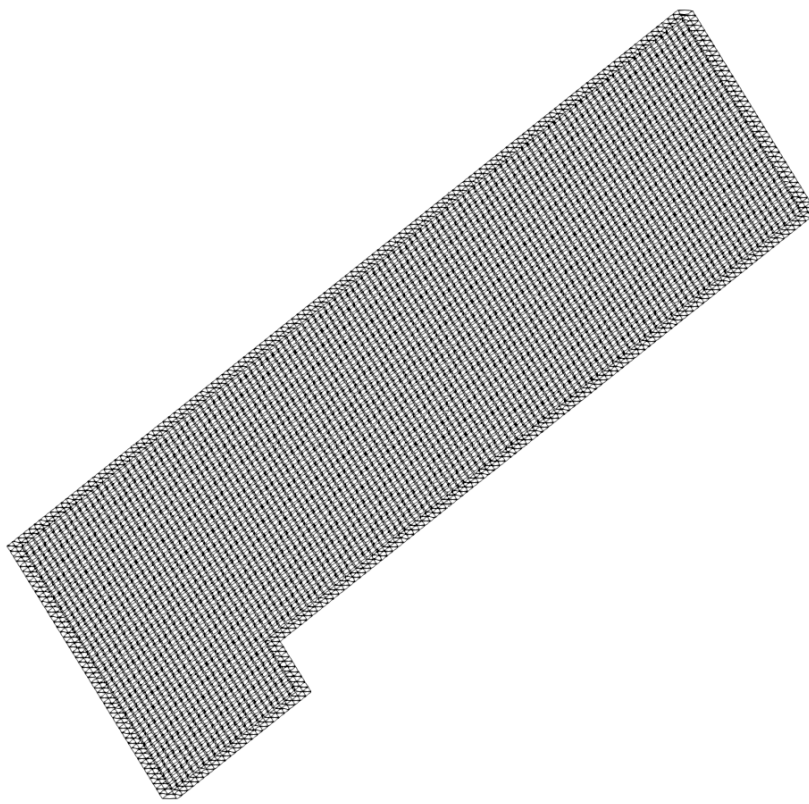


Tutorial Two


Built in Mesh



Bahram Haddadi



7th edition, March 2025

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Contributors:

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



Technische Universität Wien
Institute of Chemical, Environmental
& Bioscience Engineering



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that, they endorse you or your use of the work).
- Noncommercial — you may not use this work for commercial purposes.
- Share Alike — if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- Waiver — any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain — where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights — In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
 - The author's moral rights;
 - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice — for any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This offering is not approved or endorsed by ESI® Group, ESI-OpenCFD® or the OpenFOAM® Foundation, the producer of the OpenFOAM® software and owner of the OpenFOAM® trademark.

Available from: www.fluiddynamics.at

Background

1. What is Mesh?

When studying fluid flow and heat transfer, mathematical equations known as partial differential equations (PDEs) describe how physical properties such as mass, energy, and momentum change over space and time. However, solving these equations directly (analytically) is extremely difficult unless the problem is very simple.

To solve PDEs numerically, these equations are discretized and converted from a set of PDEs to a set of algebraic equations. This involves breaking the entire fluid domain into many smaller, manageable sections. These small sections are called grid cells, and together they form a mesh.

A mesh is like a net or grid that covers the entire area where fluid behavior is analyzed. The finer (smaller) the mesh, the more accurately flow details can be captured, but at the cost of increased computational demand. Choosing the right mesh ensures a balance between accuracy and efficiency in simulations.

One of the most common numerical methods for solving these equations is the finite volume method (FVM), which is explained below.

2. The Finite Volume Method (FVM)

OpenFOAM® applies the finite volume method (FVM) to simulate fluid flow. This method works by applying a key equation called the transport equation, which describes how physical property (such as velocity, temperature, or pressure) moves through a fluid domain over time. The general transport equation is:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\phi\mathbf{u}) = \nabla \cdot (\Gamma\nabla\phi) + S_\phi$$

Rate of change of ϕ inside fluid element	+	Net rate of flow of ϕ out of fluid element	=	Rate of change of ϕ due to diffusion	+	Rate of change of ϕ due to sources
-----------------------------------------------------	---	-------------------------------------------------------	---	----------------------------------------------	---	--------------------------------------------

