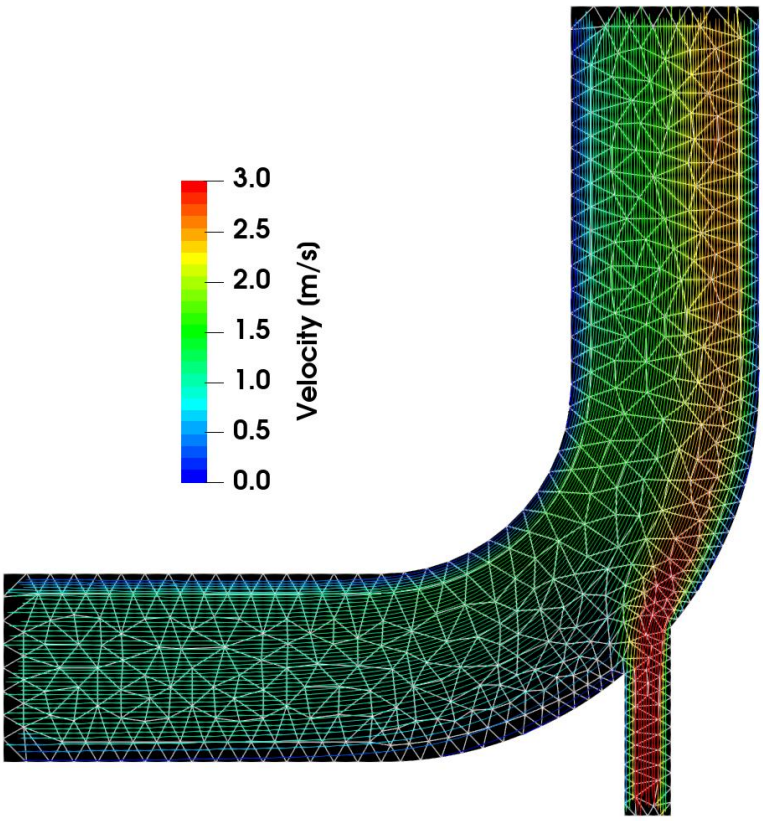
Tutorial One

Basic Case Setup



Bahram Haddadi

ENGINEERING FLUID DYNAMICS

7th edition, March 2025

Contributors:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen
- Vikram Natarajan
- Jozsef Nagy

TU WIEN ICEBE IMAGINEERING NATURE

Technische Universität Wien

Institute of Chemical, Environmental

& Bioscience Engineering

Available from: www.fluiddynamics.at

Background

1. What is CFD?

Computational Fluid Dynamics (CFD) is a method used to analyze systems involving fluid flow, heat transfer, and related phenomena such as heat and mass transfer. This analysis is performed through computer-based simulations, which help in understanding how fluids behave under different conditions. CFD is a powerful tool used in a variety of fields, including aerospace, automotive, chemical engineering, environmental studies, and biomedical applications.

The goal of CFD development is to create tools that are as reliable as other computer-aided engineering (CAE) methods like stress analysis. However, CFD is trickier due to the complex nature of fluid flow, which involves turbulence, variable properties, and nonlinear behavior. The mathematical foundation of CFD is based on the Navier-Stokes and continuity equations, which describe the motion of fluid substances and are derived from fundamental conservation laws (mass and momentum). These equations are partial differential equations that represent how fluid velocity, pressure, and density change over time and space.

While CFD offers many advantages, such as cost reduction in experimental setups and the ability to simulate complex scenarios, it is not fully automated. A good understanding of the underlying physics is necessary to set up a reliable simulation and interpret results correctly. Additionally, even with advanced computational resources, real-time simulations are still challenging due to the intensive calculations required. CFD is typically used alongside experimental methods like wind tunnel testing to validate and improve results.

CFD software comes in two main types:

- **Open-source and free (e.g., OpenFOAM®):** Offers flexibility for modification and customization, making it popular in academic and research environments.
- **Commercial and closed source (e.g., ANSYS Fluent, COMSOL):** Provides user-friendly interfaces, technical support, and advanced features, making it suitable for industrial applications.

In this guide, the focus will be on OpenFOAM®, an open-source CFD software written in C++. It allows users to access, modify, and even develop custom solvers to meet specific research or industrial needs. OpenFOAM® is widely used due to its flexibility and extensive documentation, although it requires a good understanding of both CFD principles and programming basics.

