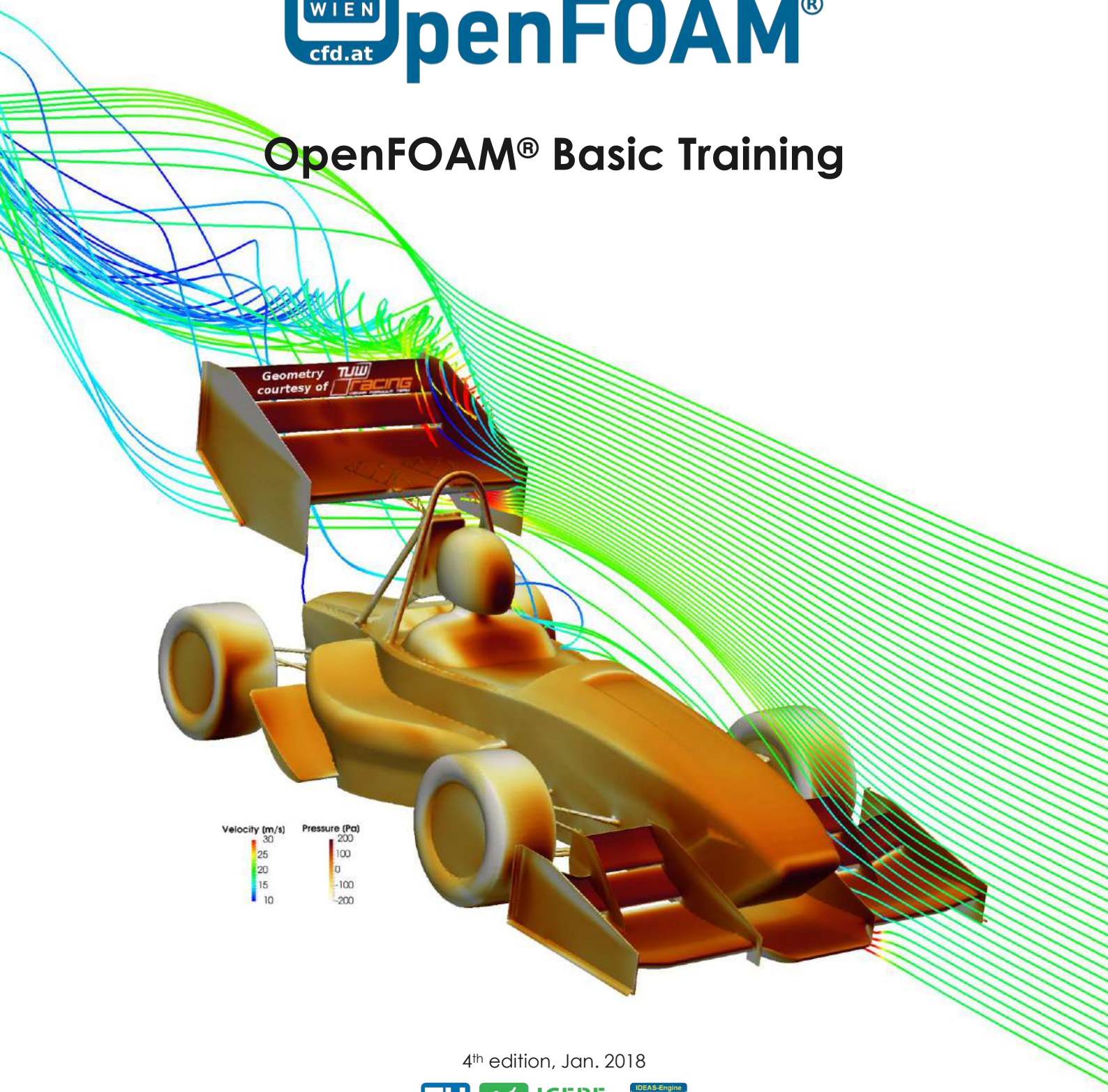
TU WARDS OpenFOAM®

OpenFOAM® Basic Training



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Technische Universität Wien
Institute of Chemical, Environmental
& Bioscience Engineering

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:

- Bahram Haddadi. Special thanks to Philipp Schretter and TUW-Racing.

This book has been used as a basis for preparing a series of video lectures
on youtube by Jozsef Nagy (JKU Linz):
www.youtube.com/channel/UCjdgpuxuAxH9BqheyE82Vvw
(Search for: Jozsef Nagy OpenFOAM at youtube.com)

In this OpenFOAM® tutorial series, we have prepared fourteen case examples that are designed to help users to learn the key utilities and features within OpenFOAM®, including mesh generation, multiphase modeling, turbulence modeling, parallel processing and reaction modeling. The base tutorial examples can be imported directly from the OpenFOAM® installation directory.

The tutorials should be primarily used for OpenFOAM® versions 5.0 and v1712, with differences in the running procedure between v1712 and 5.0 highlighted in blue boxes. So, simply ignore the blue boxes if you are running in version 5.0! The structure of each case example follow the below general structure:

- **Background**: an introduction about the key topics explored in the tutorial and the relevant CFD theory

1. **Pre-processing**: instructions on how to set up the correct case structure for a given problem using base case tutorials, with explanations on relevant dictionaries
2. **Running simulation**: instructions on running the solver and its associated commands
3. **Post-processing**: examining the results in OpenFOAM®'s post-processing application, ParaView V5.4.0

Tutorial One: **Basic Case Setup**

Solver: icoFoam
Geometry: 2-dimensional
Tutorial: elbow

Tutorial Two: **Built in Mesh**

Solver: sonicFoam
Geometry: 2-dimensional
Tutorial: forwardStep

Tutorial Three: **Patching Fields**

Solver: sonicFoam
Geometry: 1-dimensional
Tutorial: shockTube

Tutorial Four: **Discretization – Part 1**

Solver: scalarTransportFoam
Geometry: 1-dimensional
Tutorial: shockTube

Tutorial Five: **Discretization – Part 2**

Solver: scalarTransportFoam
Geometry: 2-dimensional
Tutorial: circle

Tutorial Six: **Turbulence, Steady state**

Solver: simpleFoam
Geometry: 2-dimensional
Tutorial: pitzDaily

Tutorial Seven: **Turbulence, Transient**

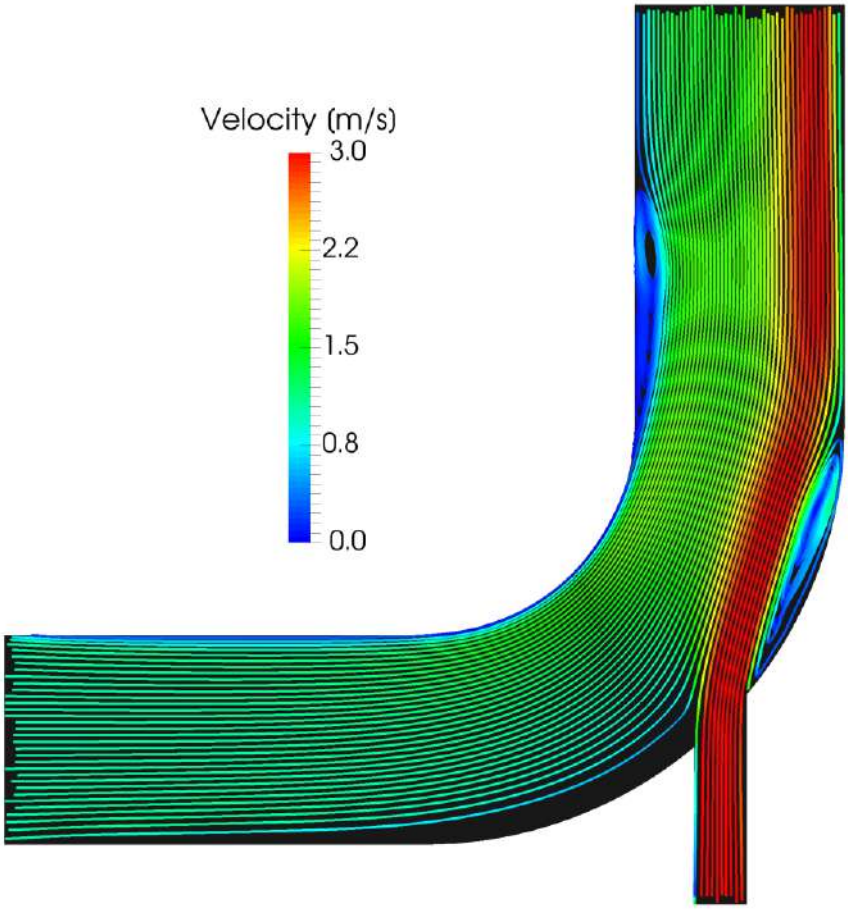
Solver: pisoFoam
Geometry: 2-dimensional
Tutorial: pitzDaily

Tutorial Eight: **Multiphase**

Solver: interFoam
Geometry: 2-dimensional
Tutorial: damBreak

Tutorial One

Basic Case Setup



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.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 lots of 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. So, 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:

1. Integration of the conservation of mass, energy and momentum equations over all the control volumes in the domain
2. Discretization – conversion of the resulting integral equations into a system of algebraic equations
3. 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 take a 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® (e.g. V5.0 and v1712), 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[1] 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 also a short description on how to use it:

```
Usage: fluentMeshToFoam [OPTIONS] <Fluent mesh file>
options:
  -case <dir>       specify alternate case directory, default is the cwd
  -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-5.0 (see www.OpenFOAM.org)
Build: 5.0
```

[1] International System of Units

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`:

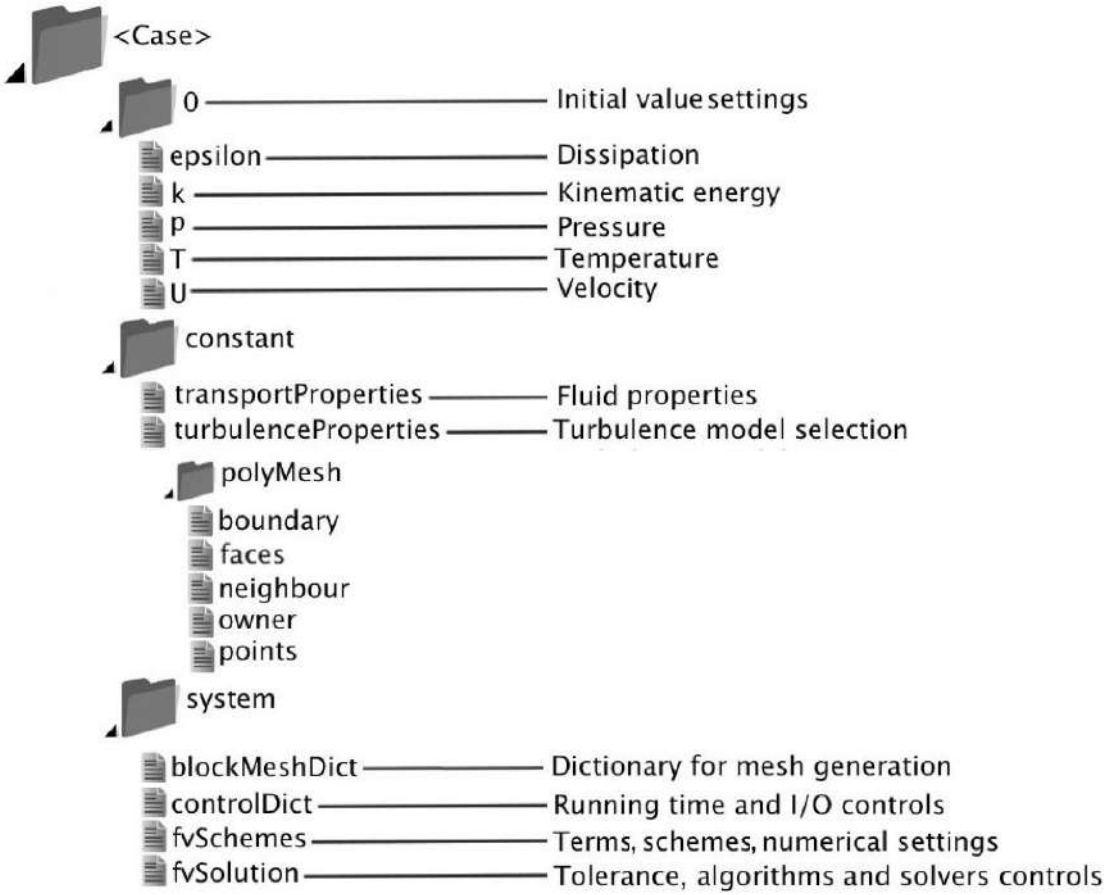
```
>fluentMeshToFoam elbow.msh -scale 1.0
```

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 also 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 foloder:

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[1] p
```

It will be like this:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    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;
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
```

[1] nano is a text editor used in Linux OS (for closing and saving: ctrl+x)

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 "reduced" 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        1(wall)
        nFaces          100;
        startFace       1300;
    }
    velocity-inlet-5
    {
        type            patch;
        nFaces          8;
        startFace       1400;
    }
…
    frontAndBackPlanes
    {
        type            empty;
        inGroups        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 also 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 converting the mesh to OpenFOAM® mesh, modify that boundary in the file constant/polyMesh/ boundary, and change its* `type` *from* `wall` *to* `empty`, *and also 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 transportProperties file, properties dimensions and also the property value can be found and edited (in v1712 no dimensions are listed), 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.

> OpenFOAM® v1712: Just nu and its value are listed – no dimensions!

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 which is 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;


