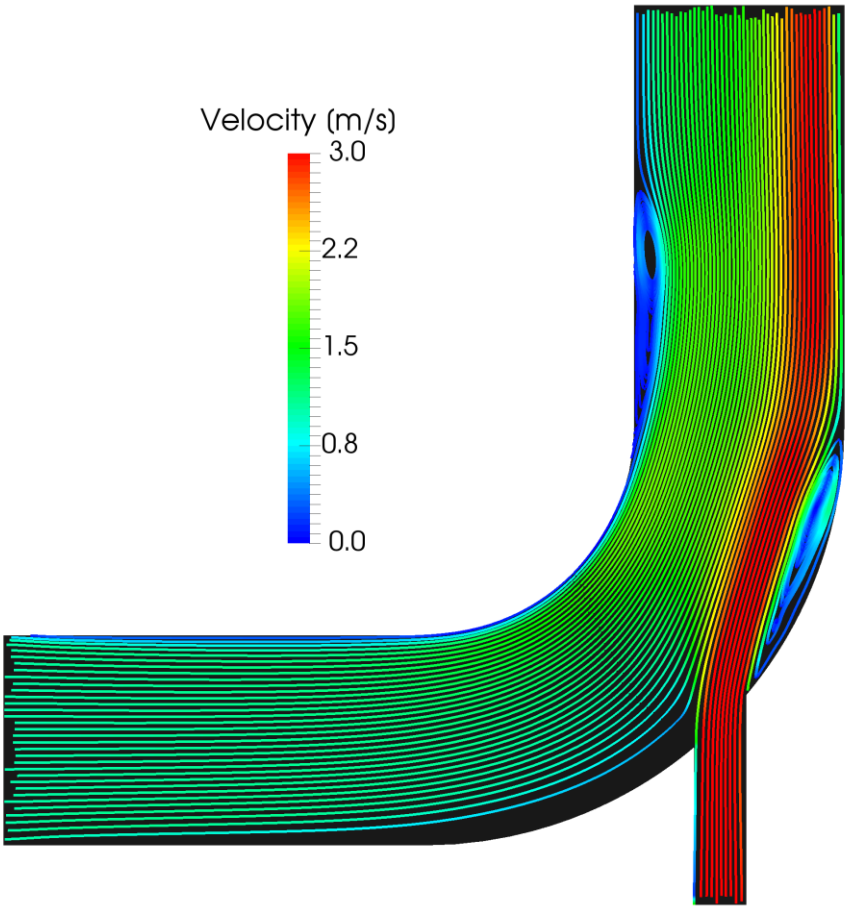
Tutorial One

Basic Case Setup



6th edition, April 2023

ENGINEERING FLUID DYNAMICS

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- OpenFOAM® v10

Cover picture from:

- Bahram Haddadi

Contributors:

- Bahram Haddadi
- Clemens Gößnitzer
- Jozsef Nagy
- Vikram Natarajan
- Sylvia Zibuschka
- Yitong Chen

TU WIEN ICEBE IMAGINEERING NATURE

ISBN 978-3-903337-02-2

Publisher: chemical-engineering.at

Available from: www.fluiddynamics.at

Background

1. What is CFD?

Computational fluid dynamics or CFD is the analysis of systems involving fluid flow, heat transfer and associated phenomena such as chemical reactions by means of computer-based simulation. The technique is very powerful and its application spans a wide range of industrial and non-industrial areas.

The ultimate aim of developments in the CFD field is to provide a capability comparable to other CAE (computer-aided engineering) tools such as stress analysis codes. The main reason why CFD has lagged behind is the tremendous complexity of the underlying behavior of fluid flows.

Although CFD has many advantages, it is not yet at the level where it can be blindly used without a working knowledge of the physics involved, and despite the increasing speed of computation available, CFD has not yet matured to a level where it can be used for real time computation. Numerical analyses require significant time to be set up and performed. CFD is still an aid to other analysis and experimental tools like wind tunnel testing, and is used in conjunction with them. So be careful!

The two most common types of CFD codes are:

- open and free
- closed source and commercial

We will be focusing on OpenFOAM®, which is a free, open source CFD code, written in C++. In addition, its source code is accessible and modifiable by its users. Therefore, you can even develop your own OpenFOAM® solver if you wish to!