For beginners, it's helpful to think of CFD as a "virtual wind tunnel" where you can simulate fluid flow without physically building models or conducting real-world experiments. This makes it a cost-effective and versatile tool, especially during the design and testing phases of engineering projects.

CFD is not only limited to air and water flow simulations, and it is extensively used in modeling different sophisticated processes such as: weather patterns (meteorology), blood flow in arteries (biomedical engineering), combustion

processes in engines (mechanical engineering), pollution dispersion in the atmosphere (environmental engineering) and many more!

2. Workflow of CFD

A typical CFD workflow consists of three main stages:

2.1 Pre-processing

This stage involves setting up the simulation, including:

- **Geometry Definition:** Creating the computational domain that represents the physical system. This can be done using CAD (Computer-Aided Design) software or built-in geometry tools in CFD software. The accuracy of geometry affects how well simulation represents the real-world scenario. Think of geometry as the "shape" or "structure" through which the fluid will flow. Beginners can start with simple geometries like pipes, ducts, or channels before progressing to complex designs.
- **Mesh Generation:** Dividing the domain into smaller, non-overlapping elements (cells) to form a grid. The quality and density of the mesh significantly affect the accuracy of the simulation. Finer meshes are used in regions with high gradients (e.g., near walls, sharp edges, or around obstacles), while coarser meshes suffice for uniform flow areas. Meshes can be structured (regular grids) or unstructured (irregular shapes), depending on the complexity of the geometry. A structured mesh is easier to generate and solve but less flexible for complex geometries, while an unstructured mesh can fit intricate shapes better. Imagine the mesh as a net spread over your geometry. The tighter the net (finer mesh), the more detailed your simulation results will be. However, this also increases computational effort, so there's a balance to be found.
- **Model Selection:** Choosing physical models to represent phenomena such as turbulence (e.g., k-epsilon, k-omega models), heat transfer, and chemical reactions. The selection depends on the flow regime (laminar or turbulent) and the specific application.
- **Fluid Properties:** Defining parameters like density, viscosity, thermal conductivity, and specific heat capacity. These properties vary with temperature, pressure, or composition in complex simulations. Incompressible flow assumes constant density, while compressible flow accounts for changes in density due to pressure and temperature variations.
- **Boundary and Initial Conditions:** Setting conditions at the domain's boundaries (e.g., velocity at an inlet, pressure at an outlet, wall conditions) and initial conditions for transient simulations. Proper boundary conditions are crucial for accurate results.

The solution variables (e.g., velocity, pressure, temperature) are calculated at specific points within each cell. The mesh's resolution influences the

simulation's accuracy and computational cost. A mesh independence study is often performed to ensure that the results are not sensitive to the mesh size. This involves running simulations with progressively finer meshes until the changes in results become negligible.

Tip for beginners: Start with a coarser mesh to get quick results, then gradually refine the mesh to see how it affects accuracy. This helps you learn how sensitive your simulation is to mesh density.

2.2 Solver

In this stage, numerical methods are applied to solve the governing equations of fluid flow, including:

- **Conservation Equations:** Mass, momentum, and energy conservation laws are integrated over each control volume. These equations are often coupled, meaning changes in one variable affect others. For example, changes in velocity can influence pressure and vice versa**.**
- **Discretization:** The continuous equations are converted into algebraic forms using methods like the finite volume method (FVM), which ensures conservation principles are maintained within each cell. Other methods include the finite difference method (FDM) and finite element method (FEM), though FVM is most common in CFD. Discretization involves approximating derivatives with algebraic expressions, allowing the equations to be solved numerically.
- **Solution Techniques:** The resulting algebraic equations are solved iteratively until convergence is achieved. Common iterative solvers include the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) and PISO (Pressure-Implicit with Splitting of Operators) algorithms. Convergence is determined when changes in the solution between iterations fall below a predefined threshold.

The finite volume method is widely used because it ensures the conservation of physical quantities within each control volume, making it both accurate and robust. It is also flexible for handling complex geometries and boundary conditions.

Think of the solver as the "engine" of CFD—it's where all the heavy lifting happens to calculate how the fluid moves. Understanding how the solver works helps in troubleshooting and optimizing simulations.

2.3 Post-processing

This stage involves analyzing and visualizing the simulation results. Key tasks include:

- **Visualization:** Using cutting planes, contour plots, vector fields, streamlines, and line plots to represent flow variables such as velocity, pressure, and temperature distributions. Visualization helps in identifying flow patterns, vortices, and areas of interest like high-pressure zones.