The finite volume method works by applying and integrating this equation over a control volume (CV), which is a small section of the mesh. A mathematical technique called the Gauss divergence theorem helps converting volume integral terms in the equation into surface integrals. This allows for the calculation of the amount of a property entering and exiting each grid cell, ensuring that all properties are conserved throughout the simulation.

$$\begin{aligned} \int_{\Delta t} \frac{\partial}{\partial t} \left(\int_{CV} \rho\phi dV \right) dt + \int_{\Delta t} \int_A \mathbf{n} \cdot (\rho\phi\mathbf{u}) dAdt \\ = \int_{\Delta t} \int_A \mathbf{n} \cdot (\Gamma\nabla\phi) dAdt + \int_{\Delta t} \int_{CV} S_\phi dV dt \end{aligned}$$

For time-dependent problems, the equation must also be integrated over a small time step (Δt) to account for changes in properties over time. This

step-by-step approach is essential for accurately capturing transient behaviors, such as turbulence or shock waves.

3. Discretization of Transport Equations

Discretization of the transport equations is critical to the finite volume method and is done using the mesh, which involves dividing the domain into smaller regions.

In CFD, the meshes can be divided into two main categories:

- **Structured meshes:** These are arranged in a regular, grid-like pattern, often using Cartesian coordinates (X, Y, Z directions). They are simple to use but may not work well for complex geometries.
- **Unstructured meshes:** These use irregularly shaped grid cells and can represent complex shapes, such as curved surfaces and complex objects, more accurately.

Mesh generation in OpenFOAM® is done using built-in tools such as blockMesh (for structured meshes) and snappyHexMesh (for unstructured meshes). External software like GAMBIT® can also be used for creating meshes. This tutorial focuses on using blockMesh, which provides a simple way to generate structured grids. More advanced mesh generation using snappyHexMesh is covered in Tutorial Twelve.

4. foamRun Solver – fluid module

In OpenFOAM® 12, the foamRun application serves as a versatile tool for executing various solver modules. Unlike traditional/legacy solvers (e.g. icoFoam) that are specific to certain types of simulations, foamRun dynamically loads and runs a solver module which can be either defined in the simulation setup or as a command-line argument. This modular approach enhances flexibility, allowing users to select appropriate solver modules for their specific simulation needs.

“fluid” is the solver module for steady or transient turbulent flow of compressible fluids with heat-transfer with optional mesh motion and change.

fluid Solver – forwardStep

Tutorial outline

Using foamRun and fluid solver, simulate 10 s of flow over a forward step.

Objectives

- Understand blockMesh
- Define vertices via coordinates as well as surfaces and volumes via vertices.

Data processing

Import your simulation into ParaView, and examine the mesh and the results in detail.

1. Pre-processing

1.1. Copying tutorial

Copy the tutorial from the following folder to your working directory:

```
$FOAM_TUTORIALS/fluid/forwardStep
```

1.2. Case structure

1.2.1. 0 directory

There are two new files in the 0 folder, T and Ma. File T includes the initial temperature values and Ma is the Mach number values which are calculated using the OpenFOAM® function objects (this can be ignored for this tutorial). Internal pressure and temperature fields are set to 1, and the initial velocity in the domain as well as the inlet boundary is set to (3 0 0).

Note: As it can be seen, the p unit is the same as the pressure unit ($\text{kg m}^{-1} \text{s}^{-2}$), because fluid module is for compressible fluids.

Note: Do not forget that, this example is a purely numeric example (you might have noticed this from the pressure values).

1.2.2. constant directory

By checking *physicalProperties* file, different properties of a compressible gas can be set:

```
// * * * * *
* * * * *//
thermoType
{
    type            hePsiThermo;
    mixture          pureMixture;
    transport        const;
    thermo           hConst;
    equationOfState  perfectGas;
    specie           specie;
    energy           sensibleInternalEnergy;
}
// Note: these are the properties for a "normalized" inviscid gas
//       for which the speed of sound is 1 m/s at a temperature of 1K
//       and gamma = 7/5
mixture
{
    specie
    {
        molWeight    11640.3;
    }
    thermodynamics
    {
        Cp           2.5;
        Hf           0;
    }
    transport
    {
        mu           0;
        Pr           1;
    }
}
// * * * * *
* * * * *//
```

In the `thermoType`, the models for calculating thermo physical properties of gas are set:

- `type`: Specifies the underlying thermos-physical model, which in this case is enthalpy based thermodynamics while incorporating the equation of state using `psi` (compressibility)
- `mixture`: Is the model, which is used for the mixture, whether it is a pure mixture, a homogeneous mixture, a reacting mixture or
- `transport`: Defines the transport model used. In this example a constant value is used for viscosity.
- `thermo`: It defines the method for calculating heat capacities, e.g. in this example constant heat capacities are used.
- `equationOfState`: Shows the relation which is used for the compressibility of gases. Here ideal gas model is applied by selecting `perfectGas`.
- `energy`: This key word lets the solver decide which type of energy equation it should solve enthalpy or internal energy.