2. Workflow of CFD

A CFD procedure is structured around the numerical algorithms that can tackle fluid flow problems, and the workflow mostly contains three main elements:

2.1. Pre-processing

- Definition of the geometry of the region of interest: the computational domain
- Grid generation – the sub-division of the flow region into a number of smaller, non-overlapping sub-domains: a grid (or mesh) of cells (or control volumes or elements)
- Selection of suitable models for the interesting physical and chemical phenomena
- Definition of fluid properties
- Specification of the appropriate chemical and physical boundary conditions at cells which coincide with or touch the domain boundary

The solution to a flow problem (velocity, pressure, temperature etc.) is defined at nodes or cell centers inside each cell. The accuracy of a CFD solution depends heavily on the number of cells in the grid. In general, the larger the number of cells, the better the solution accuracy. Optimal meshes are often non-uniform: finer in areas where large variations occur from point to point and coarser in regions with relatively little change.

2.2. Solver

There are at least four distinct streams of numerical solution techniques: finite difference, finite element, spectral methods and finite volume. We will only focus on the finite volume method, as it is central to the most well established CFD solvers. In outline, the finite volume method consists of the following steps:

- Integration of the conservation of mass, energy and momentum equations over all the control volumes in the domain
- Discretization – conversion of the resulting integral equations into a system of algebraic equations
- Solution of the algebraic equations by an iterative method

The first step, the control volume integration, makes the finite volume method different from all other CFD techniques. It makes sure that a general flow variable, e.g. momentum or enthalpy, is conserved in each finite size cell. This clear relationship between the numerical algorithm and the underlying conservation principle makes finite volume method popular and much simpler to understand.

2.3. Post-processing

This is where you look at the results and visualize them so that you can see what happens in your model. Typical elements of post-processing are:

- Definition of suitable cutting planes for visualization
- Contour plots of properties/flow variables
- Vector plots
- Streamlines
- Line plots
- Balances

There are several post-processing tools; fluent built-in post-processing tool, Ensight and TecPlot are some well-known commercial examples. There are also some open source tools such as Paraview and SALOME.

3. icoFoam solver

icoFoam is an OpenFOAM® solver suitable for analyzing incompressible, laminar flow of Newtonian fluids. It is based on the PISO algorithm (pressure-implicit split-operator), which is essentially a pressure-velocity iterative

procedure for transient problems. In each iterative step, PISO solves the momentum equation using one predictor step, with two further corrector steps for both velocity and pressure.

icoFoam – elbow

Tutorial outline

Using icoFoam solver, simulate 75 s of flow in an elbow for the following GAMBIT® meshes:

- Tri-mesh (comes with OpenFOAM® tutorial)
- Hex-mesh coarse (check GAMBIT® “elbow 2D” tutorial)
- 2 times finer hex-mesh (refined previous step mesh)

Objectives

- Basic case setup in OpenFOAM®
- Setting up initial values of p and U
- Ensuring proper boundary definitions (imported boundaries from GAMBIT®, additional surfaces during conversion and boundaries definition in OpenFOAM®)

Data processing

Import your simulation to ParaView, extract data to make two diagrams (using spreadsheet calculators) of pressure and velocity magnitude along a line between two tubes, do the same for all three simulations.

1. Pre-processing

1.1. Setting system environment

Make sure your system environment is set correctly under the chosen version of OpenFOAM® (v10), check Appendix B Part A.

1.2. Copying tutorial

Open a terminal and copy the elbow tutorial from the following path to your working directory (see Appendix A for running a terminal in Linux):

```
$FOAM_TUTORIALS/incompressible/icoFoam/elbow
```

Note: The '$' sign allows the tutorial to be extracted from the installation directory of OpenFOAM® under the current system environment.

1.3. Converting mesh

The mesh, which is produced by GAMBIT®, is not directly compatible with OpenFOAM®. First, the mesh needs to be converted to an OpenFOAM® mesh, using the following tool:

```
>fluentMeshToFoam elbow.msh
```

Note: the '>' sign is not part of the command. It is only used to show that the command should be typed inside a terminal.