- **Data Analysis:** Evaluating physical quantities like forces (drag, lift), heat transfer rates, pressure drops, and flow rates. Quantitative analysis helps validate the simulation results against experimental data or theoretical predictions.
- **Validation:** Comparing simulation results with experimental data or theoretical models to ensure accuracy. Sensitivity analysis may be conducted to understand the influence of different parameters.

Popular post-processing tools include commercial software like TecPlot and Ensight, as well as open-source tools such as ParaView and SALOME. These tools allow for advanced visualization techniques, including 3D rendering and time-dependent animations, making it easier to interpret complex flow behaviors.

Tip: Post-processing is not just about making pretty pictures. It helps you understand the flow physics and detect any errors or inconsistencies in your simulation.

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® (v12), 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/legacy/incompressible/icoFoam/elbow
```

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

Note: The tutorial can also be simply copied from the mentioned directory using your file explorer.

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` (which is not the case in this tutorial):

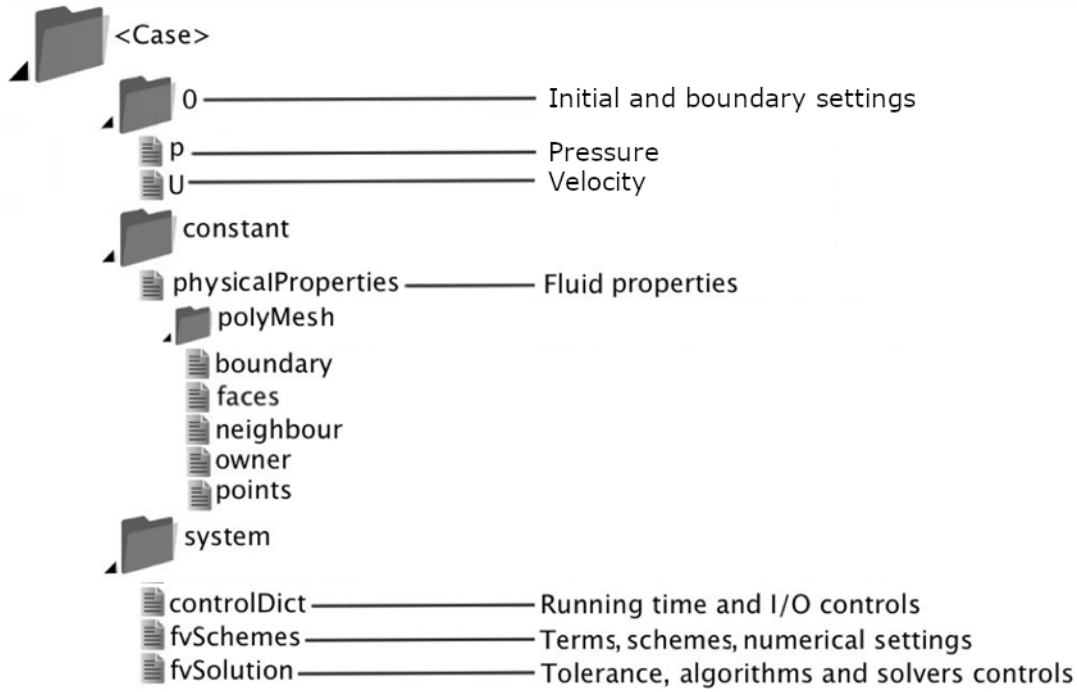
```
>fluentMeshToFoam elbow.msh -scale 0.001
```

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 and boundary 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

p file includes pressure initial and boundary 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
```

Note: nano is the command line based text editor, which comes by default with Ubuntu. You can use any other text editor (also graphical ones) for opening and editing the files.

Note: You can use ctrl+x for closing and exiting the nano.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Website:  https://openfoam.org                  |
|   \\  /    A nd           | Version:  12                                    |
|    \\/     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: In the dimension matrix the first number represents mass (kilogram), the second one the length (meter), the third one time (second), the fourth one the temperature (Kelvin), the fifth one the quantity (mole), the sixth one current (ampere) and the last one luminous intensity (candela).