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

Changed from 10 to 75 for 75 s of simulation

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: There is also 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 &
```

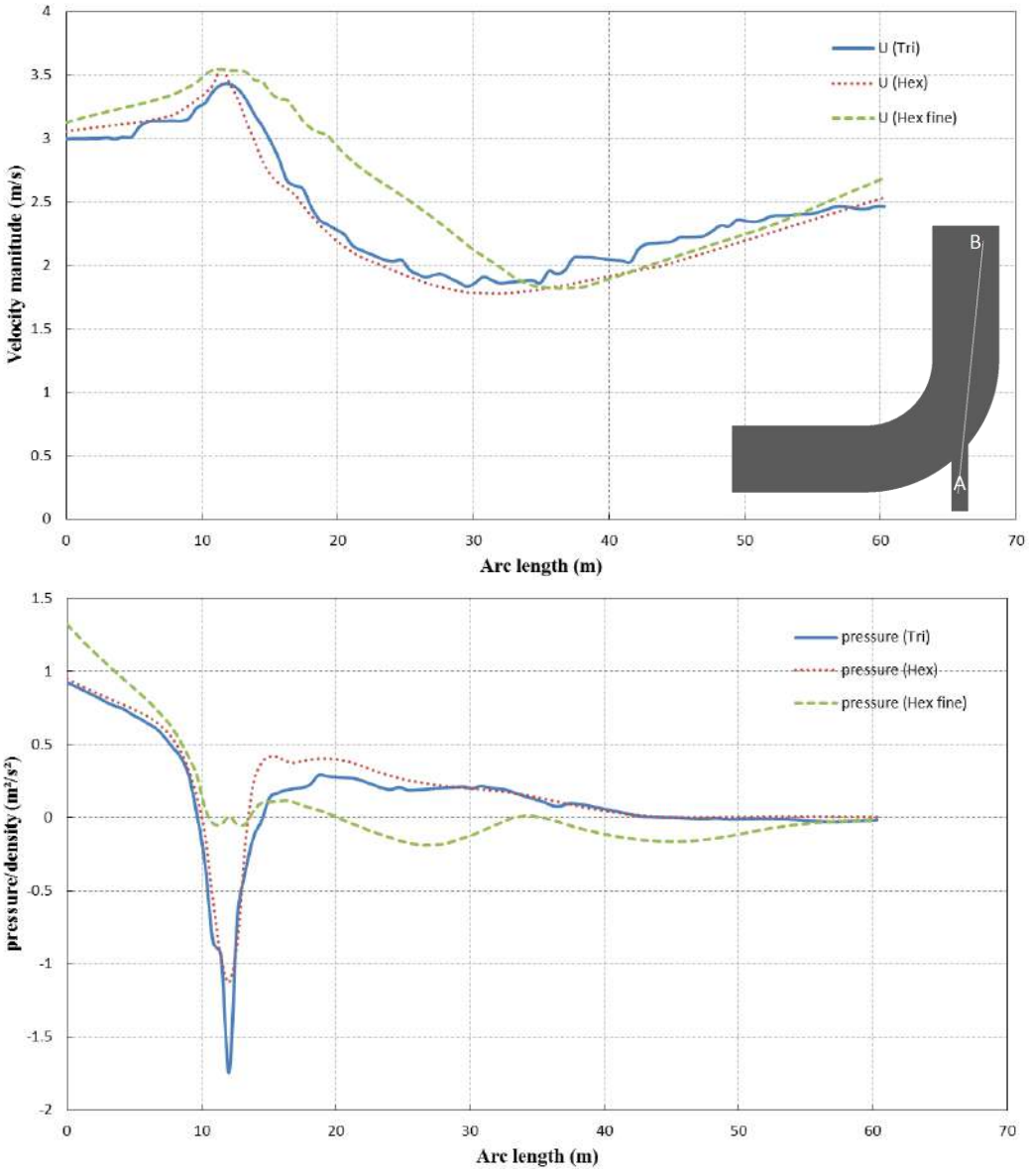
Note: By putting & at the end of command, the command line will remain active and ready for further inputs while that program is running.

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 also the mesh info are provided in that tutorial. Try to create it by using GAMBIT®. When you are done you have to convert it into a 3D mesh with 1 cell in the z-direction.

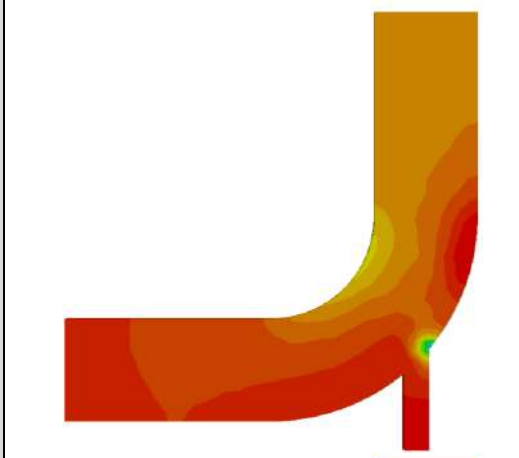
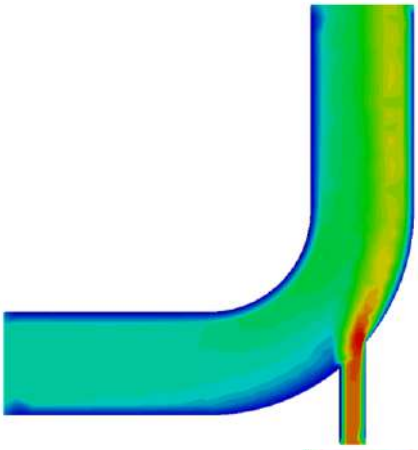
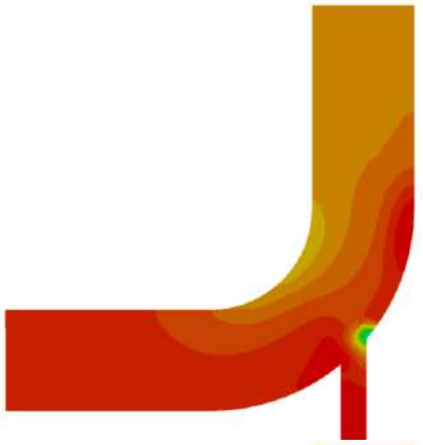
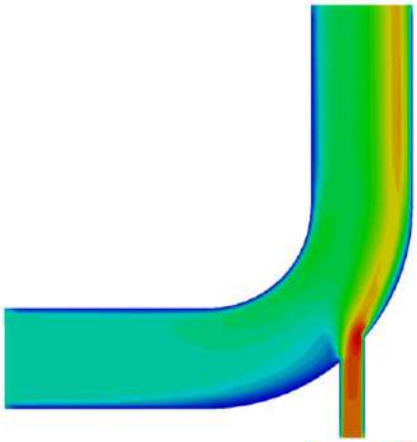
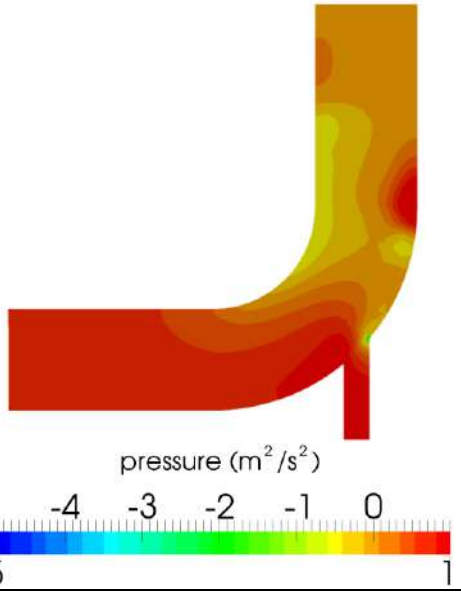
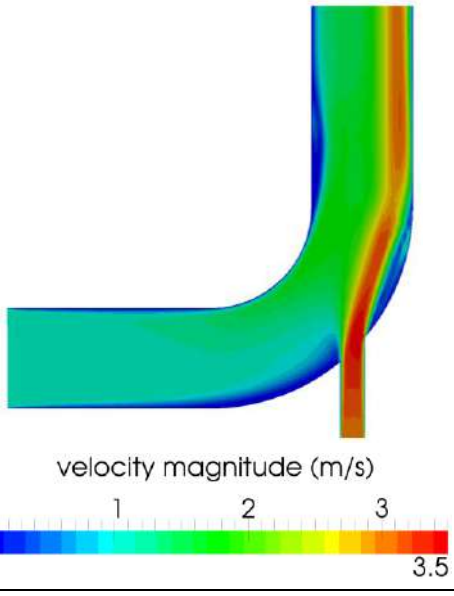
The comparisons of all three case results and charts are shown below.

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).

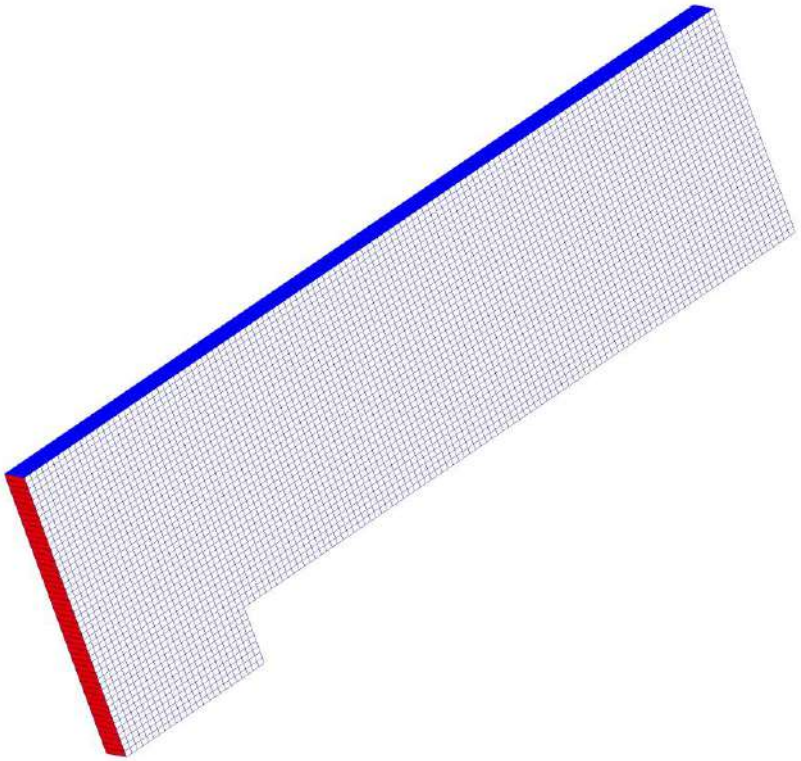
Mesh	Pressure	Velocity
Tri		
Hex		
Hex Fine		

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.

Tutorial Two

Built in Mesh



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. What is mesh?

The partial differential equations that describe fluid flow and heat transfer are the conservation equations of mass, energy and momentum. However we are usually unable to solve them analytically, except in very simple cases. This is when discretization comes in. The flow region is broken up into smaller sub-regions, with the equation solved in each sub-region. One of the methods used to solve the equations is the finite volume method, which we will cover in detail below. The sub-regions are later on referred to as grid cells, with a collection of grid cells forming a mesh.

2. Finite volume method

As discussed in Tutorial One, OpenFOAM® uses finite volume method. The starting point of this method is the transport equation for a conserved fluid property ϕ, for example pressure and flow velocity. It is shown below as:

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

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**

As you can see, the transport equation is essentially a manifestation of conservation of a fluid flow property within the problem domain.

The key step of the finite volume method is the integration of the transport equation over a three-dimensional control volume (CV). At discrete places values are calculated on a meshed geometry. The small volume which surrounds each node of the mesh is the grid cell.

In each grid cell, the volume integral of the divergence term is replaced by a surface integral, using the Gauss divergence theorem. These terms are then evaluated as fluxes at the surfaces. This not only ensures the conservation of fluxes entering and exiting the grid cell, but allows for easy formulation of the balances on unstructured meshes.

In time-dependent problems, it is also necessary to integrate with respect to time t over a small interval Δt from, say, t until $t + \Delta t$. This yields the most general integrated form of the transport equation.

$$\int_{\Delta t} \frac{\partial}{\partial t}\left(\int_{CV} \rho\varphi \, dV\right) dt + \int_{\Delta t} \int_A \boldsymbol{n} \cdot (\rho\varphi \boldsymbol{u}) \, dA dt = \int_{\Delta t} \int_A \boldsymbol{n} \cdot (\Gamma \nabla\varphi) \, dA dt + \int_{\Delta t} \int_{CV} S_\varphi dV \, dt$$

3. Discretization of transport equations

Discretization of the transport equations is critical to the finite volume method, as it provides a more cost-effective and rapid approach to numerical evaluation on digital computers. Discretization is done through the use of the mesh, which involves dividing the domain into smaller regions.

The mesh used in OpenFOAM® can be simple grid structures based on the Cartesian co-ordinate system, or complex unstructured grid arrangement that can handle curvature and geometric complexity. The mesh is generated using the in-house OpenFOAM® tool (*blockMesh* and *snappyHexMesh*) or external software, such as GAMBIT®. In this tutorial, we are going to learn how to use the *blockMesh* tool in OpenFOAM®. Refer to Tutorial Twelve for information on the *snappyHexMesh* tool.

For this tutorial, we chose to run the *sonicFoam* solver, which is slightly more complicated than *icoFoam*, as it analyses the flow of a compressible gas/fluid.

4. sonicFoam solver

sonicFoam is a transient solver. It solves trans-sonic/supersonic, turbulent flow of a compressible gas/fluid. Similar to icoFoam, sonicFoam is based on the PISO algorithm.

sonicFoam – forwardStep

Tutorial outline

Using sonicFoam 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/compressible/sonicFoam/laminar/forwardStep`

1.2. Case structure

1.2.1. 0 directory

The file T includes the initial temperature values. Internal pressure and temperature fields are set to 1, and the initial velocity in the domain is set to zero except at the inlet boundary, where it is 3.

Note: As it can be seen, the p unit is the same as the pressure unit (kg m^{-1} s^{-2}), because sonicFoam is a compressible solver.

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 *thermophysicalProperties* 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;
}
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.

- `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 used transport model. In this example a constant value is used.

- `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 turbulenceProperties 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)
);
edges
(
);
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)
        );
    }
);

mergePatchPairs
(
);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

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.

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 it was explained before, 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 the `boundary` part each boundary is defined by the vertices it is made of, and also 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):

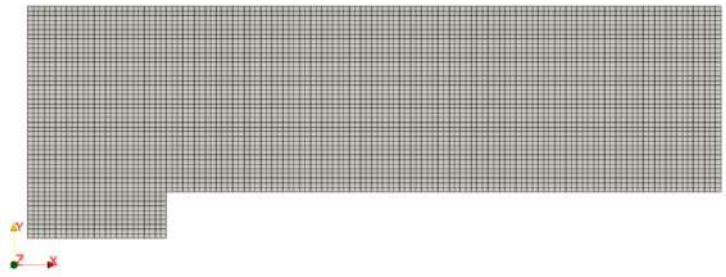
```
>blockMesh
```

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

```
>sonicFoam
```

3. Post-processing

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

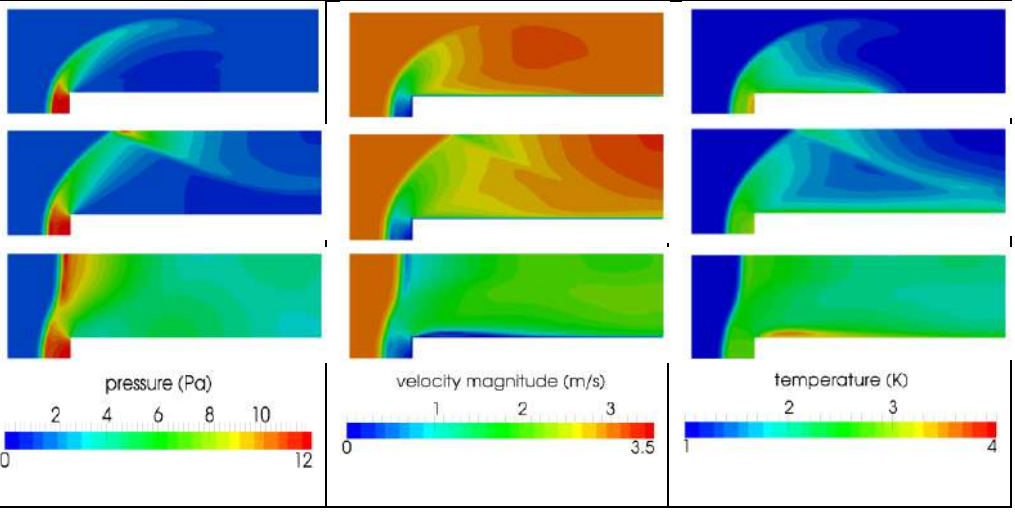


Mesh generated by blockMesh

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".

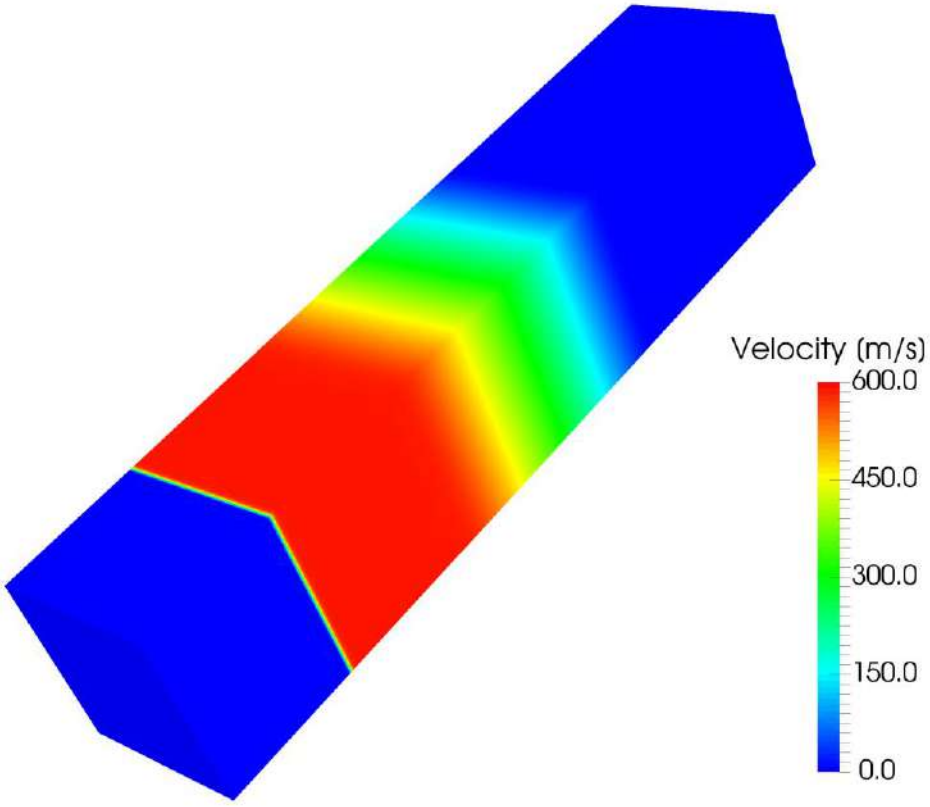
The simulation results are as follows:

Time	Pressure	Velocity	Temperature
0.5 s			
1 s			
10 s			

Pressure, velocity and temperature contours at different time steps

Tutorial Three

Patching Fields



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:

- Bahram Haddadi

Contributors:

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

For more tutorials visit: www.cfd.at

Background

1. Initial and Boundary Conditions

Before running the numerical solver, it is important to set up initial and boundary conditions correctly for the problem. Ill-defined boundary conditions will result in non-convergence or incorrect results.

The initial conditions provide the starting values for the solver and once specified, the value is put into the center of every cell in the whole domain. As the solver starts to run, the initial values will be replaced by newly calculated values. Any starting values can be used for 1st iteration. However the better the initial values, the faster the convergence. Initial conditions are mandatory for transient problems, but not absolutely required for steady state problems.

On the other hand we need to also provide boundary conditions. These will connect the simulation models with its surroundings. The values specified are located at the boundary faces of the domain, where their solution will be kept unchanged during the simulation, as the solver will not calculate them. Most boundary conditions are either steady state or transient.

In OpenFOAM®, we can use the *setFields* utility to specify a non-uniform initial condition, and this will be the focus of Tutorial Three. In addition, the boundary conditions are specified in the files in the 0 directory.

2. Courant-Friedrichs-Lewy (CFL) condition

In this tutorial, we will create meshes with 100, 1000 and 10000 cells in one dimension. However, one cannot simply increase the number of cells (i.e. decrease the cell size) without changing the time step size accordingly. This is because when running a numerical scheme (e.g. the Gauss linear scheme; more details can be found in Tutorials Four and Five), the solution is reached using the information propagated by waves or particles from one cell to the adjacent cell. To ensure a physical solution it is essential that the physical flow information e.g. velocity, composition, etc is received by all cells with in the calculation domain. It needs to be guaranteed that the information transport does not "overtake" the physical transport, otherwise the scheme will be unable to access the information required to form the solution.

The above criteria is known as the Courant-Friedrichs-Lewy (CFL) condition, and it is a necessary condition for convergence. For one-dimensional problems, it can be written as:

$$Co = \frac{u\Delta t}{\Delta x} \leq 1$$

The key dimensionless number here is the Courant number, Co, which needs to be less or equal to one.

Note: u is the velocity magnitude of compound in the 1D direction, Δt is the simulation time step size and Δx is the mesh size in the 1D direction.

As it is obvious from the equation by decreasing the mesh size (i.e. Δx), the time step size (Δt) should also be adjusted (decreased) for reaching a stable and convergent solution. Therefore the CFL condition is useful in helping us choose a suitable time step size for our simulation. A common

way of selecting the time step size is to keep Courant number at 1, using the maximum possible u and the smallest possible mesh size, a Δt that fits the criteria can be calculated.

sonicFoam – shockTube

Simulation

Use the sonicFoam solver, simulate 0.007 s of flow inside a shock tube, with a mesh with 100, 1000 and 10000 cells in one dimension, for initial values 1 bar/0.1 bar and 10 bar/0.1 bar.

Objectives

- To understand the setFields utility
- Learn how to specify initial and boundary conditions
- Investigate effect of grid resolution

Data processing

Import your simulation into ParaView, and compare results.

1. Pre-processing

Open tutorial

Copy the tutorial from the following directory to your working directory

`$FOAM_TUTORIALS//compressible/sonicFoam/laminar/shockTube`

OpenFOAM® v1712: Create a copy of 0.orig folder and rename it to 0!

system directory

Looking at the *blockMeshDict* file, it is obvious that it is a 1D mesh, because of the number of mesh cells in y and z directions is one, and also in `boundary` section, plates vertical to these directions are defined as `empty`. The mesh density can be set in the `blocks` part by changing x direction mesh size (e.g. change it from `1000` to `100` or `10000`).

Another important file is *setFieldsDict*, which is used by the tool `setFields` for patching (assign an amount to a region) in the simulation. For example, here a pressure of 10 bar is set as the default value for the entire region (from -5 to 5 in x direction), then half of the region (from 0 to 5) is patched with a pressure of 0.1 bar.

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

defaultFieldValues ( volVectorFieldValue U ( 0 0 0 ) volScalarFieldValue T 348.432
volScalarFieldValue p 1000000 );

regions         ( boxToCell { box ( 0 -1 -1 ) ( 5 1 1 ) ; fieldValues ( volScalarFieldValue T
278.746 volScalarFieldValue p 10000 ) ; } );

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

In the `defaultFieldValues`, a value is assigned to the whole domain, for example here, the velocity has been set everywhere to zero, the temperature to 348.432 K, and the pressure to 1000000 Pa. In the `regions`, a specific value is patched to a certain region of the domain. The region is by default a cube, and is defined by the coordinates of one of its diagonals in `boxToCell` .

After choosing the region, the new values are assigned to the parameters (e.g. temperature at 278.746 K and pressure at 10000 Pa).

2. Running simulation

First the mesh needs to be created:

`>blockMesh`

In order to assign the values which were set in the setFieldDict:

`>setFields`

Then run:

`>sonicFoam`

Note: In the 10000 cell case with 10 bar and 0.1 bar, the simulation will crash with the default `deltaT` *(*`1e-5`*); after checking the same case with 1000 cells, you will find that the maximum Co is around 0.6:*

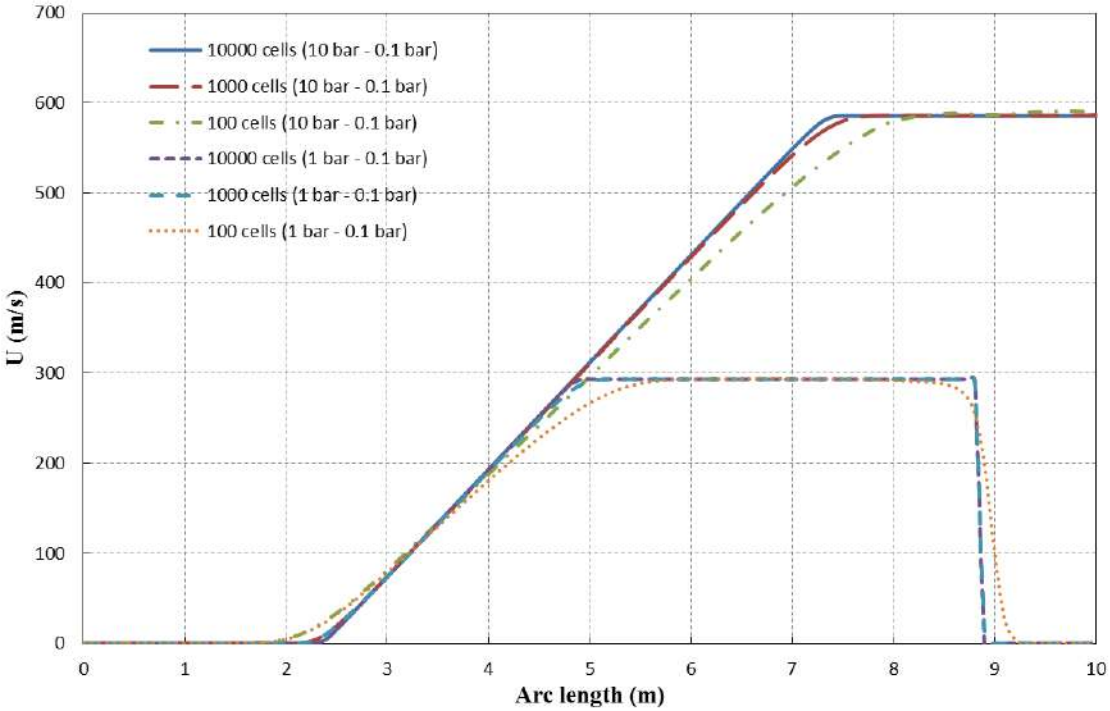
```
Time = 0.001