If the mesh was created in mm and is converted using the mentioned command it will convert the mesh with wrong dimensions, since all the units in OpenFOAM® are SI Units (International System of Units). There are different flags included with most of OpenFOAM® tools, for checking them use the flag `-help` after the command, e.g.:

```
>fluentMeshToFoam -help
```

The output gives an overview of available options of the tool and a short description on how to use it:

```
Usage: fluentMeshToFoam [OPTIONS] <Fluent mesh file>
options:
  -2D <thickness>   use when converting a 2-D mesh (applied before scale)
  -case <dir>       specify alternate case directory, default is the cwd
  -fileHandler <handler>
                    override the fileHandler
  -libs <(lib1 .. libN)>
                    pre-load libraries
  -noFunctionObjects
                    do not execute functionObjects
  -scale <factor>   geometry scaling factor - default is 1
  -writeSets        write cell zones and patches as sets
  -writeZones       write cell zones as zones
  -srcDoc           display source code in browser
  -doc              display application documentation in browser
  -help             print the usage
Using: OpenFOAM-10 (see https://openfoam.org)
Build: 10
```

The `-scale` flag is used for converting the mesh dimensions from other units to SI units, e.g. if the mesh was created in mm it will be converted to meter by using `-scale 0.001` and if the flag is omitted, uses 1:

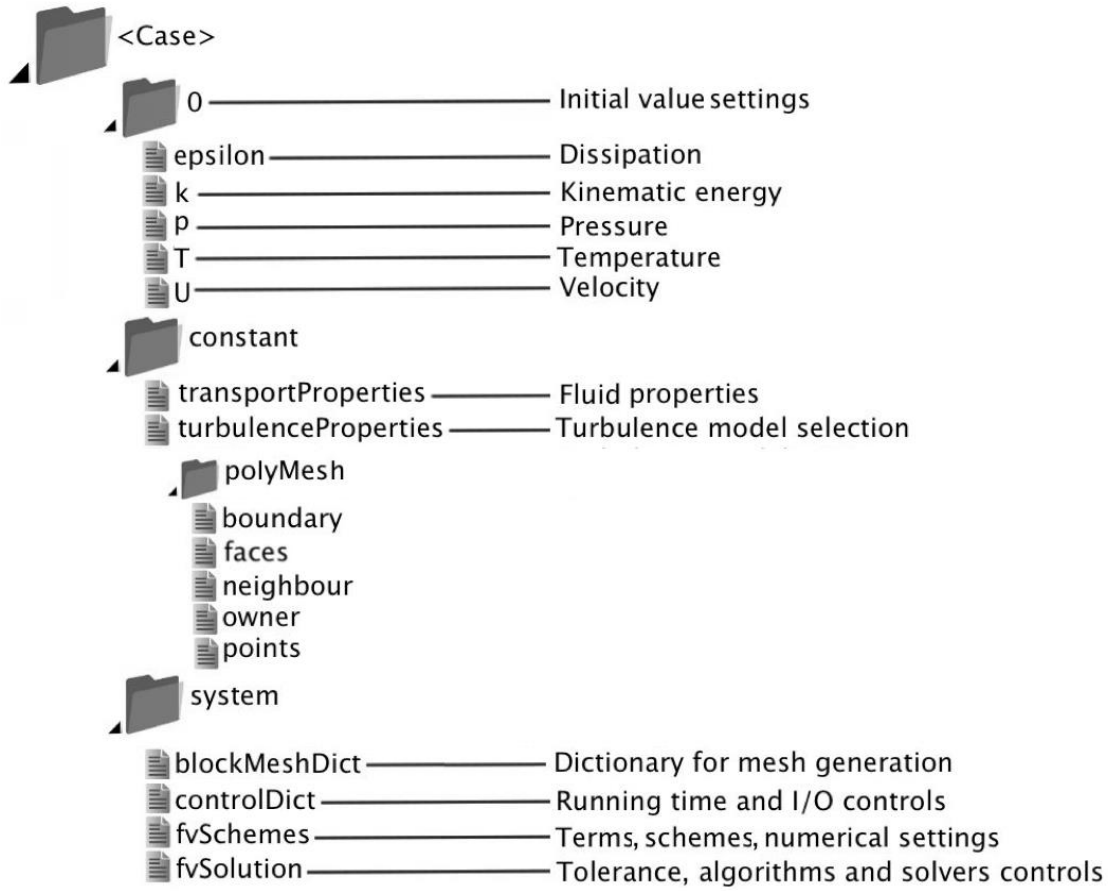
```
>fluentMeshToFoam elbow.msh -scale 1.0
```