After defining the models for different thermos-physical properties of gas, the constants and coefficients of each model are defined in the sub-dictionary `mixture`. E.g. `molWeight` shows the molecular weight of gas, `Cp` stands for heat capacity, `Hf` is the heat of fusion, `mu` is the dynamic viscosity and `Pr` shows the Prandtl number.

By opening the `momentumTransport` the appropriate turbulent mode can be set (in this case it is laminar):

```
simulationType    laminar;
```

1.2.3. system directory

In this tutorial the mesh is not imported from other programs (e.g. GAMBIT®). It will be created inside OpenFOAM®. For this purpose the `blockMesh` tool is used. `blockMesh` reads the geometry and mesh properties from the `blockMeshDict` file (found in the system directory):

```
>nano blockMeshDict
```

```
// * * * * *
* * * * *
convertToMeters 1;
vertices
(
    (0 0 -0.05)
    (0.6 0 -0.05)
    (0 0.2 -0.05)
    (0.6 0.2 -0.05)
    (3 0.2 -0.05)
    (0 1 -0.05)
    (0.6 1 -0.05)
    (3 1 -0.05)
    (0 0 0.05)
    (0.6 0 0.05)
    (0 0.2 0.05)
    (0.6 0.2 0.05)
    (3 0.2 0.05)
)
```

```

(0 1 0.05)
(0.6 1 0.05)
(3 1 0.05)
);
blocks
(
    hex (0 1 3 2 8 9 11 10) (25 10 1) simpleGrading (1 1 1)
    hex (2 3 6 5 10 11 14 13) (25 40 1) simpleGrading (1 1 1)
    hex (3 4 7 6 11 12 15 14) (100 40 1) simpleGrading (1 1 1)
);
defaultPatch
{
    type empty;
}
boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 8 10 2)
            (2 10 13 5)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (4 7 15 12)
        );
    }
    bottom
    {
        type symmetryPlane;
        faces
        (
            (0 1 9 8)
        );
    }
    top
    {
        type symmetryPlane;
        faces
        (
            (5 13 14 6)
            (6 14 15 7)
        );
    }
    obstacle
    {
        type patch;
        faces
        (
            (1 3 11 9)
            (3 4 12 11)
        );
    }
);

// * * * * *
* * * * *//

```

As noted before units in OpenFOAM® are SI units. If the vertex coordinates differ from SI, they can be converted with the `convertToMeters` command. The number in the front of `convertToMeters` shows the constant, which should be

multiplied with the dimensions to change them to meter (SI unit of length). For example:

```
convertToMeters    0.001;
```

shows that the dimensions are in millimeter, and by multiplying them by 0.001 they are converted into meters.

In the `vertices` part, the coordinates of the geometry vertices are defined, the vertices are stored and numbered from zero, e.g. vertex `(0 0 -0.05)` is numbered zero, and vertex `(0.6 1 -0.05)` points to number 6.

Note: In OpenFOAM® (and C++) counting starts from 0 and not 1!

In the `block` part, blocks are defined. The array of numbers in front each block shows the block building vertices, e.g. the first block is made of vertices `(0 1 3 2 8 9 11 10)`.

After each block, the mesh is defined in every direction. e.g. `(25 10 1)` shows that this block is divided into:

- 25 parts in x direction
- 10 parts in y direction
- 1 part in z direction

As was explained in tutorial 1, even for 2D simulations the mesh and geometry should be 3D, but with one cell in the direction, which is not going to be solved, e.g. here number of cells in z direction is one and it's because of that it's a 2D simulation in x-y plane.

The last part, `simpleGrading(1 1 1)` shows the size function, in this case 1 means there is no change in the cell size from one cell to another

In the `boundary` part, each boundary is defined by the vertices it is made of, and its `type` and `name` are defined.