Courant Number mean: 0.0508555 max: 0.589018
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
```

In the case with 10000 cells, the number of cells is increased by factor 10, so the cell size is reduced by factor 10. For keeping the Courant number in the same range (around 0.6), according to the "Background" section, `deltaT` *should be decreased by factor 10. After reducing it to* `1e-6` *the simulation will run smoothly.*

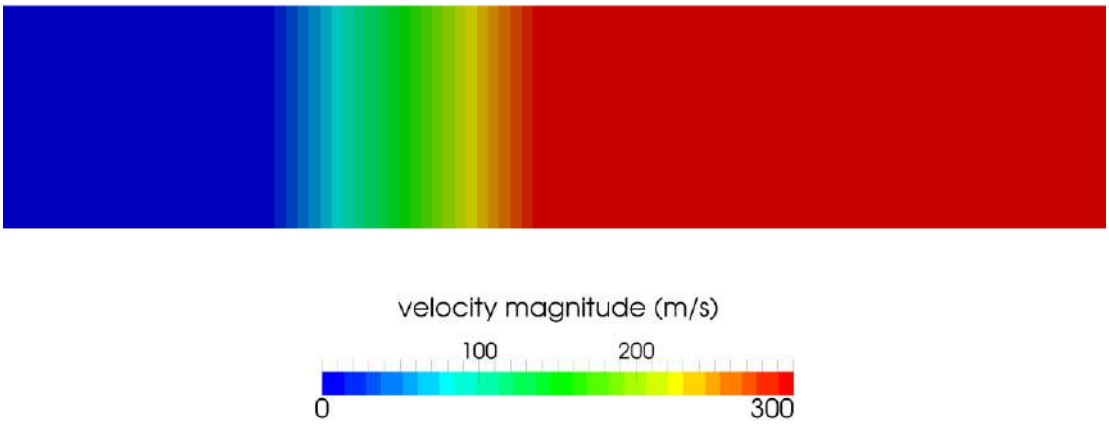
Note: After running setFields for the first time, the files in the 0 directory are overwritten. If the mesh will be changed these files are not compatible with the new mesh and the simulation will fail. To solve this problem replace the files in the 0 directory with the files in the 0.orig. In the OpenFOAM® files or directories with suffix ".orig" ("original") usually contain the backup files. If a command changes the original files these files can be replaced.

3. Post-processing

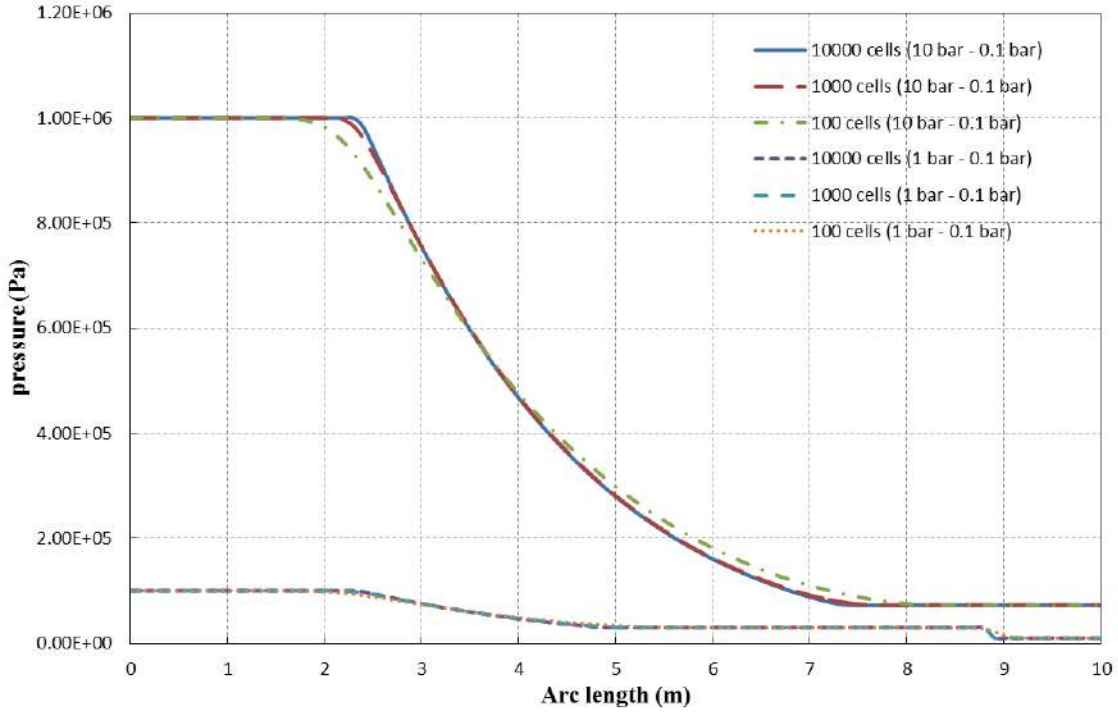
The simulation results are as follows:



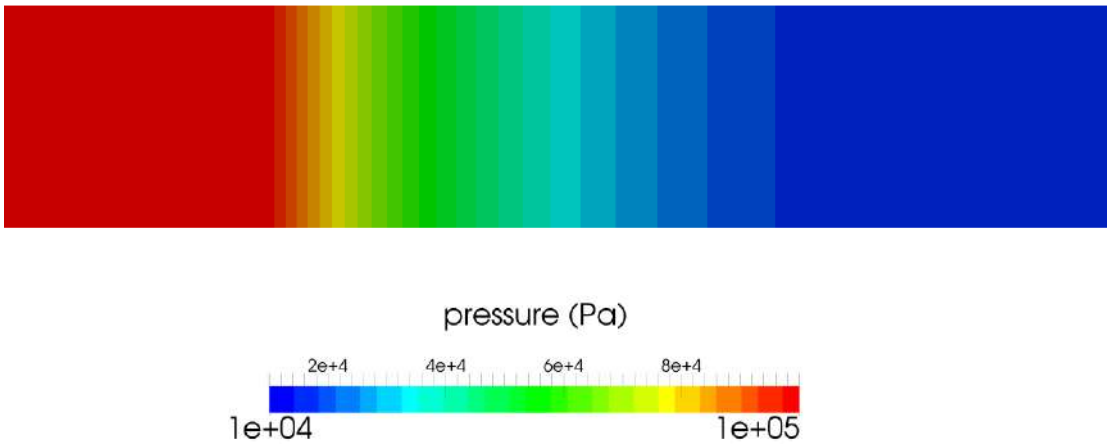
Velocities for different configurations along tube at t = 0.007 s



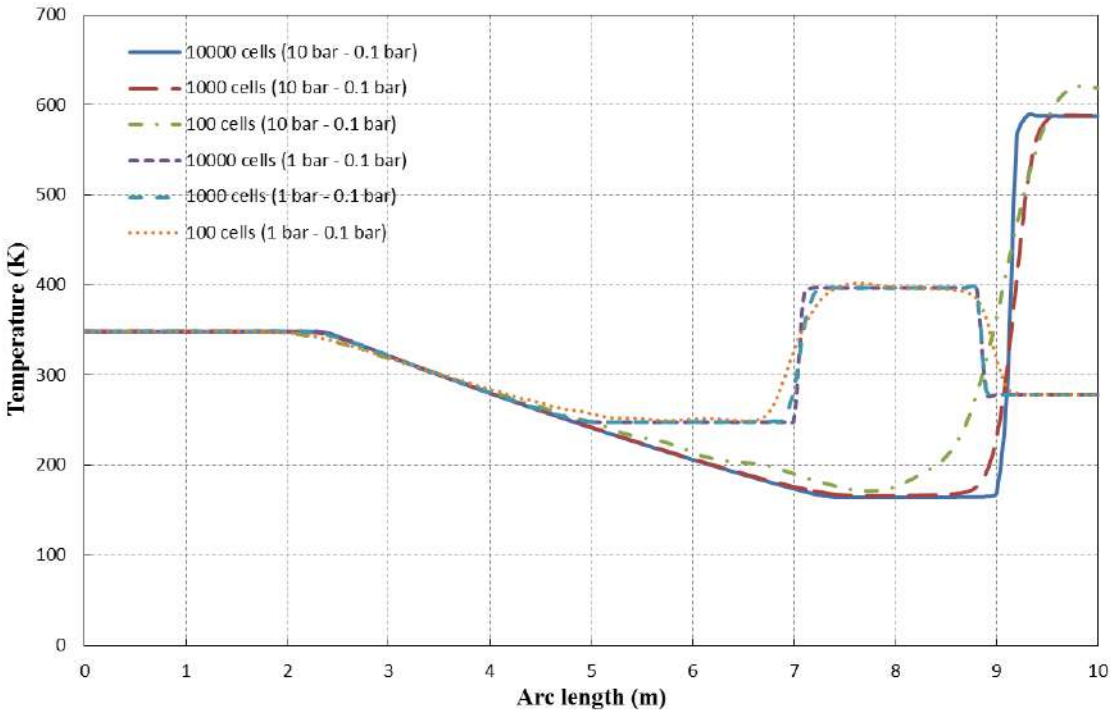
Velocity along tube axis for 10 bar/0.1bar and 10000 cells case at t = 0.007 s



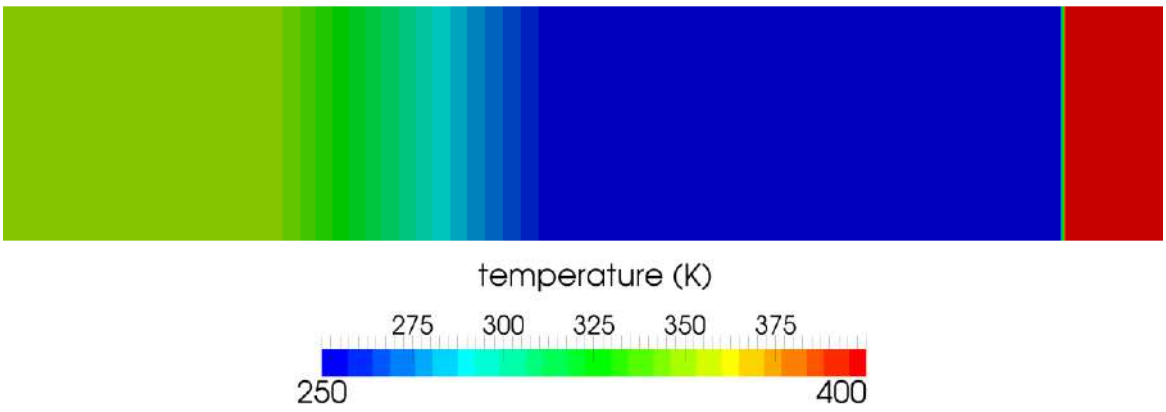
Pressures for different configurations along tube at t = 0.007 s



Pressure along tube axis for 10 bar/0.1bar and 10000 cells case at t = 0.007 s



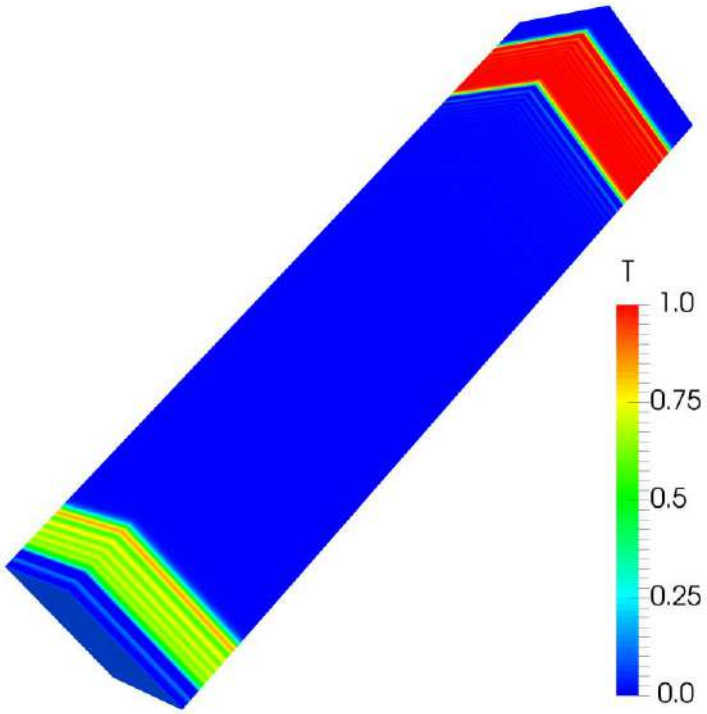
Temperature for different configurations along tube at t = 0.007 s



Temperature along tube axis for 10 bar/0.1bar and 10000 cells case at t = 0.007 s

Tutorial Four

Discretization – Part 1



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Discretizing general transport equation terms

Tutorial Four aims to help the users understand the different discretization schemes in OpenFOAM®. But before that, it is important to understand the exact mathematical procedures involved in discretization. Below is a detailed explanation of how each term of the transport equation is discretized.

1.1. Time derivative

Discretization of the time derivative such as $\frac{\partial \rho\varphi}{\partial t}$ of the transport equation is performed by integrating it over the control volume of a grid cell. Here, the Euler implicit time differencing scheme is explained. It is unconditionally stable, but only first order accurate in time. Assuming linear variation of φ within a time step gives:

$$\int_V \frac{\partial \rho\varphi}{\partial t} dV \approx \frac{\rho_P^n \varphi_P^n - \rho_P^0 \varphi_P^0}{\Delta t} V_P$$

Where $\varphi^n \equiv \varphi\ (t + \Delta t)$ stands for the new value at the time step we are solving for and $\varphi^0 \equiv \varphi(t)$ denotes old values from the previous time step.

1.2. Convection term

Discretization of convection terms is performed by integrating over a control volume and transforming the volume integral into a surface integral using the Gauss's theorem as follows:

$$\int_A \boldsymbol{n} \cdot (\rho\varphi\boldsymbol{u})\, dA \approx \sum_f \boldsymbol{n} \cdot (A\rho\boldsymbol{u})_f \varphi_f = \sum_f F\varphi_f$$

Where F is the mass flux through the face f defined as $F = \boldsymbol{n} \cdot (A\rho\boldsymbol{u})_f$. The value φ_f on face f can be evaluated in a variety of ways which will be covered later in section 2. The subscript f refers to a given face.

1.3. Diffusion term

Discretization of diffusion terms is done in a similar way to the convection terms. After integration over the control volume, the term is converted into a surface integral:

$$\int_A \boldsymbol{n} \cdot (\Gamma\nabla\varphi)\, dA = \sum_f \Gamma_f (\boldsymbol{n} \cdot \nabla_f \varphi) A_f$$

Note that the above approximation is only valid if Γ is a scalar. Here, $\nabla_f \varphi$ denotes the gradient at the face, A denotes the surface area of the control volume and A_f denotes the area of a face for the control volume. It does not, however, imply a specific

discretization technique. The face normal gradient can be approximated using the scheme:

$$\boldsymbol{n} \cdot \nabla_f \varphi = \frac{\varphi_N - \varphi_P}{|\boldsymbol{d}|}$$

This approximation is second order accurate when the vector $\boldsymbol{d}$ between the center of the cell of interest P and the center of a neighboring cell N is orthogonal to the face plane, i.e. parallel to A. In the case of non-orthogonal meshes, a correction term could be introduced which is evaluated by interpolating cell centered gradients obtained from Gauss integration.

1.4. Source term

Source terms, such as $S_\varphi$of the transport equation, can be a general function of φ. Before discretization, the term is linearized:

$$S_\varphi = \varphi S_I + S_E$$

where S_E and S_I may depend on φ. The term is then integrated over a control volume as follows:

$$\int_V S_\varphi dV = S_I \mathrm{V}_P \varphi_P + S_E \mathrm{V}_P$$

There is some freedom on exactly how a particular source term is linearized. When deciding on the form of discretization (e.g. linear, upwind), its interaction with other terms in the equation and its influence on boundedness and accuracy should be examined.

2. Discretization Schemes

Since the results of CFD simulations are typically stored at the cell centers, it is important to interpolate the results from cell centers to the face centers, to obtain the fluxes for the surface integrals in the transport equation. For each term of the transport equation, there is a variety of discretization/interpolation schemes available.

In general, interpolation needs a flux F through a general face f, and in some cases, one or more parameters γ. The face value φ_f can be evaluated from the values in the neighboring cells using a variety of schemes. The flux satisfies continuity constraints, which is prerequisite to obtaining the results.

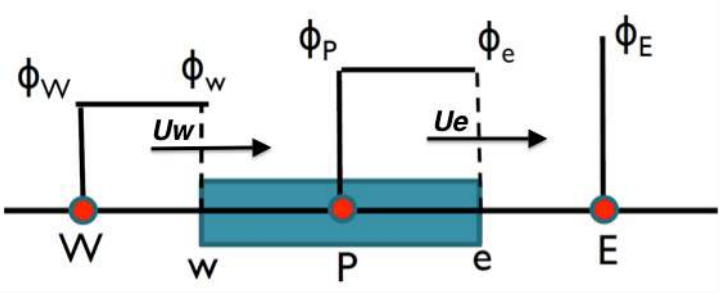
2.1. First Order Upwind Scheme

In first order upwind scheme we define φ as follows:

Note: Here we define two faces, e and w. To obtain flux through faces e and w, we need to look its neighbouring values at P/E and W/P respectively. The subscripts denote the face at which the face value φ or the flux F is located at.

$$\varphi_e = \varphi_P \qquad if, F_e > 0$$

$$\varphi_e = \varphi_E \qquad if, F_e < 0$$



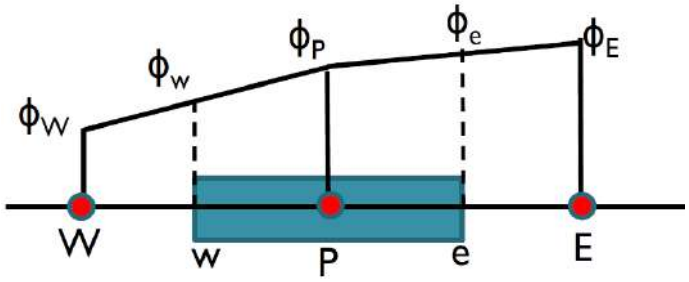
First Order Upwind Scheme

φ_w is also defined similarly (Positive direction is from W to E).

2.2. Central Differencing Scheme

Here, we use linear interpolation for computing the cell face values.

$$\varphi_e = \frac{\varphi_E + \varphi_P}{2}, \qquad \varphi_w = \frac{\varphi_P + \varphi_W}{2}$$



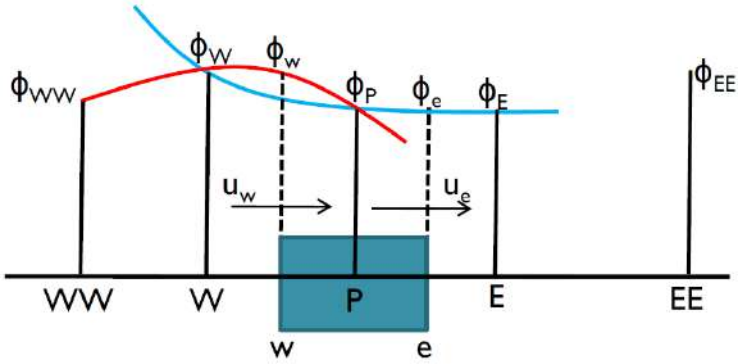
Central Differencing Scheme

2.3. QUICK

QUICK stands for Quadratic Upwind Interpolation for Convective Kinetics. In the QUICK scheme 3 point upstream-weighted quadratic interpolation are used for cell face values.

When $F_e > 0$, $\varphi_e = \frac{6}{8}\varphi_P + \frac{3}{8}\varphi_E - \frac{1}{8}\varphi_W$

When $F_w > 0$, $\varphi_w = \frac{6}{8}\varphi_W + \frac{3}{8}\varphi_P - \frac{1}{8}\varphi_{WW}$



QUICK scheme

Similar expressions can be obtained for $F_e < 0$ and $F_w < 0$.

Now that you know a bit more about discretization schemes, we can move on to the tutorial. In this tutorial the *scalarTransportFoam* solver is used. More explanation of this solver can be found below.

4. scalarTransportFoam solver

scalarTransportFoam is a basic solver which resolves a transport equation for a passive scalar. The velocity field and boundary condition need to be provided by the user. It works by setting the source term in the transport equation to zero (see equation below), and then solving the equation.

$$\frac{\partial(\rho\varphi)}{\partial t} + \nabla \cdot (\rho\varphi \boldsymbol{u}) - \nabla \cdot (\Gamma \nabla\varphi) = 0$$

scalarTransportFoam – shockTube

Simulation

Use the scalarTransportFoam solver, simulate 5 s of flow inside a shock tube, with 1D mesh of 1000 cells (10 m long geometry from -5 m to 5 m). Patch with a scalar of 1 from -0.5 to 0.5. Simulate following cases:

- Set U to uniform (0 0 0). Vary diffusion coefficient (low, medium and high value).
- Set the diffusion coefficient to zero and also U to (1 0 0) and run the simulation in the case of pure advection using following discretization schemes:
  - upwind
  - linear
  - linearUpwind
  - QUICK
  - cubic

Objectives

- Understanding different discretization schemes.

Data processing

Import your simulation into ParaView, and plot temperature along tube length.

1. Pre-processing

1.1. Compile tutorial

Create a folder in your working directory:

```
>mkdir shockTube
```

Copy the following case to the created directory:

```
$FOAM_TUTORIALS/compressible/sonicFoam/laminar/shockTube
```

In the 0 and 0.orig directories, delete *p* files. In the constant directory delete the *thermophysicalProperties* and *turbulenceProperties* files, and in the system directory delete *controlDict*, *fvSchemes* and *fvSolution* files.

From the following case:

```
$FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily
```

Copy *transportProperties* file from constant folder in the newly created case constant folder. Copy *controlDict*, *fvSchemes* and *fvSolution* from the above case system directory to the created case system directory.

1.2. constant directory

The diffusion coefficient can be set in the *transportProperties* file. For a low value try 0.00001, for a medium value use 0.01 and for a high value use 1:

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

Note: By setting the diffusion coefficient to zero, the case will be switched to a pure advection simulation with no diffusion.

OpenFOAM® v1712: Just DT and its value are listed – no dimensions!

1.3. system directory

Edit the *setFieldsDict*, to patch the T field from -0.5 m to 0.5 m and also to set the U to (0 0 0) for the whole domain. For setting U in the whole domain to (1 0 0), just change `(0 0 0)` to `(1 0 0)`:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
defaultFieldValues
(
        volVectorFieldValue U ( 0 0 0 )
        volScalarFieldValue T 0.0
);
regions
(
        boxToCell
        {
                box ( -0.5 -1 -1 ) ( 0.5 1 1 );

                fieldValues
                (
                        volScalarFieldValue T 1.0
                );
```

```
        }
);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
```

As it was mentioned before, the discretization scheme for each operator of the governing equations can be set in *fvSchemes*.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,T)      Gauss linearUpwind grad(T);
}

laplacianSchemes
{
    default         none;
    laplacian(DT,T) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

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

For each type of operation a default scheme can be set (e.g. for `divSchemes` is set to `none`, it means no default scheme is set). Also a special type of discretization for each element can be assigned (e.g. `div(phi,T)` it is set to `linearUpwind`). For each element, where a discretization method has not been set, the default method will be applied. If the default setting is `none`, no scheme is set for that element the simulation will crash.

Note: In fvSchemes, the schemes for the time term of the general transport equation are set in `ddtSchemes` *sub-dictionary.* `divSchemes` *are responsible for the advection term schemes and* `laplacianSchemes` *set the diffusion term schemes.*

Note: `divSchemes` *should be applied like this: Gauss + scheme. The* `Gauss` *keyword specifies the standard finite volume discretization of Gaussian integration which requires the interpolation of values from cell centers to face centers. Therefore, the* `Gauss` *entry must be followed by the choice of interpolation scheme (www.openfoam.org).*

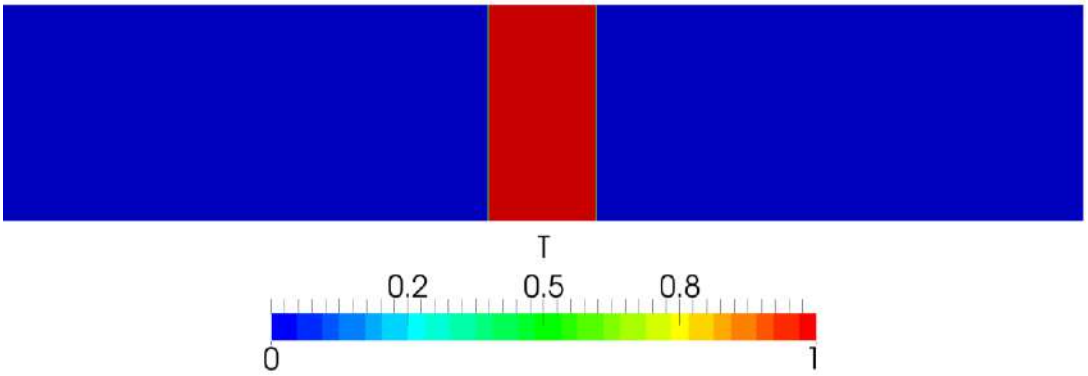
2. Running simulation

```
>blockMesh
>setFields
>scalarTransportFoam
```

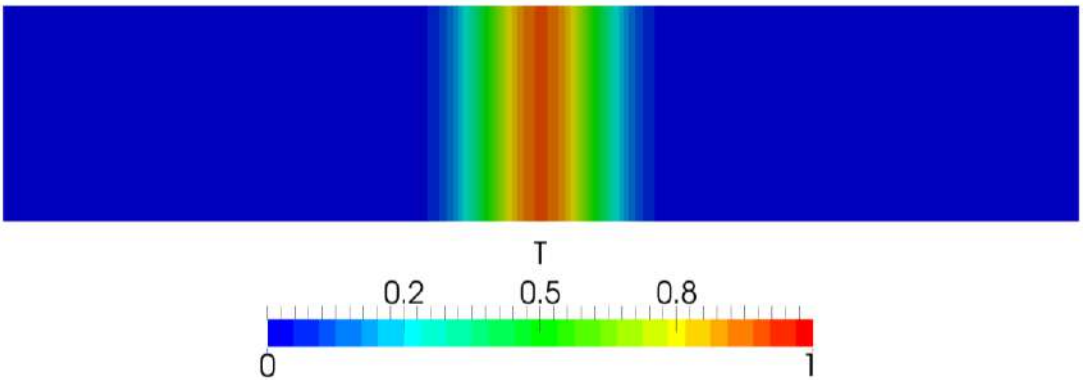
3. Post-processing

The simulation results are as follows.

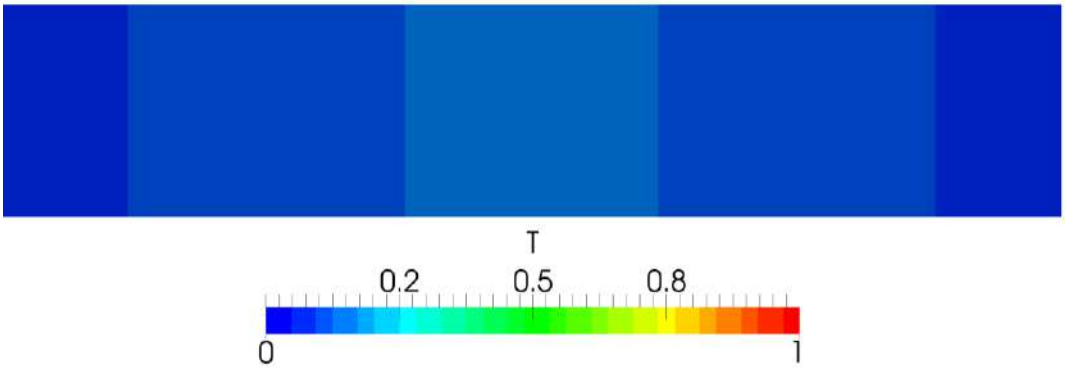
A) Case with zero velocity (pure diffusion):



Pure diffusion with low diffusivity (0.00001) at t = 5 s

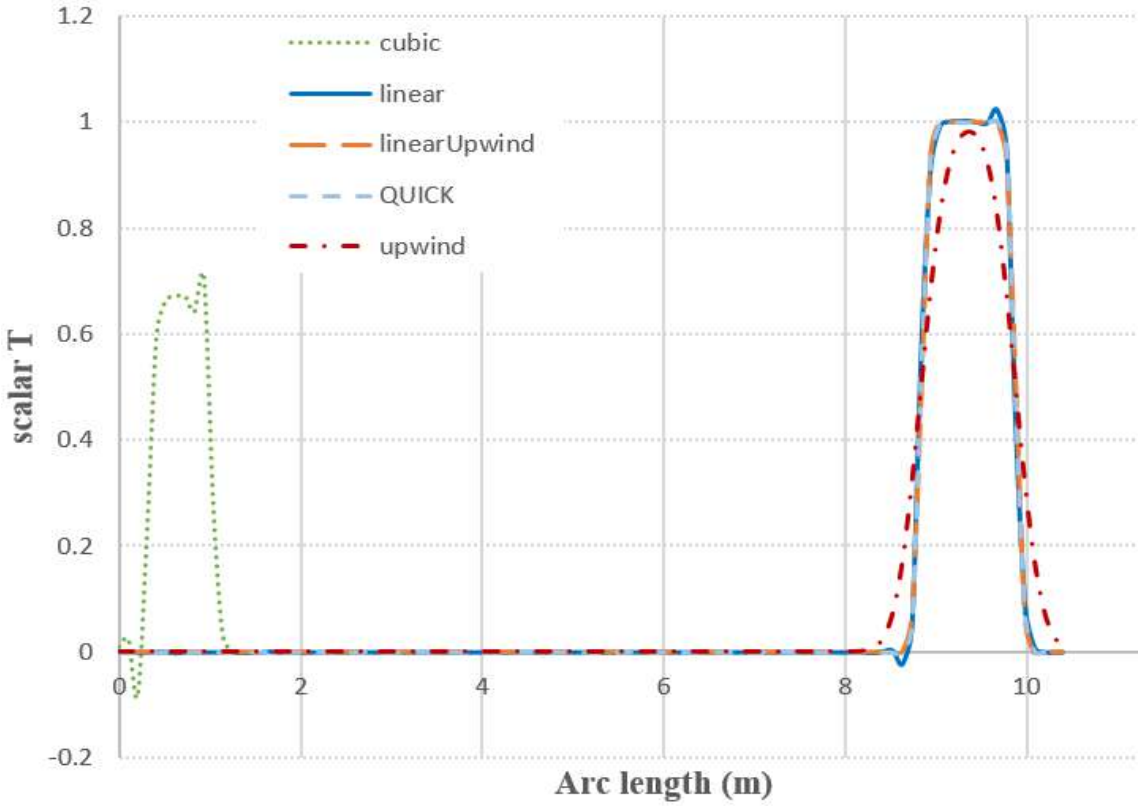


Pure diffusion with medium diffusivity (0.01) at t = 5 s



Pure diffusion with high diffusivity (1) at t = 5 s

B) Case with pure advection (diffusion coefficient = 0):

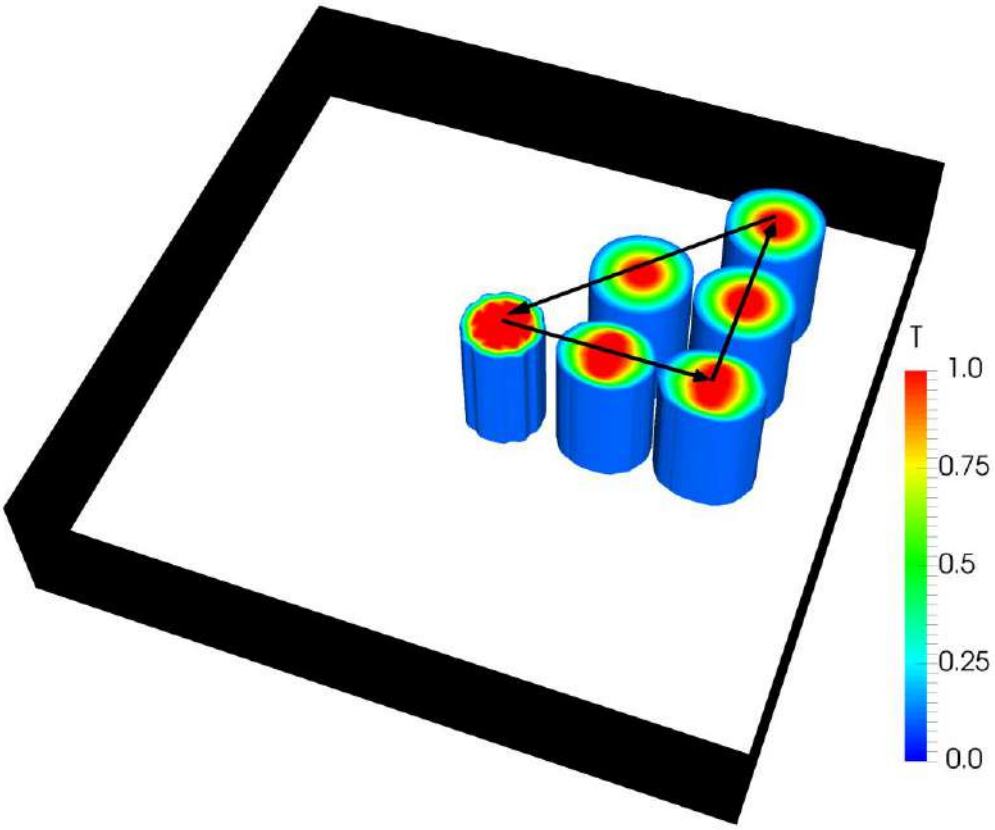


Scalar T along tube at t = 4 s

The cubic scheme predicted an unexpected rise in temperature between around 0 to 1 m, which differs hugely from the other schemes. This can be explained by looking at the numerical behavior of the cubic scheme. It is operated in fourth order accuracy with unbounded solutions, which caused another false root solution to be found. So, higher order accuracy does not always generate better results!

Tutorial Five

Discretization – Part 2



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Properties of discretization schemes

Let's explore some fundamental properties of discretization schemes. These properties are required for our numerical results to be physically realistic. An understanding of these properties will help the users to choose the appropriate discretization schemes for their model.

1.1. Conservativeness

Integration of the convection–diffusion equation over a finite number of control volumes yields a set of discretized conservation equations involving fluxes of the transported property φ through control volume faces. To ensure conservation of φ for the whole solution domain the flux of φ leaving a control volume across a certain face must be equal to the flux of φ entering the adjacent control volume through the same face. To achieve this flux through a common face must be represented in a consistent manner – by one and the same expression – in adjacent control volumes of each face.

1.2. Boundedness

Normally we use iterative numerical techniques to solve discretized equations at each nodal point. The methods start with a guessed distribution of the initial conditions of the variable φ and perform successive updates until a converged solution is obtained.

The sufficient condition for a converged solution is:

$$\frac{\sum|a_{nb}|}{|a'_P|} \begin{cases} \leq 1 \; at \; all \; nodes \\ < 1 \; at \; one \; node \; at \; least \end{cases}$$

Here a'_P is the net coefficient of the central node P (i.e. $a'_P - S_P$), a_{nb} are the coefficient of the neighbouring nodes. If the condition is satisfied, the resulting matrix of coefficients is diagonally dominant. We need the net coefficients to be as large as possible, this means that S_P should be always negative. If this is the case, S_P becomes positive due to the modulus sign and adds to a_P.

1.3. Transportiveness

To understand transportiveness, one should look at a dimensionless number called the Peclet number, Pe. It measures the relative strengths of convection, N_{conv} and diffusion, N_{diff}.

$$Pe = \frac{N_{conv}}{N_{diff}} = \frac{LU}{D}$$

Note: L is a characteristic length scale, U is the velocity magnitude, D is a characteristic diffusion coefficient.

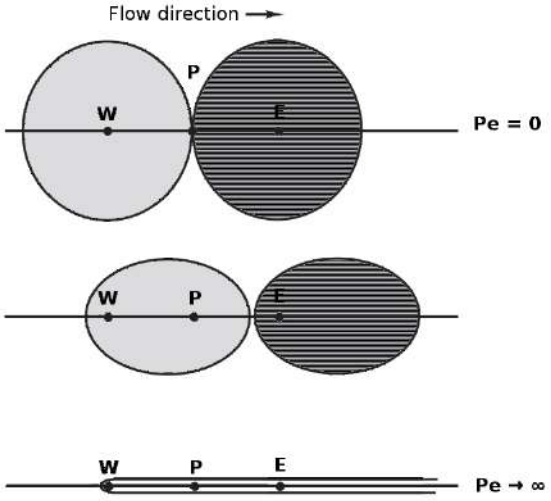
The primary goal is to ensure that the transportiveness is borne out of the discretization scheme.