Note: The mesh which is imported to OpenFOAM® should be a three dimensional mesh. For carrying out 2D (also 1D) simulations, a three-dimensional mesh should be created with just one cell in the third dimension (for 1D, one cell in the second and one cell in the third direction).

Note: If there are internal boundaries in the mesh, there is another tool, fluent3DMeshToFoam. Using this tool, the internal boundaries will be kept during conversion.

1.4. Case structure

Most of the cases in OpenFOAM® have the following basic case structure (directory tree):



There are three main directories (0, constant, system) in each case folder:

1.4.1. 0 directory

The 0 directory includes the initial conditions for running the simulation. In each file in this folder, the initial conditions for one property can be set. The files are named after the property they are standing for, e.g. usually T file includes temperature initial conditions. In the elbow example, there are only two files inside the 0 directory, p and U. p stands for pressure and U stands for velocity. Checking p:

```
>nano  p
```

It will be like this:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Website:  https://openfoam.org                  |
|   \\  /    A nd           | Version:  10                                    |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//

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

internalField   uniform 0;

boundaryField
{
    wall-4
    {
        type            zeroGradient;
    }

    velocity-inlet-5
    {
        type            zeroGradient;
    }

    velocity-inlet-6
    {
        type            zeroGradient;
    }

    pressure-outlet-7
    {
        type            fixedValue;
        value           uniform 0;
    }