Note: For creating a face, the vertices should be chosen clockwise when looking at the face from inside of the geometry.

2. Running simulation

Before running the simulation, the mesh has to be created. In the previous step, the mesh and the geometry data were set. For creating it, the following command should be executed from the case main directory (e.g. `forwardStep`):

```
>blockMesh
```

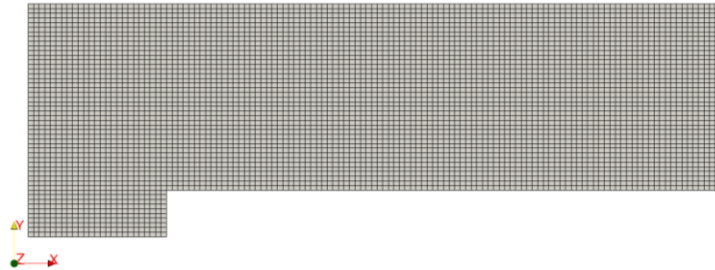
After that, the mesh is created in the `constant/polyMesh` folder. For running the simulation, type the solver name from case directory and execute it:

```
>foamRun -solver fluid
```

Note: The solver can be also defined in the `controlDict` (which is the case here) and then the simulation can be performed simply using `foamRun` command without the solver flag.

3. Post-processing

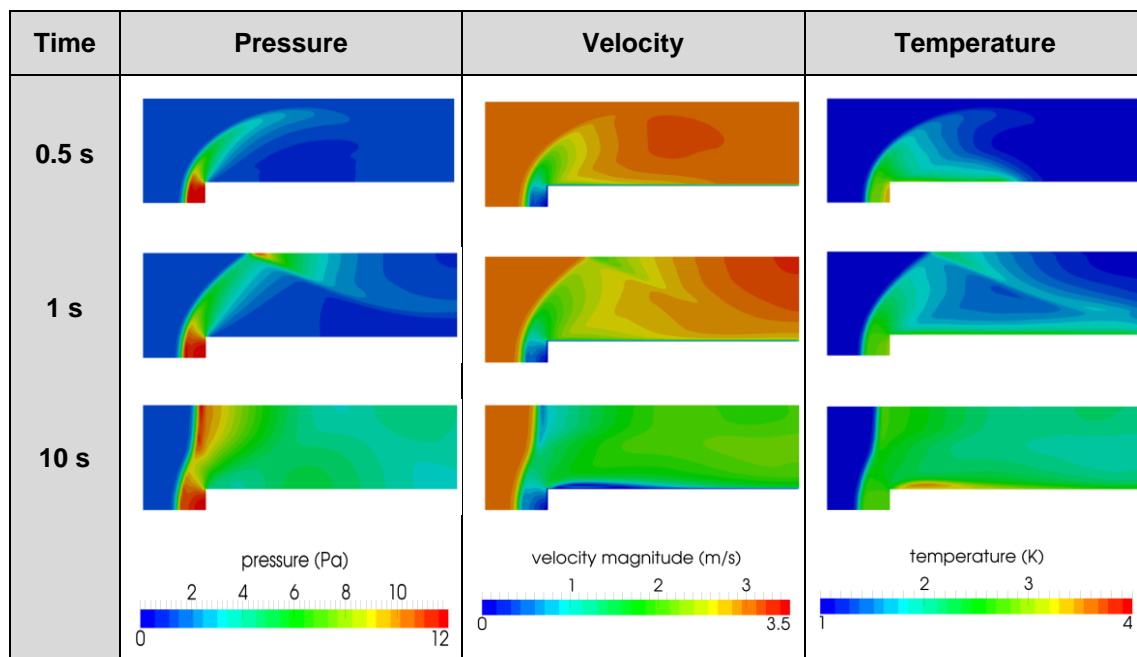
The mesh is presented in the following way in ParaView, and you can easily see the three blocks, which were created.



Mesh generated by blockMesh

Note: When a cut is created by default in ParaView, the program shows the mesh on that plane as a triangular mesh even if it is a hex mesh. In fact, ParaView changes the mesh to a triangular mesh for visualization, where every square is represented by two triangles. For avoiding this when creating a cut in ParaView in the Slice properties window, uncheck “Triangulate the Slice”.

The simulation results are as follows:



Pressure, velocity and temperature contours at different time steps