Let's consider the effect at a point P due to two constant sources of φ at nearby points *W* and *E* on either side, in three cases.

1) When Pe = 0 (pure diffusion), the countours of φ are circles, as φ is spread out evenly in all directions

2) As Pe increases, the contours become elliptical, as the values of φ are influenced by convection

3) When Pe→∞, the countours become straight lines, since φ are stretched out completely and affected only by upstream conditions



Transportiveness property

2. Assessing the general discretization schemes

It is useful to compare the different types of general discretization schemes covered in Tutorial Four based on their conservativeness, boundedness and transportiveness properties.

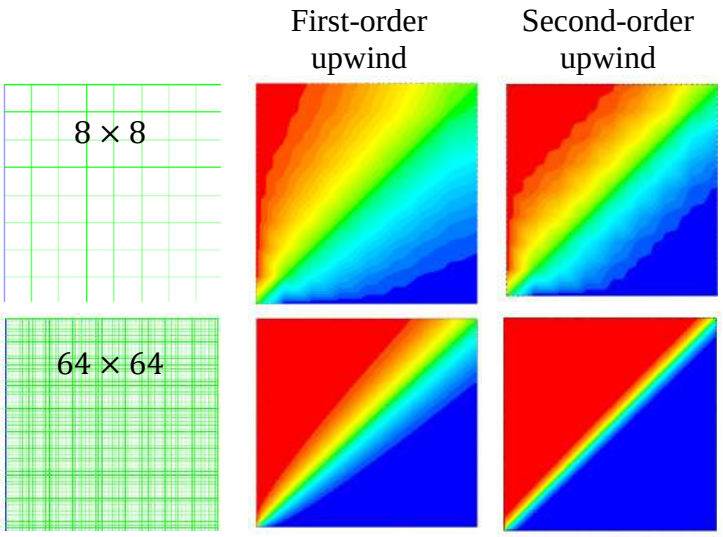
Different discretizing schemes assessment

Scheme	Conser -vative	Bounded	Accuracy	Trans- portive	Remarks
Upwind	Yes	Unconditionally bounded	First order	Yes	Include false diffusion if the velocity vector is not parallel to one of the coordinate directions
Central Differencing	Yes	Conditionally bounded*	Second order	No	Unrealistic solutions at large Pe number
QUICK	Yes	Unconditionally bounded	Third order	Yes	Less computationally stable. Can give small undershoots and overshoots

****Pe should be less than 2.***

3. Numerical (false) diffusion

Numerical diffusion is a multidimensional phenomenon and it occurs when the flow is not perpendicular to the grid lines. It is a numerically introduced diffusion and arises in convection dominated flows, i.e. high Pe number flows.



Numerical diffusion

4. Numerical behavior of OpenFOAM® discretization schemes

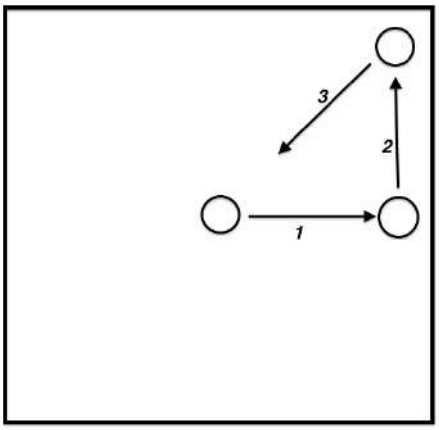
The choice of discretization scheme for this tutorial should depend critically on the numerical behaviour of the scheme. Using higher order schemes, numerical diffusion errors can be reduced, however it requires higher computational efforts.

Scheme	Numerical behaviour
upwind	First order, bounded
linear	Second order, unbounded
linearUpwind	First/second order, bounded
QUICK	Second order or higher, bounded
cubic	Fourth order, unbounded

scalarTransportFoam – circle

Simulation

Use the scalarTransportFoam solver, do simulate the movement of a circular scalar spot region (radius = 1 m) at the middle of a 100 × 100 cell mesh (10 m × 10 m), then move it to the right, to the top and diagonally.



Schematic sketch of the problem

Objectives

- Choosing the best discretization scheme.

Data processing

Examine your simulation in ParaView.

1. Pre-processing

1.1. Compile tutorial

Create the new case in your working directory like in tutorial four.

1.2. 0 directory

To move the circle to right change the `internalField` to `(1 0 0)` in the U file for setting the velocity field towards the right. Modify U at suitable times, to obtain a velocity field which will move the circle up and also diagonally.

1.3. constant directory

In the *transportProperties*, set `DT` to zero (no diffusion!).

1.4. system directory

Modify the *blockMeshDict* for creating a 2D geometry with 100 × 100 cells mesh.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
convertToMeters 1;

vertices
(
    (-5 -5 -0.01)
    (5 -5 -0.01)
    (5 5 -0.01)
    (-5 5 -0.01)
    (-5 -5 0.01)
    (5 -5 0.01)
    (5 5 0.01)
    (-5 5 0.01)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (100 100 1) simpleGrading (1 1 1)
);
edges
(
);
boundary
(
    sides
    {
        type patch;
        faces
        (
            (1 2 6 5)
            (0 4 7 3)
            (3 7 6 2)
            (0 1 5 4)
        );
    }
    empty
    {
        type empty;
        faces
        (
            (5 6 7 4)
            (0 3 2 1)
        );
    }
);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

Choose a discretization scheme based on the results from the previous example and set it in the *fvSchemes*.

In the *setFieldsDict* patch a circle to the middle of the geometry using the following lines.

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

defaultFieldValues (volScalarFieldValue T 0 );

regions
(
	cylinderToCell
	{
		p1 ( 0 0  -1 );
		p2 ( 0 0   1 );
		radius 0.5;
		fieldValues
		(
			volScalarFieldValue T 1
		) ;
	}
);

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

`cylinderToCell` command is used to patch a cylinder to the region, `p1` and `p2` show the two ends of cylinder center line, in the `radius` the radius is set.

Check *controlDict*, in the first part of simulation, where the circle should move to the right set the `startFrom` to `startTime` and `startTime` to `0`. By a simple calculation it can be seen that the `endTime` should be `3` s. Similar calculations need to be done for the two other parts, except the `startTime` is set to the `endTime` of previous part, and new `endTime` should be that part "simulation time" plus `endTime` of the previous part.

2. Running Simulation

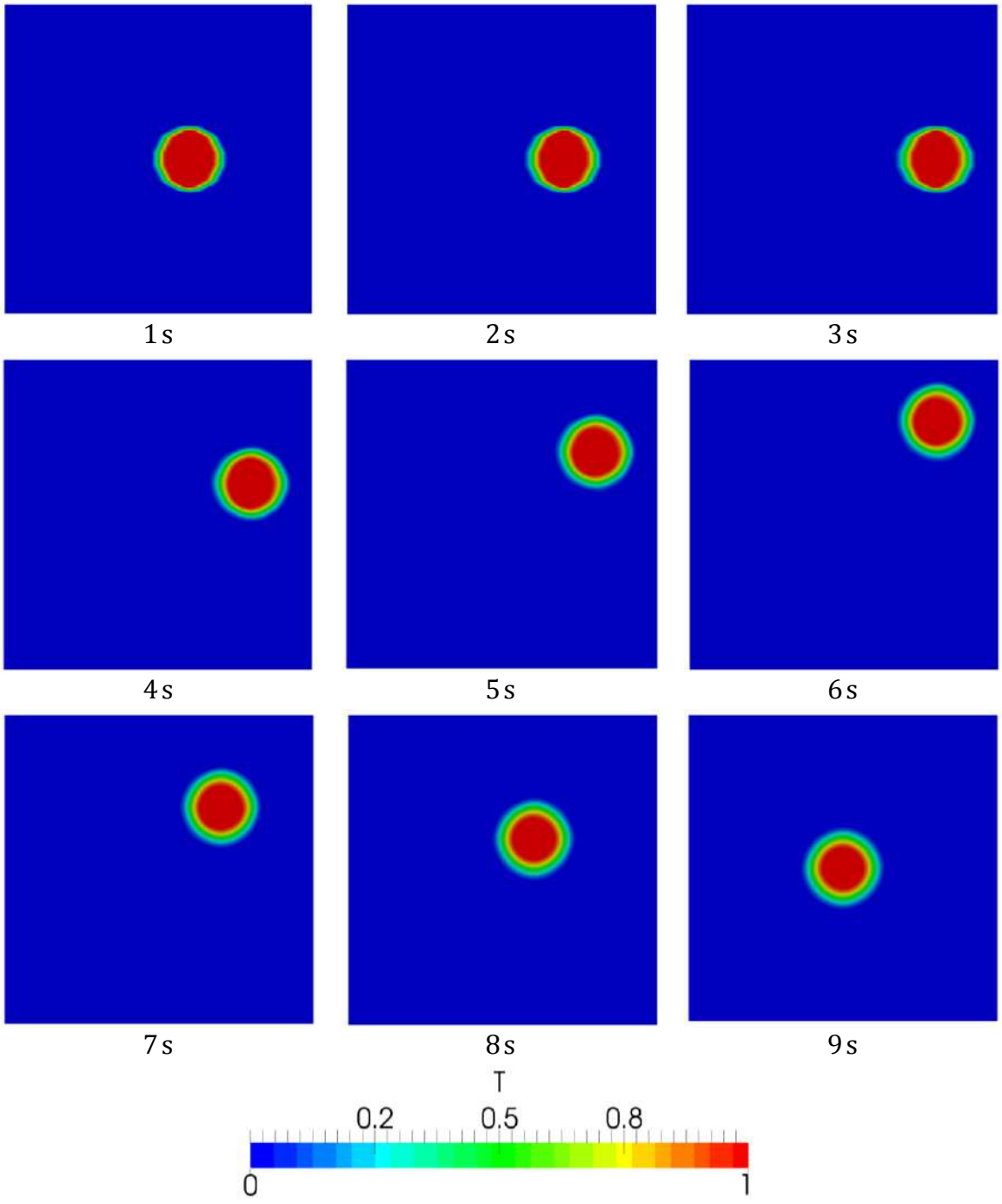
```
>blockMesh
>setFields
>scalarTransportFoam
```

For running the further parts (moving the circle to top, and then diagonally) change the velocity field in the last time step directory, i.e. change the velocity in the time step directory 3 to (0 1 0) so the circle moves up, further change the velocity in the directory 6 to (-1 -1 0) to move the circle diagonally back to the original position.

After moving the circle to the right and changing the velocity field, the simulation is resumed. It can be seen that the circle does not go up but moves to the right. This occurs due to the fact that OpenFOAM® used the previous time step fluxes (phi) to do the calculations. We can solve this problem by deleting phi file from the latest time step (of the previous part of simulation, e.g. 3). In this way, OpenFOAM® creates new fluxes based on the new velocity field that we just updated. So, easily delete phi and enjoy!

3. Post-processing

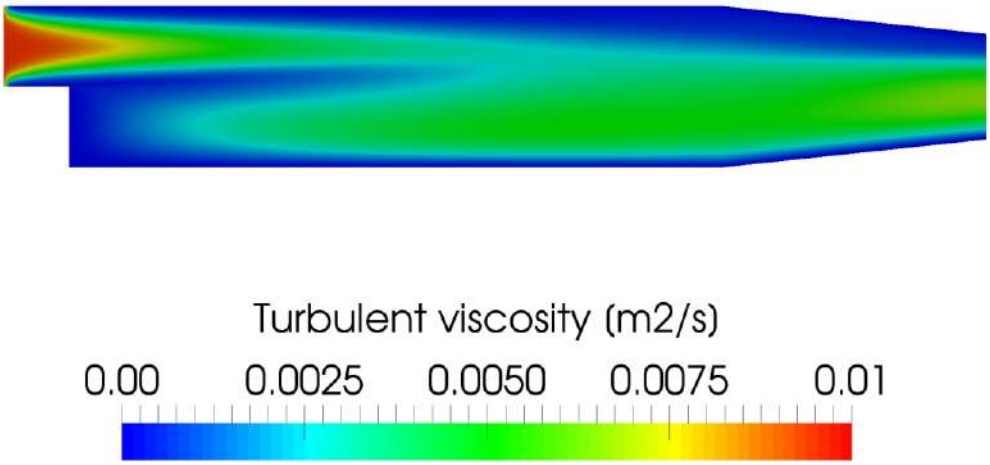
The simulation results are as follows:



Position of the circle at different time steps

Tutorial Six

Turbulence – Steady State



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Why turbulence modeling?

Many engineering applications are turbulent. Turbulence is a highly transient phenomenon, characterized by a wide range of eddy sizes. One can solve these eddies numerically and obtain a full profile of the turbulent flow field. However, this is not possible as it requires a huge amount of computational effort. Hence we require a turbulence model.

An important feature in turbulence modeling is averaging, which simplifies the solution of the governing equations of turbulence. As calculating resources are limited, it is usually not possible to model the phenomena with desired grid and time resolution, so to represent scales of the flow that are not resolved by the grid models need to be applied.

There are different types of turbulence models:

- RANS-based models:
  - Linear eddy-viscosity models
    - Algebraic models
    - One and two equation models
  - Non-linear eddy viscosity models and algebraic stress models
  - Reynolds stress transport models
- Large eddy simulations
- Detached eddy simulations and other hybrid models

In this tutorial, RANS-based model is explained in detail. In the next tutorial, large eddy simulations (LES) and Smagorinsky-Lilly model will be covered.

2. RANS-based models

The governing equations for a Newtonian fluid are:

- Conservation of mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \tilde{\boldsymbol{u}}) = 0$$

- Conservation of momentum (Navier-Stokes equation)

$$\frac{\partial (\rho \tilde{u}_i)}{\partial t} + \nabla \cdot (\rho \tilde{u}_i \tilde{\boldsymbol{u}}) = -\frac{\partial \tilde{p}}{\partial x_i} + \nabla \cdot (\mu \nabla \tilde{u}_i) + \tilde{S}_{Mi}$$

- Conservation of passive scalars (given a scalar $\tilde{e}$)

$$\frac{\partial (\rho \tilde{e})}{\partial t} + \nabla \cdot (\rho \tilde{e} \tilde{\boldsymbol{u}}) = \nabla \cdot \left(k \nabla \tilde{T}\right) + \tilde{S}_e$$

Note: *suffix notation is used in the conservation of momentum equation for simplicity, with* $i = 1$ *corresponding to the x-direction,* $i = 2$ *the y-direction and* $i = 3$ *the z-direction.*

One of the solutions to the problem is to reduce the number of scales (from infinity to 1 or 2) by using the Reynolds decomposition. Any property (whether a vector or a scalar) can be written as the sum of an average and a fluctuation, i.e. $\tilde{\varphi} = \Phi + \varphi$ where the capital letter denotes the average and the lower case letter denotes the fluctuation of the property. Using the Reynolds decomposition in the Navier-Stokes equations we obtain RANS or Reynolds Averaged Navier Stokes Equations.

- Average conservation of mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0$$

- Average conservation of momentum

$$\frac{\partial (\rho \mathrm{U}_i)}{\partial t} + \nabla \cdot (\rho \mathrm{U}_i \mathbf{U}) = -\frac{\partial P}{\partial x_i} + \nabla \cdot (\mu \mathrm{U}_i) - \left(\frac{\partial (\rho \overline{u u_i})}{\partial x} + \frac{\partial (\rho \overline{v u_i})}{\partial y} + \frac{\partial (\rho \overline{w u_i})}{\partial z} \right) + \mathrm{S}_{Mi}$$

- Average conservation of passive scalars (given a scalar $\tilde{e}$)

$$\frac{\partial (\rho \mathrm{E})}{\partial t} + \nabla \cdot (\rho \mathrm{E} \mathrm{U}) = \nabla \cdot (k \nabla \mathrm{T}) - \left(\frac{\partial (\rho \overline{u e})}{\partial x} + \frac{\partial (\rho \overline{v e})}{\partial y} + \frac{\partial (\rho \overline{w e})}{\partial z} \right) + \mathrm{S}_e$$

Note: a special property of the Reynolds decomposition is that the average of the fluctuating component is identically zero, a fact that is used in the derivation of the above equations.

However, by using the Reynolds decomposition, there are new unknowns that were introduced such as the turbulent stresses ($\rho\overline{uu}$, $\rho\overline{vu}$, $\rho\overline{wu}$, $\rho\overline{uv}$, $\rho\overline{vv}$, $\rho\overline{wv}$, $\rho\overline{uw}$, $\rho\overline{vw}$, $\rho\overline{ww}$) and turbulent fluxes ($\rho\overline{ue}$, $\rho\overline{ve}$, $\rho\overline{we}$) and therefore, the RANS equations describe an open set of equations (where the over bar denotes an average). The need for additional equations to model the new unknowns is called Turbulence Modeling.

We now have 9 additional unknowns (6 Reynolds stresses and 3 turbulent fluxes). In total, for the simplest turbulent flow (including the transport of a scalar passive scalar, e.g. temperature when heat transfer is involved) there are 14 unknowns (include *u, v, w, p, T*)!

One possible approach to model the additional unknowns is to use the PDEs for the turbulent stresses and fluxes as a guide to modeling. The turbulent models are as follows, in order of increasing complexity:

- Algebraic (zero equation) models: mixing length (first order model)

- One equation models: k-model, μ_t-model (first order model)
- Two equation models: k-ε, k-kl, k-ω, low Re k-ε (first order model)
- Algebraic stress models: ASM (second order model)
- Reynolds stress models: RSM (second order model)
- Zero-Equation Models

In OpenFOAM®, there are two simulation types for turbulence flow, RAS and LES. As the name suggest, the RAS simulation is based on the RANS-based models covered above and will be the sole focus of this tutorial. In the next tutorial, we will move on to LES modeling and compare the results generated from these two modeling types.

simpleFoam – pitzDaily

Simulation

Use simpleFoam solver, run a steady state simulation with following turbulence models:

- kEpsilon (RAS)
- kOmega (RAS)
- LRR (RAS)

Objectives

- Understanding turbulence modeling
- Understanding steady state simulation

Data processing

Show the results of U and the turbulent viscosity in two separate contour plots.