    wall-8
    {
        type            zeroGradient;
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}
```

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

In `dimensions`, the physical dimension according to SI base units of the quantity is defined, for example here it shows that the p dimension is $(m/s)^2$.

Note: As you can see the p unit is not the pressure unit (Pa). It is due to the fact that in incompressible solvers in OpenFOAM® p is defined as pressure divided by density.

Note: In the dimension matrix the first number represents mass (kilogram), the second one the length (meter), the third one time (second), the forth one the temperature (Kelvin), the fifth one the quantity (mole), the sixth one current (ampere) and the last one luminous intensity (candela).

The `internalField` sets the initial field of a specific quantity in the solution domain. There are two types: uniform and non-uniform. Uniform field assigns a single value to all elements, whereas non-uniform field specifies a unique value to each field element.

The type of each of our boundaries as well as the value of this quantity on the boundaries is defined in the `boundaryField`. There are different types of boundary conditions in OpenFOAM®:

- `zeroGradient`: Applies a zero gradient boundary type to this boundary (Neumann boundary condition).
- `fixedValue`: Applies a fixed value to this boundary (Dirichlet boundary condition).
- `empty`: It is for sides, which are vertical to the direction that is not going to be considered (e.g. in 2D simulations these boundaries are vertical to the third dimension). In this boundary type both of the sides vertical to one dimension should be selected together and named as one boundary.

Note: If a `fixedValue` boundary condition with `value` equals `$internalField` is used, it is equal to using `zeroGradient`, except `zeroGradient` applies the boundary condition implicitly, but `fixedValue` with `$internalField` value applies the boundary condition explicitly.

The U file has to be defined via three components (since velocity is a vector): first one stands for the x component, second one for the y component, and the third one for the z component. For this case setup the z component is always zero because it is a 2D simulation and no calculations will be done in the z direction. The boundaries vertical to z direction have been already set to empty.

1.4.2. constant directory

The constant directory usually consists of a subdirectory and some files. The files (usually) include material properties, simulation physics and chemistry. In the directory “polyMesh” the mesh data are stored (in this case the data for converted mesh). The boundary file in this polyMesh directory includes the mesh boundary data, e.g. type, the patch group, number of faces on this boundary and also starting face number (unique face IDs) for this boundary (for

the sake of space, the dictionary headers will not be included in this scope any more):

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

6
(
    wall-4
    {
        type            wall;
        inGroups        List<word> 1(wall)
        nFaces          100;
        startFace       1300;
    }
    velocity-inlet-5
    {
        type            patch;
        nFaces          8;
        startFace       1400;
    }
…
    frontAndBackPlanes
    {
        type            empty;
        inGroups        List<word> 1(empty);
        nFaces          1836;
        startFace       1454;
    }
)

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

Comparing the boundary names with the ones set in GAMBIT®, they should be the same. Starting cell number and number of each face cells can also be checked here.

Note: *However, in terms of boundary type, empty boundary condition does not exist in GAMBIT®. All the faces perpendicular to the direction, which is not going to be considered, are defined as a new boundary with type wall. After importing the mesh to OpenFOAM®, modify that boundary in the file constant/polyMesh/boundary, and change its type from `wall` to `empty`, and change `inGroups` from `wall` to `empty`. In this case, after converting the mesh, the face `frontAndBackPlanes` needs to be modified for both hex-mesh and finer hex-mesh.*

By opening the physicalProperties file, properties dimensions and the property value can be found and edited, e.g.:

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

`nu` is the fluid kinematic viscosity, which is 0.01 m^2/s for this example.

1.4.3. system directory

Solver and finite volume methods settings can be found and changed in this directory. There are three main files in this directory:

- **fvSchemes**: The discretization scheme used for each term of the equations are set in this file.

- **fvSolution**: Contains the settings to the coupling method of pressure and velocity, the numerical methods, which are used for solving different quantities, and also the final tolerance for convergence of that quantity.
- **controlDict**: The time from where simulation starts (`startFrom`), the time when the simulation finishes (`stopAt`), the time step (`deltaT`), the data saving interval (`writeInterval`), the saved data file format (`writeFormat`), the saved file data precision (`writePrecision`), and also if changing the files during the run can affect the run or not (`runTimeModifiable`) are set in this file.

Note: *If the write format is `ascii`, then the simulation data which is written to the file can be opened and read using any text editor. If the format is `binary`, the data will be written in binary style and is not readable by text editors. The advantage of binary over ascii is the smaller file size, and consequently faster conversion and writing to disk, for big simulations.*

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
application      icoFoam;

startFrom        latestTime;

startTime        0;

stopAt           endTime;

endTime          75;

deltaT           0.05;

writeControl     timeStep;

writeInterval    20;

purgeWrite       0;

writeFormat      ascii;

writePrecision   6;

writeCompression off;

timeFormat       general;

timePrecision    6;

runTimeModifiable true;

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

Note: *This simulation continues from the last time step data, which is saved (`latestTime`). If there was no saved data, it will start from start time (`startTime`), which is zero in this case.*

2. Running simulation

The simulation can be run by typing the solver's name and executing it:

```
>icoFoam
```

Note: For running the simulation, the solver command (e.g. icoFoam) should be executed inside the copy of the tutorial main folder. For example: The command should be executed in the elbow folder, if it was run at some subfolders or somewhere else, the simulation will fail.

3. Post-processing

3.1. Exporting simulation data

The data files created by OpenFOAM® should be exported (converted) by the appropriate tools, to the post processing tools data format. For ParaView:

```
>foamToVTK
```

where VTK is the ParaView data format. This command should be also executed in the case main directory, e.g. elbow. Here, ParaView is used as the post-processing tool, for running it

