`
  1. It has more information on things like alias, environment variables, case structure, etc.
  2. It is roughly the replicate of this lecture. You need to at least do it once by yourself.
- `Tutorial_build_your_own_OpenFOAM_cases.pdf`
  1. More complicated cases: flow over step, meander channel, etc.
  2. Extra scripting tool: `m4`

Hands-on exercises

Build your own simulation cases:

- Flow over a step:
  - meshing
  - run
  - visualization

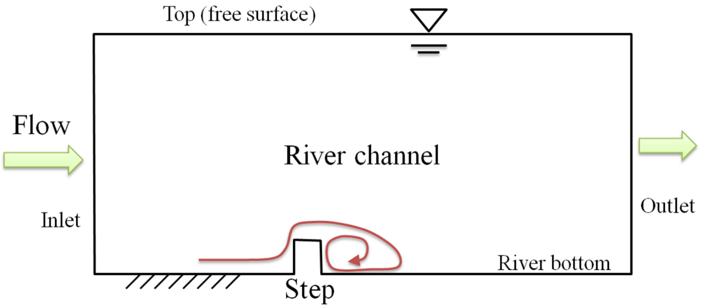
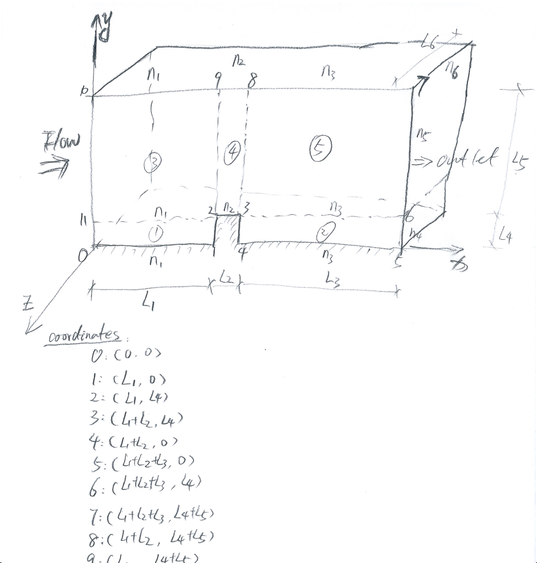


Figure: Scheme of the flow over step case

Hands-on exercises

Build your own simulation cases:

- Flow over a step:



coordinates:

0: $(0, 0)$

1: $(L_1, 0)$

2: (L_1, L_4)

3: (L_1+L_2, L_4)

4: $(L_1+L_2, 0)$

5: $(L_1+L_2+L_3, 0)$

6: $(L_1+L_2+L_3, L_4)$

7: $(L_1+L_2+L_3, L_4+L_5)$

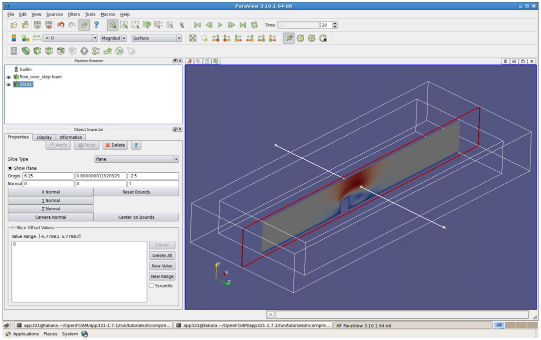
8: (L_1+L_2, L_4+L_5)

9: (L_1, L_4+L_5)

Hands-on exercises

Build your own simulation cases:

- Flow over a step:



U · Magnitud · Surface

builtin:
flow_over_step.foam
Slice1

Slice Type: Plane
[✓ Show Plane]
Origin: 6.25 | 0.800000011920929 | -2.5
Normal: 0 | 0 | 1

Slice Offset Values
Value Range: [-6.77883, 6.77883]
0

app321@takara:~/OpenFOAM/app321-1.7.1/run/tutorials/incompre...
app321@takara:~/OpenFOAM/app321-1.7.1/run/tutorials/incompre...
ParaView 3.10.1 64-bit

Figure: Visualization of flow over step case

Hands-on exercises

Build your own simulation cases:

- Meander channel case:

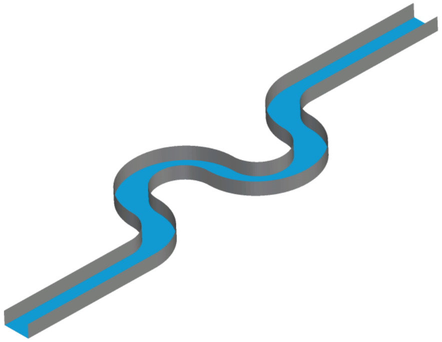
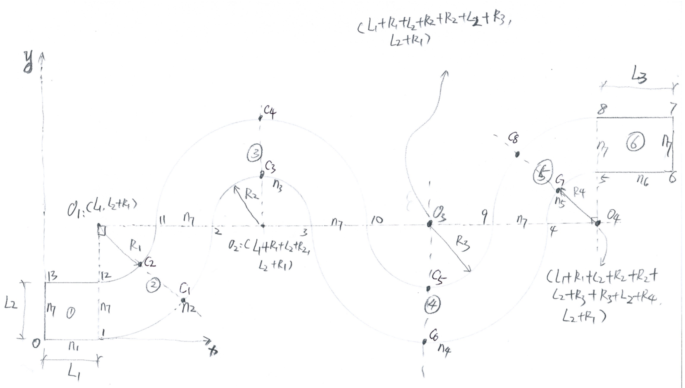


Figure: Scheme of the 3D meander channel case

Hands-on exercises

Build your own simulation cases:

- Meander channel case:



coordinates:

0: $(0, 0)$

1: $(L_1, 0)$

2: $(L_1+R_1+L_2, L_2+R_1)$

3: (x_2+2R_2, y_2)

4: $(x_3+2L_2+2R_3, y_2)$

5: (x_4+R_4, y_4+R_4)

6: (x_5+L_3, y_5)

7: (x_6, y_6+L_2)

8: (x_5, y_5+L_2)

9: $(x_4{}^{-L_2}, y_4)$

10: (x_9-2R_3, y_4)

11: $(x_{10}-2L_2-2R_3, y_{10})$

12: (x_1, L_2)

13: $(0, L_2)$

Hands-on exercises

Build your own simulation cases:

- Meander channel case:

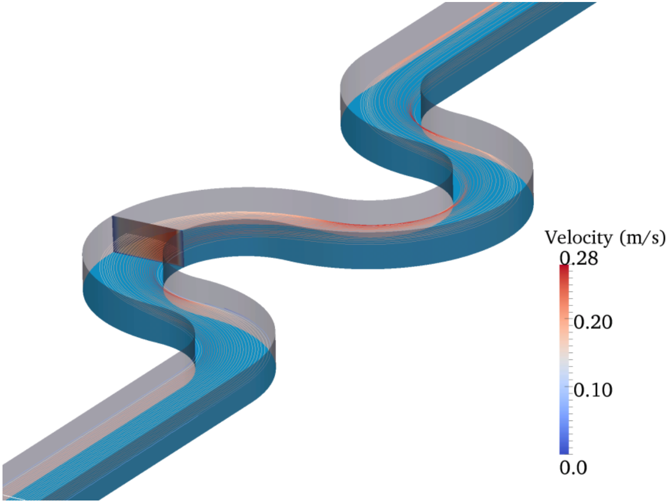


Figure: Streamlines of the 3D meander channel case