1. Pre-processing

1.1. Copy tutorial

$FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily

1.2. 0 directory

When a turbulent model is chosen, the value of its constants and its boundary values should be set in the appropriate files. For example in kEpsilon model the k and epsilon files should be edited. See below for the epsilon file:

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

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

internalField   uniform 14.855;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 14.855;
    }
    outlet
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            epsilonWallFunction;
        value           uniform 14.855;
    }
    lowerWall
    {
        type            epsilonWallFunction;
        value           uniform 14.855;
    }
    frontAndBack
    {
        type            empty;
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
```

Note: Here is a list of files which should be available at 0 directory and need to be modified for each turbulence model:

- *laminar: no file*
- *kEpsilon (RAS): k and epsilon*
- *kOmega (RAS): k and omega*
- *LRR (RAS): k, epsilon and R*
- *Smagorinsky (LES): nuSgs*
- *kEqn (LES): k and nuSgs – This model is called 'oneEqEddy' in V2.3.0*

- *SpalartAllmaras (LES): nuSgs and nuTilda*

Some files are available, e.g. epsilon, k and nuTilda, some files should be created by the user, e.g. R, nuSgs. Templates for these files can be also found in the examples of older versions of OpenFOAM®, e.g. 1.7.1.

Note: A missing R file can be created by OpenFOAM®. Open the turbulenceProperties file in the constant directory, set the `simulationType` *to* `RAS`*, and* `RASModel` *to* `kEpsilon`*. Run the command 'simpleFoam –postProcess –fun R' from terminal, it will create the turbulenceProperties:R file in the 0 directory. Then simply rename the file to 'R'.*

1.3. constant directory

In the *turbulenceProperties* file, the `simulationType` can be set as either `RAS`, `LES` or `laminar`. Then the corresponding sub-dictionary of the chosen simulation type needs to be defined. In this case, the sub-dictionary for RAS contains information about the chosen RAS model (kEpsilon), and the status of `turbulence` and `printCoeffs` are turned to on.

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

simulationType        RAS;

RAS
{
     RASModel         kEpsilon;
     turbulence       on;
     printCoeffs      on;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
```

Note: For the laminar model, set `turbulence` *and* `printCoeffs` *to* `off`*.*

1.4. system directory

Note: Since it is a steady state simulation, `endTime` *in controlDict shows the number of iterations instead of time and deltaT should be* `1`*, because it is the amount of increase in the iteration number.*

For the LRR model, discretization model for the new variable R needs to be specified. It is done through the *fvSchemes* file,

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

ddtSchemes
{
      default          steadyState;
}

gradSchemes
{
      default          Gauss linear;
}

divSchemes
{
```

```
      default          none;
      div(phi,U)       bounded Gauss linearUpwind gradf(U);
      div(phi,k)       bounded Gauss limitedLinear 1;
      div(phi,epsilon) bounded Gauss limitedLinear 1;
      div(phi,omega)   bounded Gauss limitedLinear 1;
      div(phi,v2)      bounded Gauss limitedLinear 1;
      div(phi,R)       bounded Gauss limitedLinear 1;
      div(R)           Gauss linear;
      div((nuEff*dev2(T(grad(U)))))  Gauss linear;
      div(nonlinearStress)     Gauss linear;
}

laplacianSchemes
{
      default      Gauss linear corrected;
}

interpolationSchemes
{
      default      linear
}

snGradSchemes
{
      default      corrected
}

wallDist
{
      Method          meshWave;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
```

Furthermore, *fvSolution* needs to be changed due to the new R parameter. The solver type for R is defined, in this case the solver used will be the same as the one for other variables (U, k, epsilon, omega).

2. Running simulation

```
>blockMesh
>simpleFoam
```

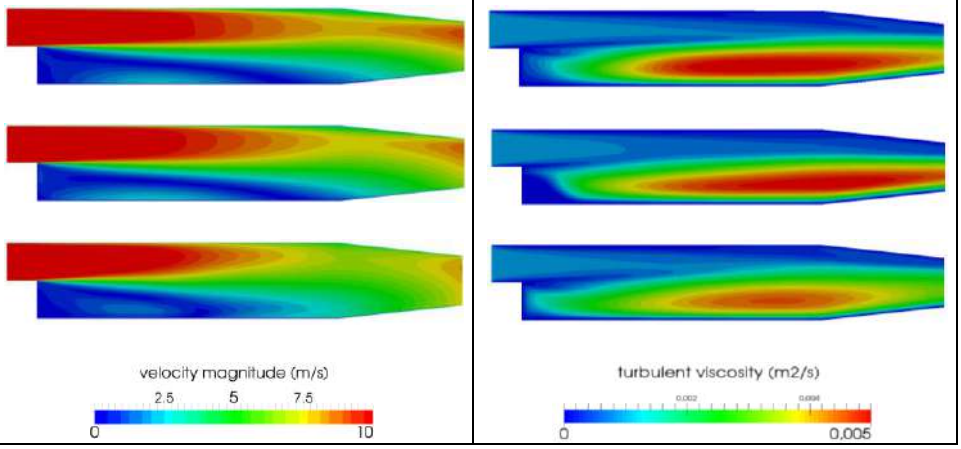
Note: When the solution converges, "`SIMPLE solution converged in … iterations`*" message will be displayed in the Shell window. If nothing happens and you do not see a message after a while (this is not the case in here, it converges after a short time), then you should check the residuals which are displayed in the Shell window manually (you should check* `initial residual` *values, it shows the difference between this iteration and the last one), if all of the* `Initial residual` *(see below) values are close to amounts you have set in the fvSolution then you can stop simulation (ctrl+c).*

```
Time = 298

smoothSolver:  Solving for Ux, Initial residual = 0.00013831, Final residual =
9.28001e-06, No Iterations 6
smoothSolver:  Solving for Uy, Initial residual = 0.000977894, Final residual =
6.73868e-05, No Iterations 6
GAMG:  Solving for p, Initial residual = 0.00192871, Final residual =
0.000174838, No Iterations 7
time step continuity errors : sum local = 0.000840075, global = 6.13868e-05,
cumulative = -0.193739
```

```
smoothSolver:  Solving for epsilon, Initial residual = 0.000175322, Final
residual = 1.138e-05, No Iterations 2
smoothSolver:  Solving for k, Initial residual = 0.000404928, Final residual =
2.99083e-05, No Iterations 2
ExecutionTime = 56.7 s  ClockTime = 57 s

SIMPLE solution converged in 298 iterations
```

3. Post-processing

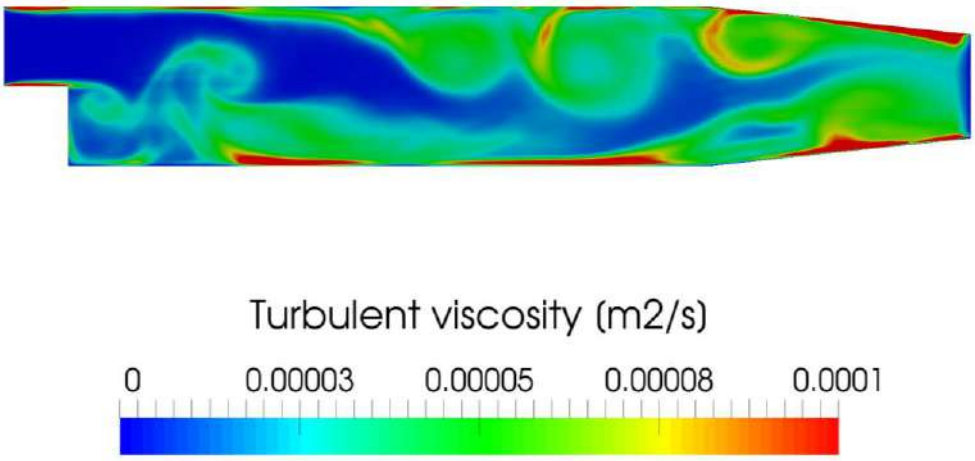
The simulation results are as follows (all simulations scaled to the same range):

RAS model	Velocity magnitude	Turbulent viscosity
kEpsilon		
kOmega		
LRR		

Velocity magnitude and turbulent viscosity for different RAS models

Tutorial Seven

Turbulence - Transient



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Large eddy simulation (LES)

In LES model, it is assumed that large eddies of the flow are dependent on the geometry, while the smaller eddies are more universal. One can then explicitly solve for the large eddies in a calculation by resolving them in a grid and implicitly account for the small eddies by using a sub grid-scale model (SGS model).

Mathematically, it is like separating the velocity field into a resolved and sub-grid part using a filter function. The resolved part of the field represents the large eddies, while the sub grid part of the velocity represents the small eddies whose effect on the resolved field is included through the sub grid-scale model. Formally, one may think of filtering as the convolution of a function with a filtering kernel G:

$$\bar{u}_i(\vec{x}) = \int G(\vec{x} - \vec{\xi}) u(\vec{\xi}) d\vec{\xi}$$

resulting in

$$u_i = \bar{u}_i + u'_i$$

Where $\bar{u}_i$ is the resolvable scale part and u'_i is the subgrid-scale part. However, most practical (and commercial) implementations of LES use the grid itself as the filter and perform no explicit filtering. The filtered equations are developed from the incompressible Navier-Stokes equations of motion:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\upsilon \frac{\partial u_i}{\partial x_j}\right)$$

Substituting in the decomposition $u_i = \bar{u}_i + u'_i$ and $p = \bar{p} + p'$ and then filtering the resulting equation gives the equations of motion for the resolved field:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\upsilon \frac{\partial \bar{u}_i}{\partial x_j}\right) + \frac{1}{\rho}\frac{\partial \tau_{ij}}{\partial x_j}$$

We have assumed that the filtering operation and the differentiation operation commute, which is not generally the case. It is thought that the errors associated with this assumption are usually small, though filters that commute with differentiation have been developed. The extra term $\partial \tau_{ij}/\partial x_j$ arises from the non-linear advection terms, due to the fact that:

$$u_j \frac{\partial u_i}{\partial x_j} \neq \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j}$$

and hence

$$\tau_{ij} = \bar{u}_i \bar{u}_j - \overline{u_i u_j}$$

Similar equations can be derived for the sub grid-scale field. Sub grid-scale turbulence models usually employ the Boussinesq hypothesis, and seek to calculate (the deviatoric part of) the SGS stress using:

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\mu_t \bar{S}_{ij}$$

where $\bar{S}_{ij}$ is the rate-of-strain tensor for the resolved scale defined by

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)$$

and υ_t is the subgrid-scale turbulent viscosity. Substituting into the filtered Navier-Stokes equations, we then have:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left([\upsilon + \upsilon_t]\frac{\partial \bar{u}_i}{\partial x_j}\right)$$

where we have used the incompressibility constraint to simplify the equation and the pressure is now modified to include the trace term $\tau_{kk}\delta_{ij}/3$.

2. Smagorinsky-Lilly model

A simple model for Sub grid-scale model is the Smagorinsky model, which can be summarized as:

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2(C_s\Delta)^2|\bar{S}|S_{ij}$$

In the Smagorinsky-Lilly model, the eddy viscosity is modeled by

$$\mu_{sgs} = \rho(C_s\Delta)^2|\bar{S}|$$

Where the filter width is usually taken to be

$$\Delta = (Volume)^{1/3}$$

and

$$\bar{S} = \sqrt{2S_{ij}S_{ij}}$$

The effective viscosity is calculated from

$$\mu_{eff} = \mu_{mol} + \mu_{sgs}$$

The Smagorinsky constant usually has the value: $C_s = 0.1 - 0.2$

pisoFoam – pitzDaily

Simulation

Use the pisoFoam solver, run a backward facing step case for 0.2 s with different turbulence models:

- Smagorinsky (LES)
- kEqn (LES)
- kEpsilon (RAS)

Objectives

- Understanding turbulence models
- Understanding the difference between transient and steady state simulation
- Finding appropriate turbulence model

Data processing

Display the results of U and the turbulent viscosity in two separate contour plots at three different time steps. Compare with steady state simulation (Tutorial Six).

1. Pre-processing

1.1. Copy tutorial

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

`$FOAM_TUTORIALS/incompressible/pisoFoam/les/pitzDaily`

1.2. 0 directory

Set the turbulence model initial and boundary values.

Note: For different turbulent models, different files should be modified (check Tutorial Six).

1.3. constant directory

As mentioned in Tutorial Six, in *turbulenceProperties* the turbulent model type has to be set. The `simulationType` can be changed to `LES` or `RAS`. Depending on which type is selected, the corresponding sub-dictionary needs to be specified. Below is the *turbulenceProperties* file for the kEqn model which is an LES model.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
simulationType  LES;

LES
{
        LESModel        kEqn;
        turbulence      on;
        printCoeffs     on;
        delta           cubeRootVol;

        dynamicKEqnCoeffs
        {
            filter      simple;
        }

        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        PrandtlCoeffs
        {
            delta           cubeRootVol;
            cubeRootVolCoeffs
            {
                deltaCoeff      1;
            }

            smoothCoeffs
            {
                delta           cubeRootVol;
                cubeRootVolCoeffs
                {
                    deltaCoeff      1;
                }

                maxDeltaRatio   1.1;
            }

            Cdelta          0.158;
        }

        vanDriestCoeffs
```

```
        {
            delta           cubeRootVol;
            cubeRootVolCoeffs
            {
                deltaCoeff      1;
            }

            smoothCoeffs
            {
                delta           cubeRootVol;
                cubeRootVolCoeffs
                {
                    deltaCoeff      1;
                }

                maxDeltaRatio   1.1;
            }

            Aplus           26;
            Cdelta          0.158;
        }

        smoothCoeffs
        {
            delta           cubeRootVol;
            cubeRootVolCoeffs
            {
                deltaCoeff      1;
            }

            maxDeltaRatio   1.1;
        }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
```

2. Running simulation

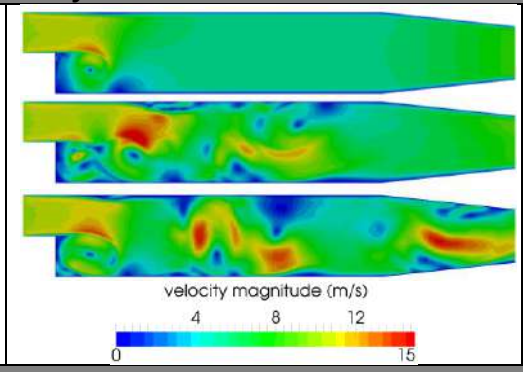
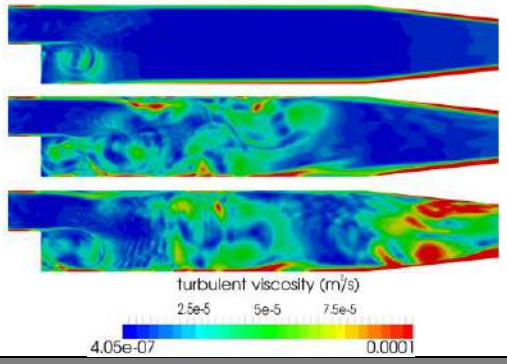
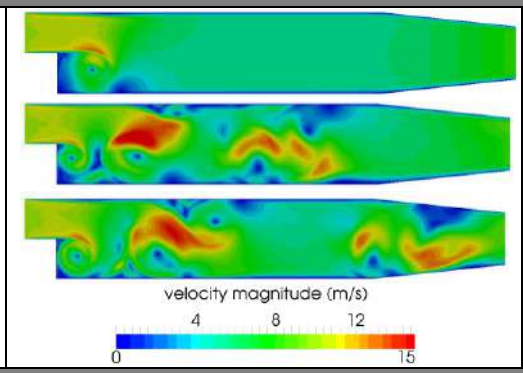
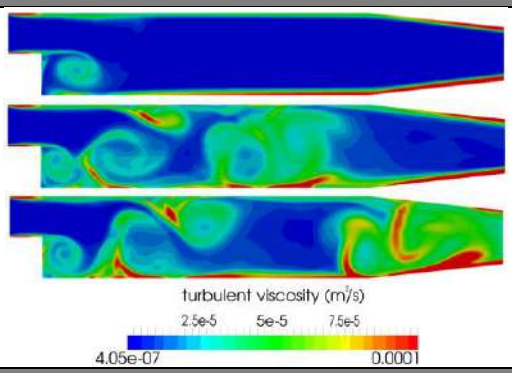
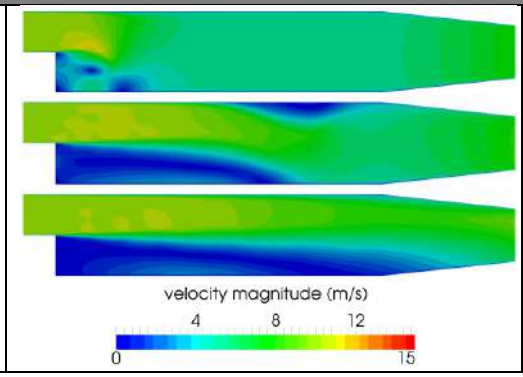
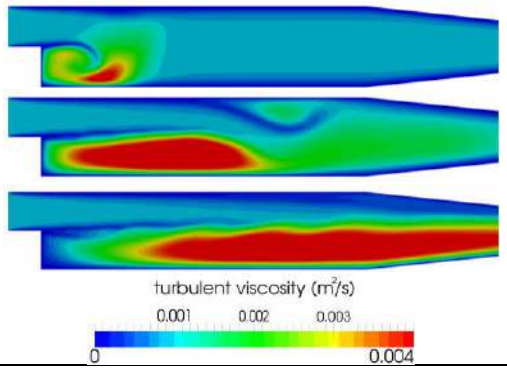
```
>blockMesh
>pisoFoam
```

3. Post-processing

The simulation results are as follows:

For the kEpsilon model after 0.2 s the results are similar to the steady state simulation. Therefore, it can be assumed it has reached the steady state. Other models do not have a steady situation and are fluctuating all the time, so they require averaging for obtaining steady state results.

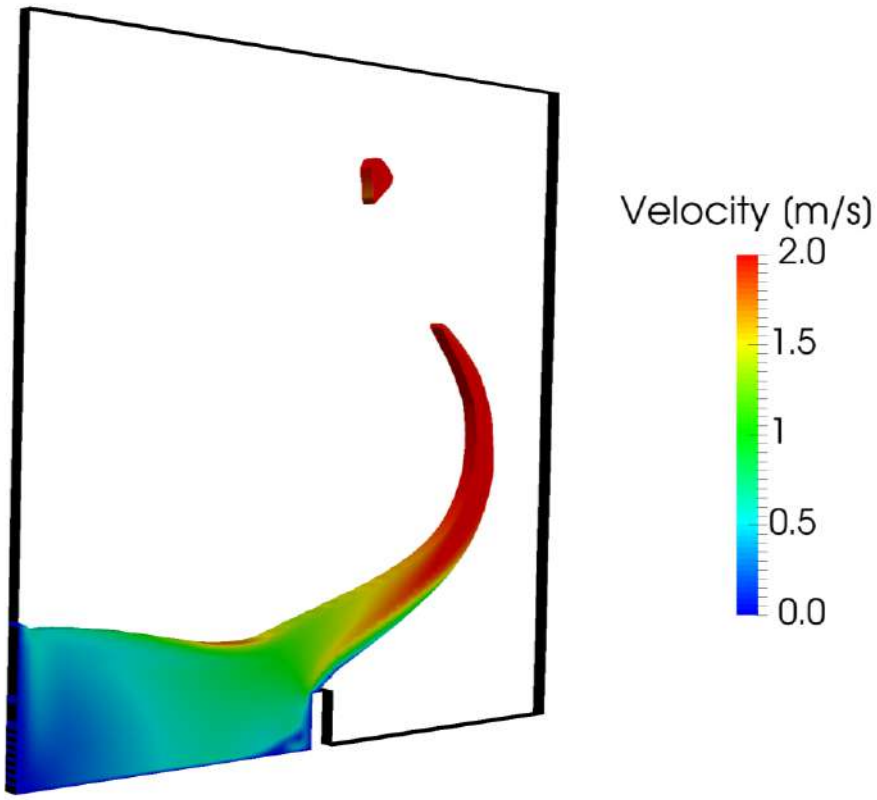
kEpsilon and other RAS models use averaging to obtain the turbulence values, but LES does not include any averaging by default. Therefore, LES simulations should use a higher grid resolution (smaller cells) and smaller time steps (for reasonable Co number). Contour plots or other LES results should be presented time averaged over reasonable number of time steps (not done in this tutorial).