```
>paraview &
```

Note: Another option to open the OpenFOAM® simulation results with ParaView without converting them to VTK; Create an empty text file in the main case directory, name it <someName>.foam (e.g. foam.foam), and execute the following command. This method is good for fast evaluation of the data in the middle of the simulation or with a decomposed case in parallel simulations:

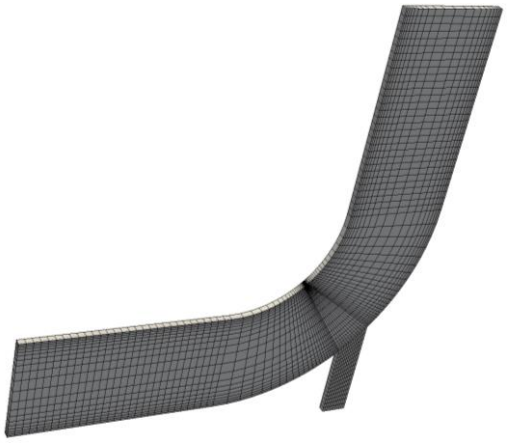
```
>paraview foam.foam &
```

Note: By putting & at the end of command, the command line will remain active and ready for further inputs while that program is running.

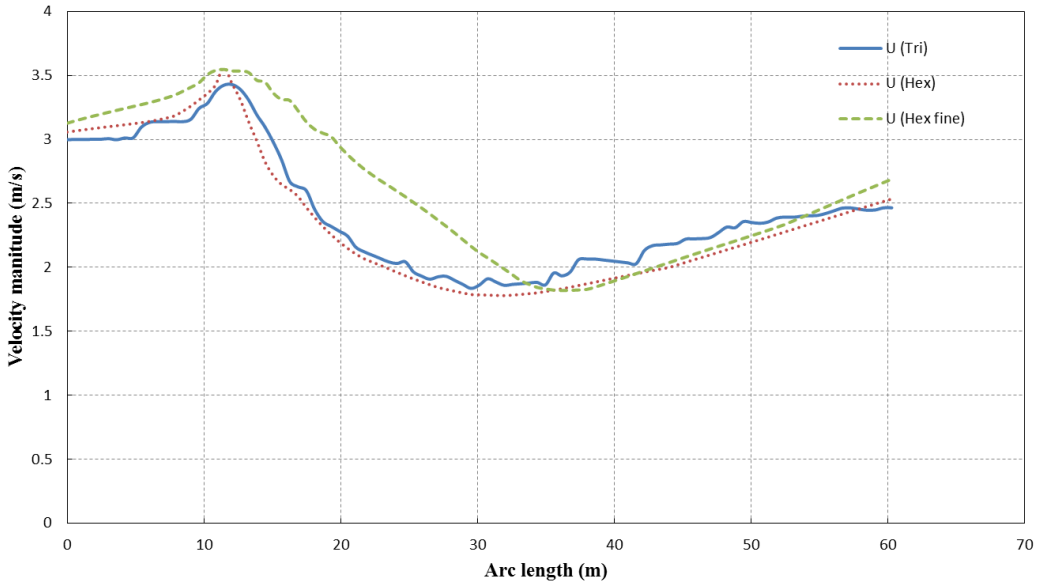
3.2. Examining different meshes

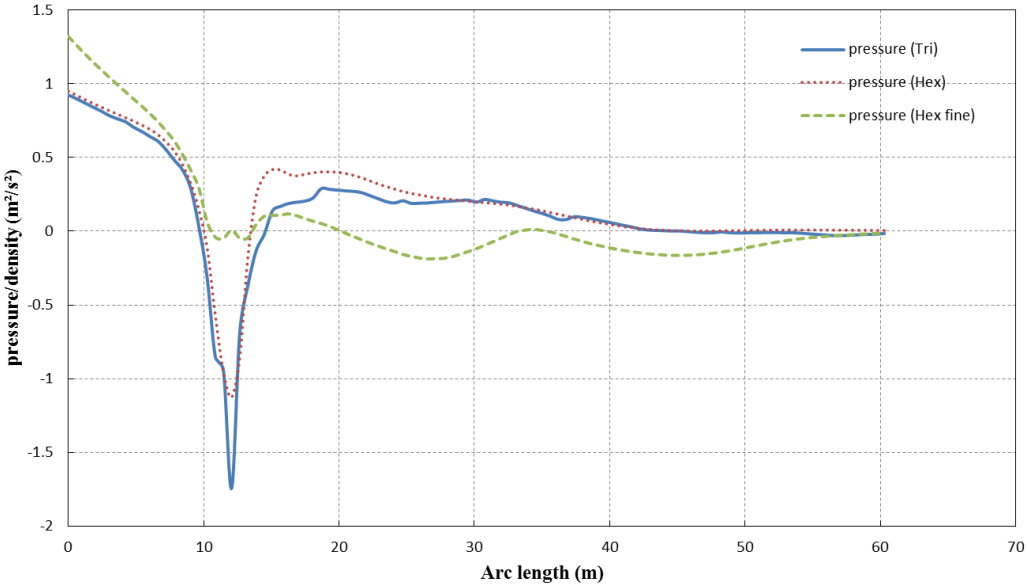
Do the same for the other two meshes. Only the mesh for the first simulation is included in the elbow example of OpenFOAM®. For the other two simulations, the mesh should be provided by the user. For finding the tutorials on how to create the geometry and the mesh, search the internet for "GAMBIT® elbow mesh 2D". The dimensions and the mesh info are provided in that tutorial. Try to create it by using GAMBIT® (or any other similar mesh creation tools). When you are done, you have to convert it into a 3D mesh with one cell in the z-direction.