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

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 cells, 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 many different types of boundary conditions in OpenFOAM®, a few very common ones:

- `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 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 of the velocity. 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 the mesh subdirectory and some files. In the sub-directory “polyMesh” the mesh data are stored (in this case the data for imported mesh). Among the files in the polyMesh directory, the boundary file is relevant for users and includes the mesh boundary data, e.g. name, type and the patch group which can be modified by the user for changing the boundary type or name for a created or imported mesh (for the sake of space, the dictionary headers will not be included in this scope anymore):

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

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 and types with the ones set in GAMBIT®, they should be the same.

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.

The files in the constant directory (usually) include material properties, simulation physics and chemistry, e.g. 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 (it will be discussed in more detail in the next tutorials).
- **fvSolution**: Contains the settings to the coupling method of pressure and velocity, the numerical methods, which are used for solving different quantities, and 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.

Note: Our first modification in the simulation is changing the `endTime` from the original value of 10s to 75s, for running the simulation up to 75s.

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:

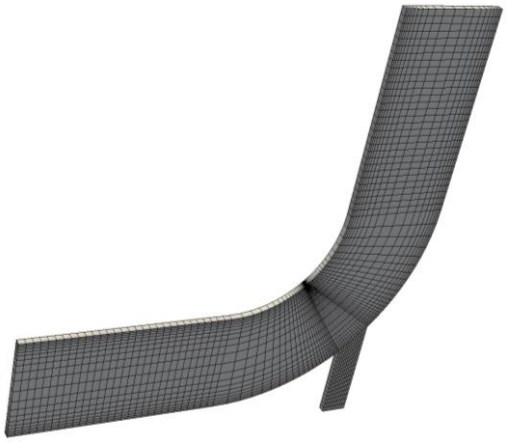
```
>paraview foam.foam &
```

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

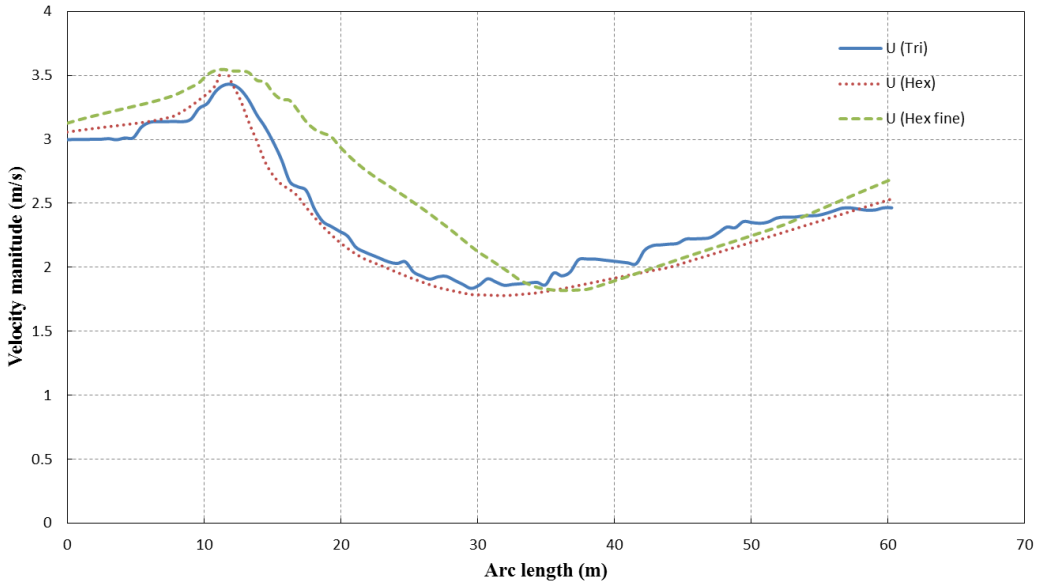
3.2. Examining different meshes

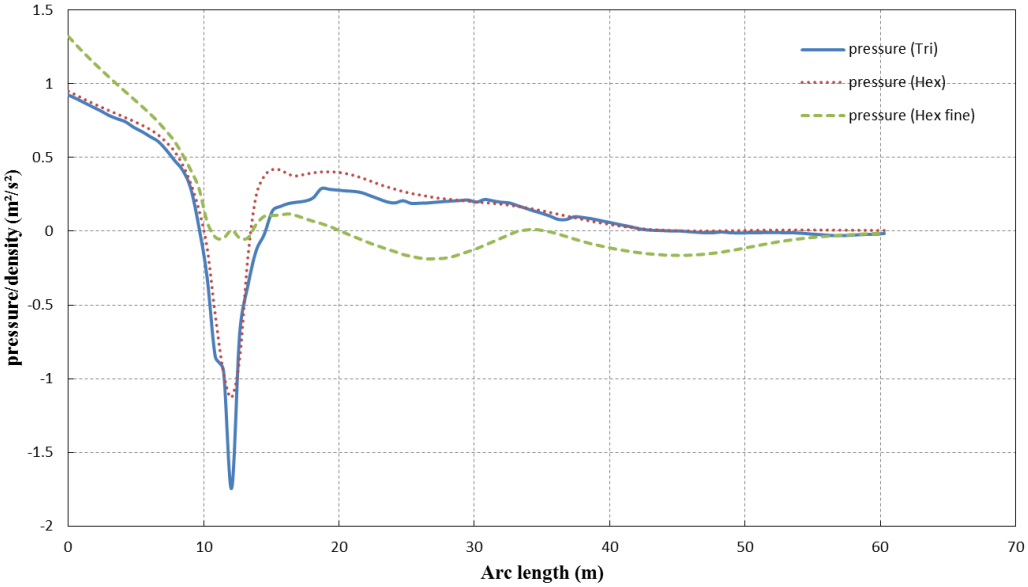
Do the same for the other two meshes. Only the mesh for the first simulation is included in the elbow example of OpenFOAM®. For the other two simulations, the mesh should be provided by the user. For finding the tutorials on how to create the geometry and the mesh, search the internet for "GAMBIT® elbow 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.



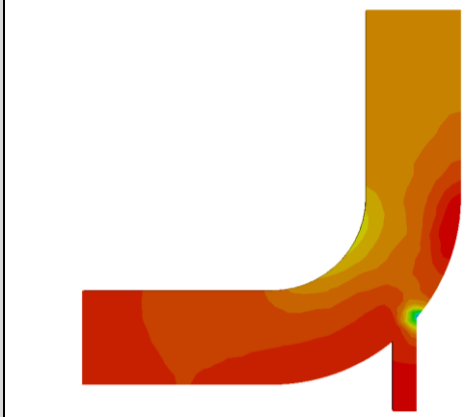
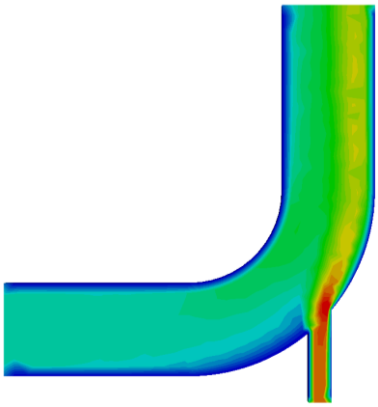
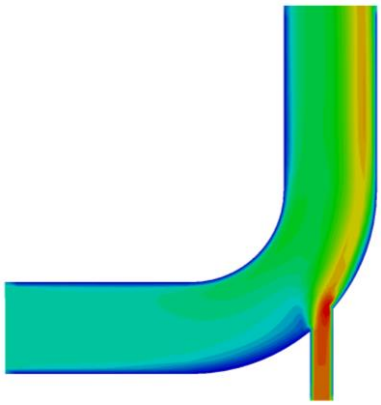
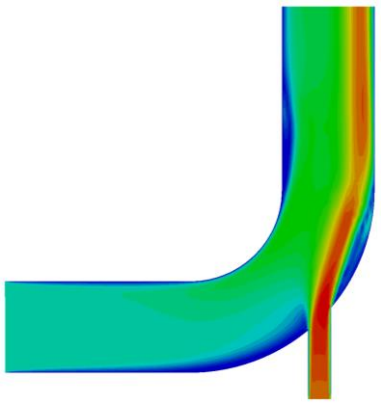
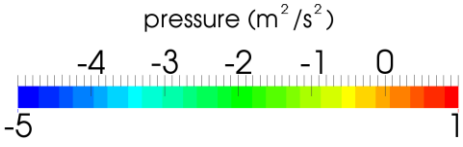
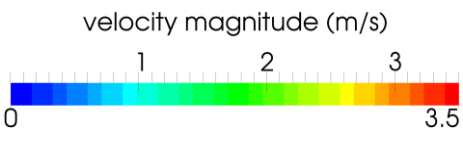
The Hex Fine mesh





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.