	Velocity magnitude	Turbulent viscosity
Smagorinsky		
0.01 s		
0.05 s		
0.2 s		
kEqn		
0.01 s		
0.05 s		
0.2 s		
kEpsilon		
0.01 s		
0.05 s		
0.2 s		

Comparison of different turbulent models for transient simulation.

Tutorial Eight

Multiphase



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

In this tutorial we are going to solve a problem of dam break using the *interFoam* solver. The main feature of this problem is flow of water and air separated by a sharp interface. Before starting, let's cover some of the basics of multiphase flow.

1. Multiphase flow

Multiphase flow is simultaneous flow of materials in different phases. There can be multiple components present in each phase. The common types of multiphase flows are: gas-liquid, gas-solid, liquid-solid, liquid-liquid and three-phase flows.

Multiphase flow can be further categorized based on the visual appearance of the flow into separated, mixed or dispersed flow. In dispersed flow, one phase exist as a continuous fluid, while all other phases act as discontinuous particles flowing through the continuous fluid. In mixed flow regions, dispersed particles as well as semi-continuous interfaces exist together.

So why is multiphase flow important? Multiphase flow is present in many industrial processes, such as bubble columns, absorption, adsorption and stripping columns. Modeling of multiphase flow can help maximizing contact between different phases, hence increasing the efficiency of the process.

2. Modeling approaches

Modeling of multiphase flow can be extremely complex, due to possible flow regime transitions. To simplify the matter, different modeling approaches can be adopted and they generally fall into two categories: lagrangian and Eulerian. In the case of dispersed configuration, Lagrangian approach is more suitable. This involves tracking individual point particles during its movement. The other approach is the Eulerian approach, which observes fluid behavior in a given control volume.

Below we will cover some common modeling approaches of multiphase flow.

2.1. Euler-Euler approach (Multi-fluid model)

All phases are treated as continuous in the Euler-Euler approach. This approach is suitable for separated flows where each phase behaves as a continuum, rather than being discrete. The phases interact through the drag and lift forces acting between them, as well as through heat and mass transfer. The Euler-Euler approach is also capable of modeling dispersed flow, where we are interested in the overall motion of particles rather than tracking individual particles.

In the Euler-Euler approach, we introduce the concept of phasic volume fractions. These fractions are assumed to be continuous functions of space and time, with their sum equal to one. For each phase, a set of conservation equations for mass, momentum and energy is solved individually; in addition, a transport equation for the volume fraction is solved. Coupling between the phases is achieved through a shared pressure and interphase exchange coefficients.

2.2. Eddy Interaction Model

In the Eddy Interaction Model, each particle interacts with a succession of eddies. The fluid motion of the particle is characterized by three parameters: i) eddy velocity, ii) eddy lifetime, iii) eddy length. It follows the particle-tracking Lagrangian approach.

The eddy lifetime (t_e) and eddy length scale (l_e) are estimated from the local turbulence properties. From the length scale and the particle velocity, one can calculate the eddy transit time (t_c), i.e. the time taken for a particle to cross the eddy. The particle is then assumed to interact with the eddy for a time which is the minimum of the eddy life time and the eddy transit time.

$$t_{int} = \min(t_e, t_c)$$

During that interaction the fluid fluctuating velocity is kept constant and the discrete particle is moved with respect to its equation of motion. Then a new fluctuating fluid velocity is sampled and the process is repeated.

2.3. Volume of Fluid (VOF) method

VOF method belongs to the Eulerian class of modeling approach. It is based on the idea of **fraction function C**. Fraction function indicates whether a chosen phase is present inside the control volume. If C=1, the control volume is completely filled with the chosen phase; if C=0, the control volume is filled with a different phase. A value between 0 and 1 indicates that the interface between phases is present inside the control volume. It is important in VOF method that the flow domain is modeled on a fine grid, i.e. the interface should be resolved.

The focus of the VOF method is to track the interface between phases. To do this, the transport equations are solved for mixture properties, assuming that all field variables are shared between the phases. Then an advection equation for the fraction function C is solved. The discretization of the fraction function equation is crucial for obtaining a sharp interface.

The multiphase flow in this tutorial is analysed using the *interFoam* solver. Here is a brief explanation of the solver below.

3. interFoam solver

interFoam is suitable for solving multiphase flow between 2 incompressible, isothermal immiscible fluids. It is based on the Volume of Fluid (VOF) approach.

interFoam – damBreak

Simulation

Use the interFoam solver to simulate breaking of a dam for 2s.

Objectives

- Understanding how to set viscosity, surface tension and density for two phases

Data processing

See the results in ParaView.

1. Pre-processing

1.1. Copy tutorial

Copy tutorial from the following folder to your working directory:

```
$FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak/damBreak
```

1.2. 0 directory

In the 0 directory the following files exist:

```
alpha.water  alpha.water.orig  p_rgh  U
```

In the alpha.water.orig and p_rgh files the initial values and also boundary conditions for phase water and also pressure are set. Copy alpha.water.orig to alpha.water (remember: the *.orig files are back up files, and solvers do not use them). E.g. in alpha.water:

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

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    leftWall
    {
        type            zeroGradient;
    }

    rightWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }

    defaultFaces
    {
        type            empty;
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Note: The `inletOutlet` and the `outletInlet` boundary conditions are used when the flow direction is not known. In fact, these are derived types and are a combination of two different boundary types.

- `inletOutlet`: *When the flux direction is toward the outside of the domain, it works like a zeroGradient boundary condition and when the flux is toward inside the domain it is like a fixedValue boundary condition.*

- `outletInlet`: *This is the other way around, if the flux direction is toward outside the domain, it works like a fixedValue boundary condition and when the flux is toward inside the domain, it is like a zeroGradient boundary condition.*

E.g. if the velocity field outlet is set as `inletOutlet` *and the* `inletValue` *is set to* `(0 0 0)`, *it avoids backflow at the outlet! The "*`inletValue`*" or "*`outletValue`*" are values for* `fixedValue` *type of these boundary conditions and "*`value`*" is a dummy entery for OpenFOAM® for finding the variable type. Using* `(0 0 0)`, *OpenFOAM® understands that the variable is a vector.*

1.3. constant directory

In the *transportProperties* file the properties of two phases can be set under each phase sub-dictionary, e.g. `water` or `air`:

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

phases (water air);

water
{
    transportModel  Newtonian;
    nu              1e-06;
    rho             1000;
}

air
{
    transportModel  Newtonian;
    nu              1.48e-05;
    rho             1;
}

sigma           0.07;

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

In both phases the coefficients for different models of viscosity are given, e.g. `nu` and `rho`. Depending on which model is selected, the coefficients from the corresponding sub-dictionary are read. The selected model is `Newtonian`, only the `nu` coefficient is used.

`sigma` is the surface tension between two phases, in this example it is the surface tension between air and water.

Checking the g file, the gravitational field and also its direction are defined, it is `9.81` m/s^2 in the negative y direction.

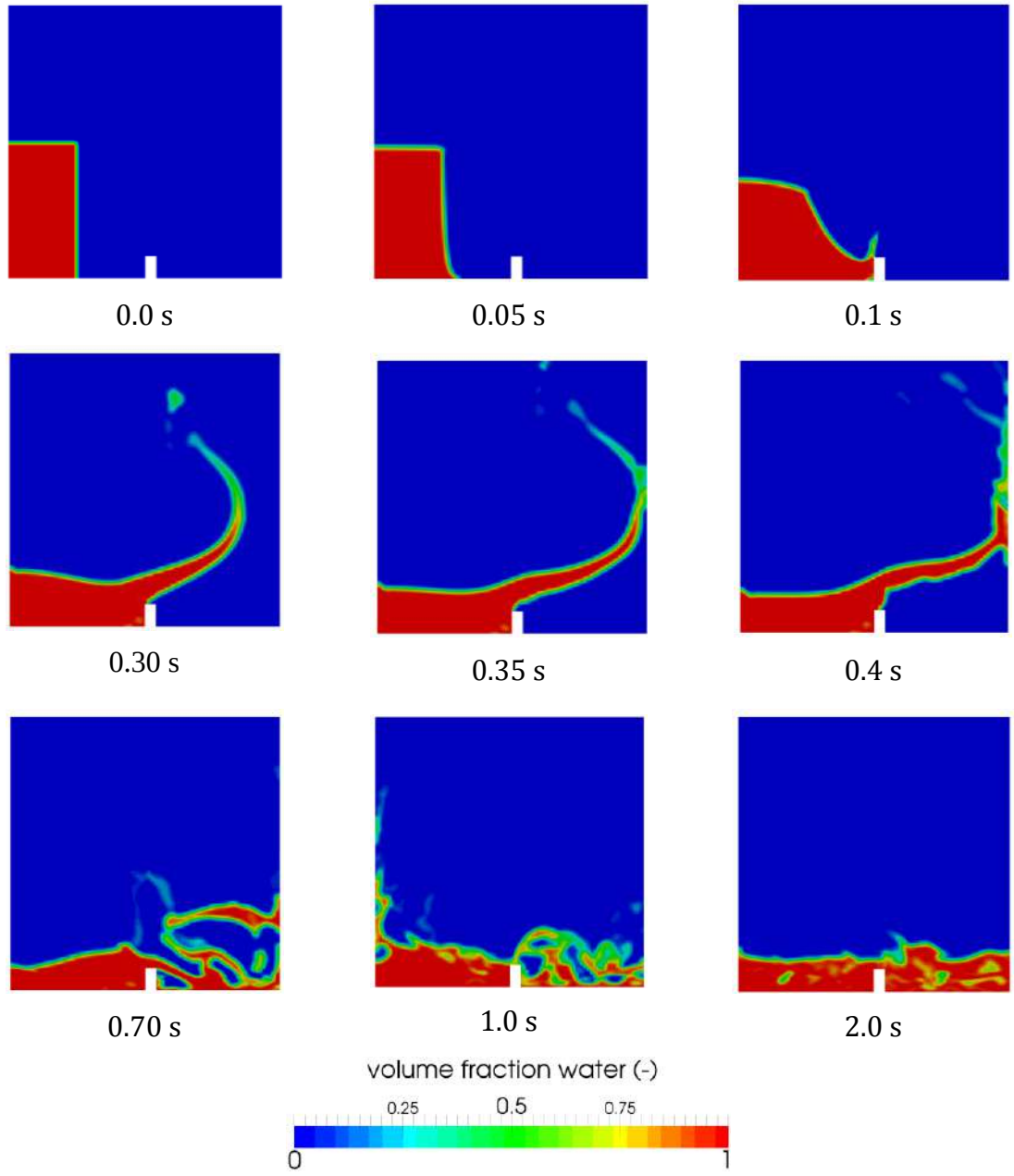
```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//


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

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

2. Running simulation

```
>blockMesh
>setFields
>interFoam
```

3. Post-processing

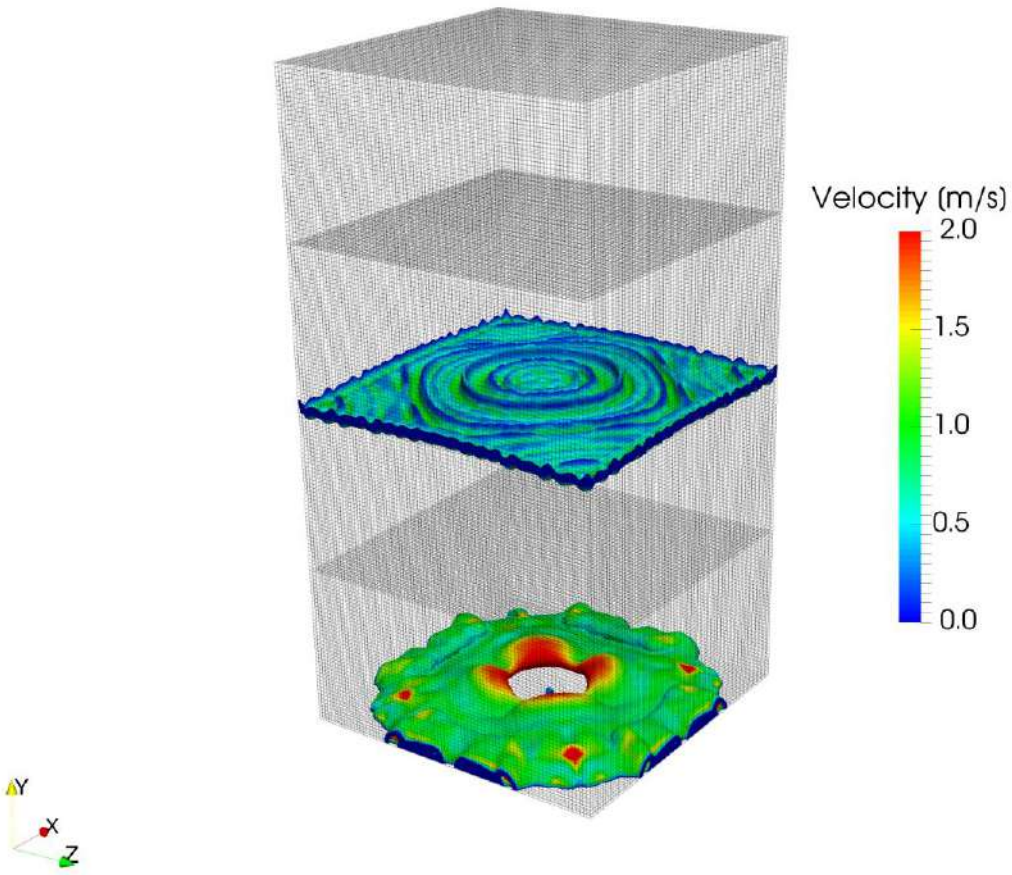
The simulation results are as follows (these are not the results for the original mesh, but a 2x refined mesh):



Contours of the water volume fraction at different time steps

Tutorial Nine

Parallel Processing



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

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

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

In this tutorial we will analyze compressible fluid flow in OpenFOAM®. Parallel processing is utilized to speed up the simulation. In this introduction part, theory behind compressible flow, solvers for compressible flow and parallel computing will be explained in detail.

1. Introduction to compressible flow

So far we have only considered incompressible fluid flows, however in many situations; there may be a significant change in the density. One example of compressible flow is the flow through a diverging-converging nozzle. Compressibility becomes dominant in flows when the Mach number is greater than about 0.3. The Mach number is defined as follows:

$$Ma = \frac{u}{c} = \frac{local\ velocity}{speed\ of\ sound}$$

When a fluid flow is compressible, temperature and pressure are affected strongly by variations in density. It is therefore important to take into account the linkage between pressure, temperature and density in compressible flow, usually by applying an equation of state from thermodynamics (e.g. the ideal gas equation).

2. Compressible flow solvers

There are two general types of solution schemes for compressible flow: pressure-based and density-based.

2.1. Pressure-based solvers

This type of solver was historically derived from the solution approach used on incompressible flows. They solve for the primitive variables. The discretized momentum and energy equations are used to update velocities and energy. The pressure is obtained by applying a pressure-correction algorithm on the continuity and momentum equations. Density is then calculated from the equation of state.

2.2. Density-based solvers

Density-based solvers are suitable for solving the conserved variables. Similar to pressure-based solvers, the conversed velocity and energy terms are updated from the discretized momentum and energy equations. We can then solve for density from the continuity equation, afterwards we use the equation of state to update the pressure.

In general, density based solvers are more suitable for high speed compressible flows with shocks. This is because density based solvers solve for conserved quantities across the shock, so the discontinuities will not affect the results.

3. Parallel computing

Imagine if we need to tackle a complex CFD problem that involves complex geometry, multiphase flow, turbulence and reaction, how do we adopt a methodical computational approach to save time and cost? This is when parallel computing

comes in. Parallel computing is defined as the simultaneous use of more than one processor to execute a program. The geometry of the domain will be partitioned into sub-domains, with each sub-domain assigned to a single processor. Furthermore data and computational tasks will be partitioned and divided amongst the processors. This step is known as **domain decomposition**.

Parallel computing can be carried out in two ways. One is done on a single computer with multiple internal processors, known as a **Shared Memory Multiprocessor**. The other way is achieved through a series of computers interconnected by a network, known as a **Distributed Memory Multicomputer**.

3.1. Shared versus distributed memory

	Shared Memory Multiprocessor	***Distributed Memory Multicomputer***
Memory	Data is saved in a global memory that can be accessed by all processors	Each computer has a local memory and a processor can only access its local memory
Data transfer between processors	The sender processor simply needs to write the data in a global variable and the receiver can read it	Message is sent explicitly from one computer to another using a message passing library, e.g. Message Passing Interface (MPI)

In OpenFOAM® the application of parallel computing can be executed using the *decomposePar* command. This allows the solver to be run on multiple processors. The workflow of parallel computation in OpenFOAM® is summarized below:

- Division of the mesh into sub-domains
- Running of the solver in parallel
- Reconstruction of the meshes and connecting the results

compressibleInterFoam – depthCharge3D

Simulation

Use the compressibleInterFoam solver, simulate the example case for 0.5 s.

Objectives

- Understanding the difference between incompressible and compressible solvers
- Understanding parallel processing and different discretization methods

Data processing

Investigate the results in ParaView.

1. Pre-processing

1.1. Copy tutorial

Copy the tutorial from following directory to your working directory:

`$FOAM_TUTORIALS/multiphase/compressibleInterFoam/laminar/depthCharge3D`

1.2. 0 directory

Create a 0 directory and copy all the files from the 0.orig directory to the 0 directory.

1.3. constant directory

Phases and common physical properties of the two phases are set in the *thermophysicalProperties* file. Individual phase properties are set in *thermophysicalProperties.phase* files, e.g. *thermophysicalProperties.air*.

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

phases (water air);

pMin            10000;

sigma           0.07;

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

1.4. system directory

The decomposeParDict file includes the parallel settings, such as the number of domains (partitions) and also how the domain is going to be divided into these subdomains for parallel processing.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
numberOfSubdomains 4;

method          hierarchical;

simpleCoeffs
{
    n               ( 1 4 1 );
    delta           0.001;
}

hierarchicalCoeffs
{
    n               ( 1 4 1 );
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

OpenFOAM® v1712: In this file just the coefficients for hierarchical method are listed!

`numberOfSubdomains` should be equal to the number of cores used. `method` should show the method to be used. In the above example, the case is simulated with the `hierarchical` method and `4` processors.

If the `simple` method is being used, the parameter `n` must be changed accordingly. The three numbers `(1 4 1)` indicate the number of pieces the mesh is split into in the x, y and z directions, respectively. Their multiplication result should be equal to `numberOfSubdomains`.

If the `hierarchical` method is being used, these parameters and also the order in which the mesh should be split up in each direction should be provided.

If the `scotch` method is being used, then no user-supplied parameters are necessary except for the number of subdomains.

There is also a parameter delta, known as the cell skew factor. This factor is set to a default value of `0.001`, and measures to what extent skewed cells should be accounted for.

Note: In order to check the quality of the mesh, the checkMesh tool can be used (run it from main case directory). If the message "`Mesh OK`" is displayed – the mesh is fine and no corrections need to be done.

If the mesh fails in one or more tests, try to recreate or refine the mesh for a better mesh quality (less non-orthogonally and skewness). If the error exists after correcting the mesh then a possible course of action is to increase the delta parameter (for example: to 0.01) and then rerun the blockMesh and checkMesh tools.

If non-orthogonal cells exist in a mesh, another option is using non-orthogonal corrections in the fvSolution file in the algorithm sub-dictionary (e.g. `PIMPLE` *or* `PISO`*). Usually using* `1` *or* `2` *as* `nNonOrthogonalCcorrectors` *is enough.*

2. Running simulation

```
>blockMesh
>setFields
```

For running the simulation in parallel mode the computing domain needs to be divided into subdomains and a core should be assigned to each subdomain. This is done by following command:

```
>decomposePar
```

This decomposes the mesh according to the supplied instructions. One possible source of error is the product of the parameters in `n` does not match up to the number of the subdomains. This appears for the simple and hierarchical methods.

After executing this command four new directories will be made in the simulation directory (processor0, processor1, processor2 processor3), and each subdomain calculation will be saved in the respective processor directory.

Note: When the domain is divided to subdomains in parallel processing new boundaries are defined. The data should be exchanged with the neighbor boundary, which it is connected to in the main domain.