The comparisons of all three case results and charts are shown below.



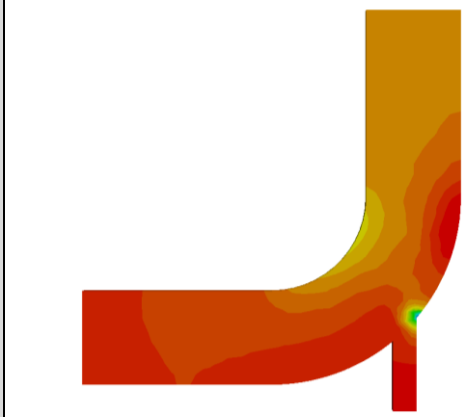
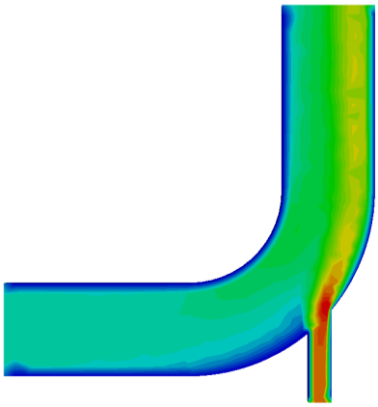
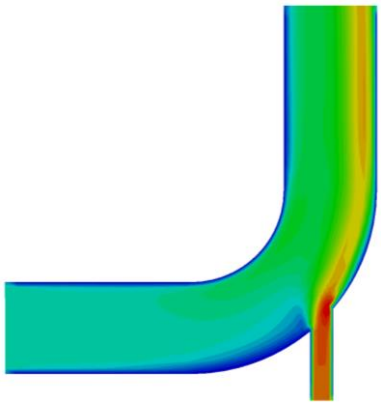
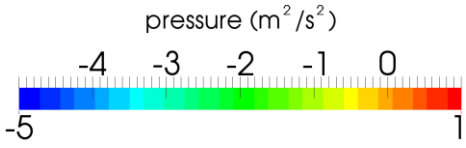
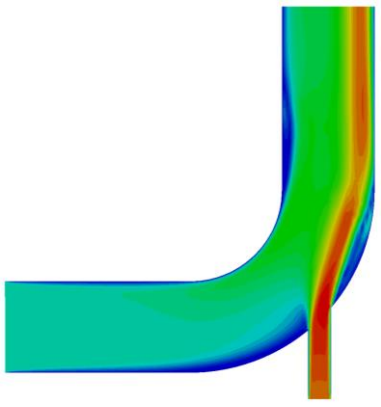
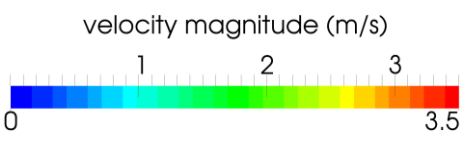
The Hex Fine mesh created using GAMBIT®





Pressure and velocity for different meshes at t=75 s, along the arc shown

The comparison plots are along the line between points A (54 0 0) at the small tube entrance and B (60 60 0) at the large tube exit part (length units are in meter) for Tri-mesh, for other two meshes created using GAMBIT® the points are A (22 -33 0) and B (27 30 0).

Mesh	Pressure	Velocity
Tri		
Hex		
Hex Fine	 	 

Comparison of different mesh type results at t = 75 s

Note: For extracting data over a line, the line should be defined in ParaView using "Plot Over Line", then the data over this line can be exported by choosing Save Data from File menu in ParaView.

ISBN 978-3-903337-02-2