```
>mpirun -np <No of cores> solver -parallel > log
```

`<No of cores>` is the number of cores being used. `solver` is the solver for this simulation. For example, if `4` cores are desired, and the solver is compressibleInterFoam following command is used:

```
>mpirun -np 4 compressibleInterFoam -parallel > log
```

`> log` is the filename for saving the simulation status data, instead of printing them to the screen. For checking the last information which is written to this file the following command can be used during the simulation running:

```
>tail -f  log
```

Note: Before running any simulation, it is important to run the top command (type the 'top' command in the terminal), to check the number of cores currently used on the machine. Check the load average. This is on the first line and shows the average number of cores being used. There are three numbers displayed, showing the load averages across three different time scales (one, five and 15 minute respectively).

Add the number of cores you plan to use to this number – and you will get the expected load average during your simulation. This number should be less than the total number of cores in the machine – or the simulation will be slowed or the machine will crash (if you run out of memory). If you are running on a multi user server it is recommended to leave at least a few cores free, to allow for any fluctuations in the machine load.

Note: top command execution can be interrupted by typing q (or ctrl+c)

The simulation can take several hours, depending on the size of the mesh and time step size.

3. Post-processing

For exporting data for post processing, at first all the processors data should be put together and a single combined directory for each time step was created. By executing the following command all the cores data will be combined and new directories for each time step will be created in the simulation main directory:

```
>reconstructPar
```

Convert the data to ParaView format:

```
>foamToVTK
```

Note: To do the reconstruction or foamToVTK conversion from a start time until an end time the following flags can be used:

```
>reconstructPar -time [start time name, e.g. 016]:[end time name, e.g. 020]
```

```
>foamToVTK -time [start time name, e.g. 016]:[end time name, e.g. 020]
```

Using above commands without entering end time will do the reconstruction or conversion from start time to the end of available data:

```
>reconstructPar -time [start time name, e.g. 016]:
```

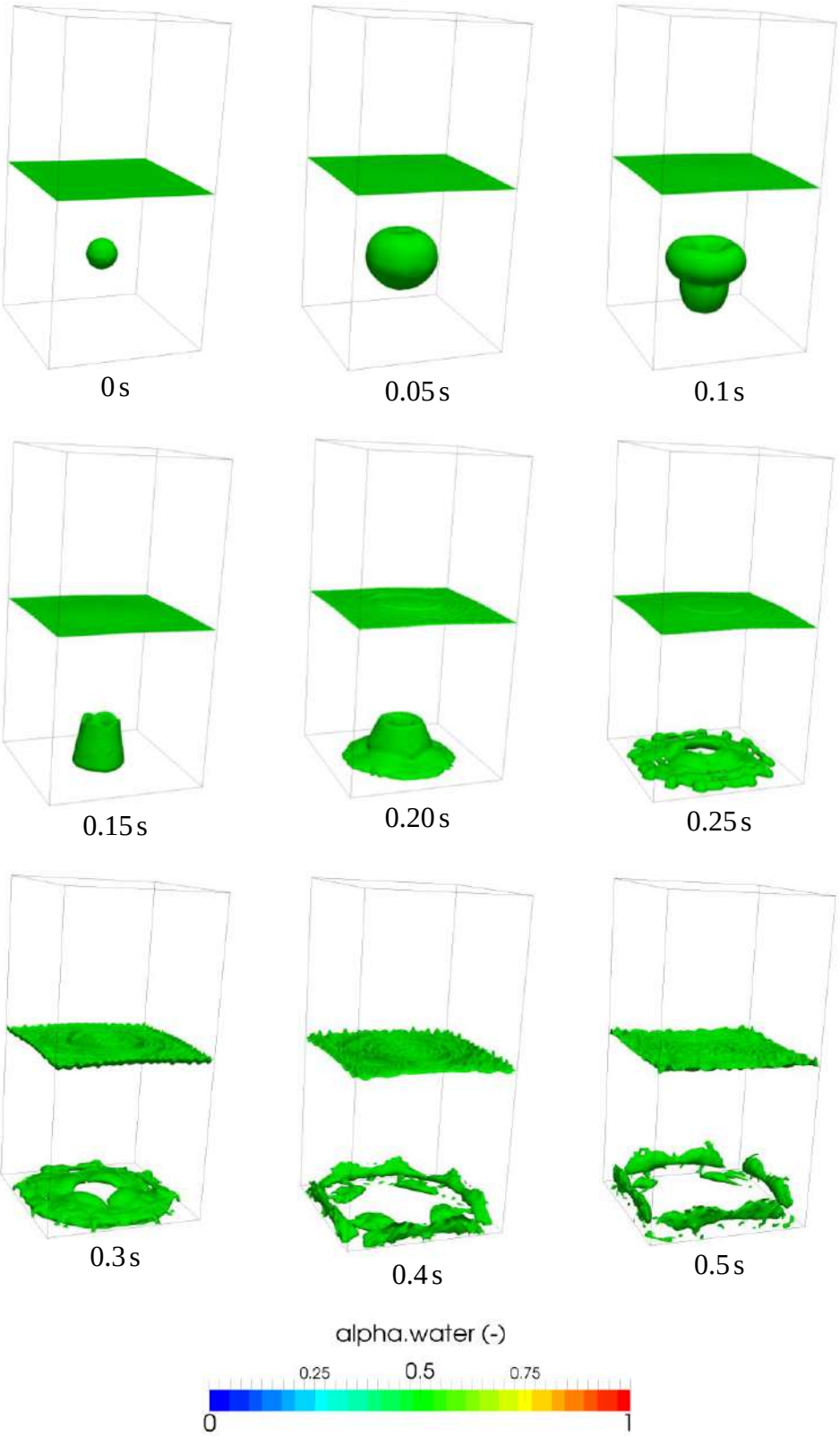
```
>foamToVTK -time [start time name, e.g. 016]:
```

For reconstructing or converting only one time step the commands should be used without end time and ":":

```
>reconstructPar -time [start time name, e.g. 016]
```

```
>foamToVTK -time [start time name, e.g. 016]
```

The simulation results are as follows:



3D depth charge, alpha = 0.5 iso-surfaces, parallel simulation

4. Manual method

4.1 Case set-up and running simulation

The manual method for decomposition is slightly different from the other three. In order to use it:

After running the blockMesh and setFields utilities, set the decomposeParDict file as any other simulation. For decomposition method, choose either simple, hierarchical or scotch. Set the number of cores to the same number which is going to be used for manual.

```
>decomposePar -cellDist
```

Once the decomposition is done, check the cellDecomposition file in the constant directory. It should have a format similar to:

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

1024000
(
0
0
0
0
0
0
0
0
0
0
0
0
1
1
1
1
1
1
1
1
1
1
1
1
1 ...)
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * *//
```

Note: If the above output is not displayed, but a stream of NUL characters, your text editor is probably printing binary. To fix this, open system/controlDict, and change the `writeFormat` *field from* `binary` *to* `ascii`.

The first number `n` after the header, but before the opening brackets, `1024000` in this example, refers to the number of points in the mesh. Within the brackets, `n` lines follow. Each line contains one number between `0` and `n-1`, where `n` is the number of cores to be used for the computation. This number refers to the core being used to compute the corresponding cell in the points file in the constant directory. For example, if the second line in the points file brackets reads `(0.125 0 0)` and the

second line in the cellDecomposition directoy reads `0`, this means that the cell `(0.125 0 0)` will be processed by processor `0`.

This cellDecomposition file can now be edited. Although this can be done manually, it is probably not feasible for any sufficiently large mesh. The process must thus be automated by writing a script to populate the cellDecomposition file according to the desired processor breakdown.

When the new file is ready, save it under a different name:

```
>cp cellDecomposition manFile
```

Now, edit the decomposeParDict file. Select decomposition method `manual`, and for the `dataFile` field in the manual coeffs range, specify the path to the file which contains the manual decomposition. Note that OpenFOAM® searches in the constant directory by default, in case relative paths are being used:

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

numberOfSubdomains 4;

method          manual;

simpleCoeffs
{
    n               ( 1 4 1 );
    delta           0.001;
}

hierarchicalCoeffs
{
    n               ( 1 4 1 );
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "manFile";
}

distributed     no;

roots           ( );

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

Run the simulation as usual.

4.2. Visualizing the processor breakdown

It may be interesting to visualize how exactly OpenFOAM® breaks down the mesh. This can be easily visualized using ParaView. After running the simulation, but before running the reconstructPar command, repeat the following for each of the processor directories:

```
>cd processor<n>
```

where `n` is the processor number

```
>foamToVTK
```

convert the individual processor files to VTK, next, open ParaView:

```
>paraview &
```

For each of the processor directories, perform the following steps:

- Open the VTK files in the relevant processor directory

- Double click them to open them and click on “Apply”

- The part of the mesh decomposed by that core will appear, in grey.

- Change the color in the drop-down menus in the toolbar. This is to ensure that each individual part can be easily seen.

Once this is done for all processors, the entire mesh will appear. However, the processor regions can now easily be seen in a different color.

In order to save this, there are two options. The first option is to take a screenshot:

File > Save a screenshot

The second option is to save the settings and modifications as a ParaView state file.

File > Save State

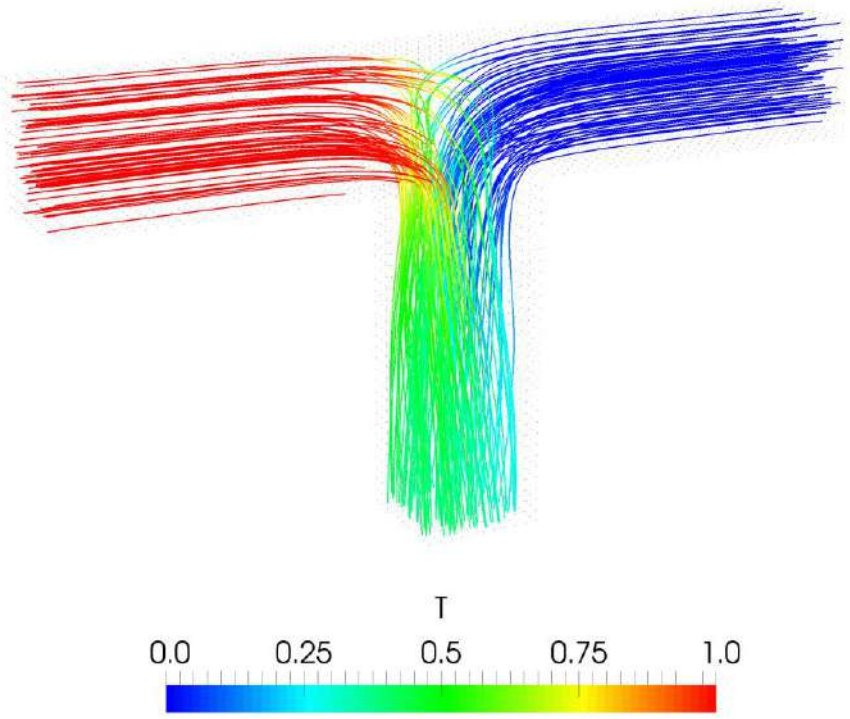
The current settings and modifications can then be easily recovered by:

File > Load State

Saving the state allows changes to be made afterwards. Saving a screenshot keeps only a picture, while losing the ability to make changes after exiting ParaView. Doing both is recommended.

Tutorial Ten

Residence Time Distribution



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:
- Bahram Haddadi
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

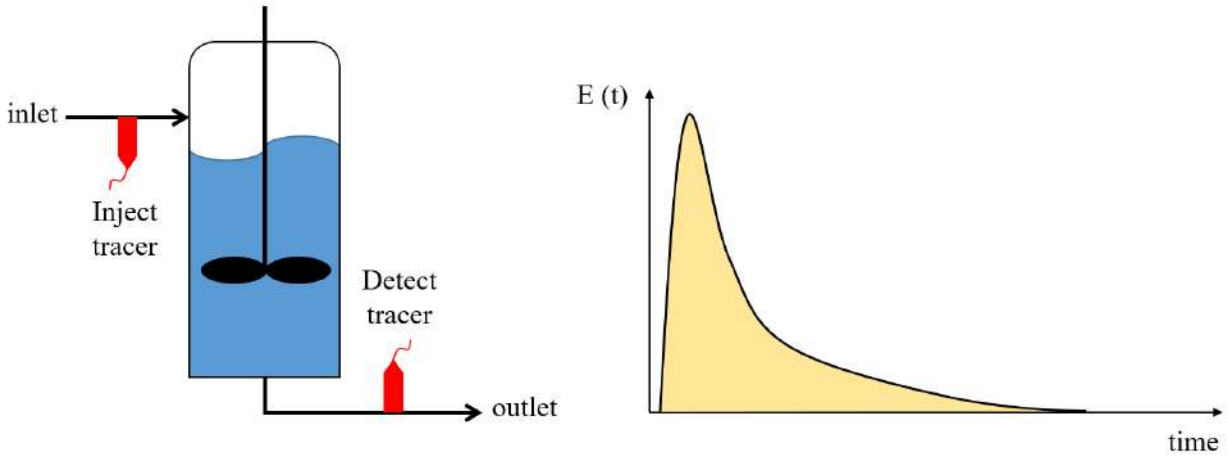
Background

In this tutorial we will carry out Residence Time Distribution (RTD) analysis of fluid flow through a T-junction pipe.

1. Residence Time Distribution (RTD)

Residence time distribution is a probability distribution function that provides information about the amount of time a tracer element spends within a process unit, such as a reactor or a column. RTD analysis is important because in almost all real-life processes, the mixing is not ideal and chemical engineers will need RTD to analyze the real mixing characteristics inside for example, a continuously stirred reactor. They can also use RTD analysis and to obtain information about the flow pattern, back mixing and bypassing behavior of a process unit.

2. Tracer Analysis



Tracer analysis and RTD distribution of an ideal process

Sometimes a radioactive tracer is usually used to determine RTD of a process unit. Based on the above diagram, first the tracer is injected into the inlet, and then the exit tracer concentration, $C(t)$, is measured at regular time intervals. This allows the exit age distribution, $E(t)$, to be calculated.

$$E(t) = \frac{C_T(t)}{\int_0^\infty C_T(t)\,dt} = \frac{Tracer\ concentration\ at\ time\ t}{Total\ tracer\ concentration}$$

It is clear from the above equation that the fraction of tracer molecules exiting the reactor that have spent a time between t and $t + dt$ in the process unit is $E(t)dt$. Since all tracer elements will leave the unit at some point, RTD satisfies the following relationship:

$$\int_0^\infty E(t)\,dt = 1$$

simpleFoam & scalarTransportFoam – TJunction

Simulation

Use the simpleFoam and scalarTransportFoam to simulate the flow through a square cross section T pipe and calculate RTD (Residence Time Distribution) for both inlets using a step function injection:

- Inlet and outlet cross sections: 1×1 m^2
- Gas in the system: air at ambient conditions
- Operating pressure: 10^5 Pa
- Inlet 1: 0.1 m/s
- Inlet 2: 0.2 m/s

Objectives

- Understanding RTD calculation using OpenFOAM®
- Using multiple solver for a simulation

Data processing

Plot the step response function and the RTD curve.

1. Pre-processing

1.1. Copy tutorial

Copy the following tutorial to your working directory as a base case:

`$FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily`

1.2. 0 directory

Update p, U, nut, nuTilda, k and epsilon files with the new boundary conditions, e.g. U:

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

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

internalField   uniform (0 0 0);

boundaryField
{
    inlet_one
    {
        type            fixedValue;
        value           uniform (0.1 0 0)
    }
    inlet_two
    {
        type            fixedValue;
        value           uniform (-0.2 0 0)
    }
    outlet
    {
        type            zeroGradient;
    }
    walls
    {
        type            fixedValue;
        value           uniform (0 0 0)

    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

1.3. constant directory

Check turbulenceProperties file for the turbulence model (`kEpsilon`).

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType  RAS
RAS
{
       RASModel         kEpsilon;

       turbulence       on;

       printCoeffs      on;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

1.4. system directory

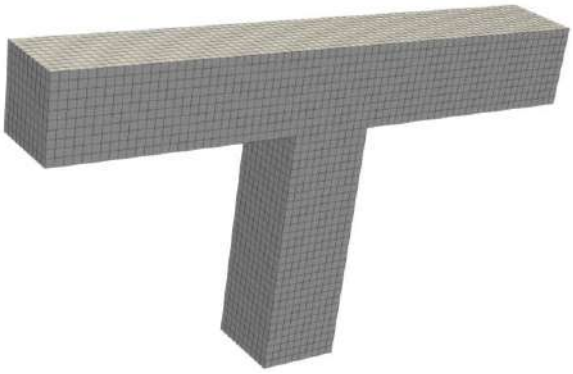
Edit the blockMeshDict to create an appropriate geometry.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 1.0;

vertices
(
```

```
    (0 4 0) // 0
    (0 3 0) // 1
    (3 3 0) // 2
    (3 0 0) // 3
    (4 0 0) // 4
    (4 3 0) // 5
    (7 3 0) // 6
    (7 4 0) // 7
    (4 4 0) // 8
    (3 4 0) // 9
    (0 4 1) // 10
    (0 3 1) // 11
    (3 3 1) // 12
    (3 0 1) // 13
    (4 0 1) // 14
    (4 3 1) // 15
    (7 3 1) // 16
    (7 4 1) // 17
    (4 4 1) // 18
    (3 4 1) // 19

);
blocks
(
    hex (0 1 2 9 10 11 12 19) (10 30 10) simpleGrading (1 1 1)
    hex (9 2 5 8 19 12 15 18) (10 10 10) simpleGrading (1 1 1)
    hex (8 5 6 7 18 15 16 17) (10 30 10) simpleGrading (1 1 1)
    hex (2 3 4 5 12 13 14 15) (30 10 10) simpleGrading (1 1 1)
);
edges
(
);
patches
(
    patch inlet_one
    (
        (0 10 11 1)
    )
    patch inlet_two
    (
        (7 6 16 17)
    )
    patch outlet
    (
        (4 3 13 14)
    )
    wall walls
    (
        (0 1 2 9)
        (2 5 8 9)
        (5 6 7 8)
        (2 3 4 5)
        (10 19 12 11)
        (19 18 15 12)
        (18 17 16 15)
        (15 14 13 12)
        (0 9 19 10)
        (9 8 18 19)
        (8 7 17 18)
        (2 1 11 12)
        (3 2 12 13)
        (5 4 14 15)
        (6 5 15 16)
    )
);

mergePatchPairs
(
);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

2. Running simulation

```
>blockMesh
```



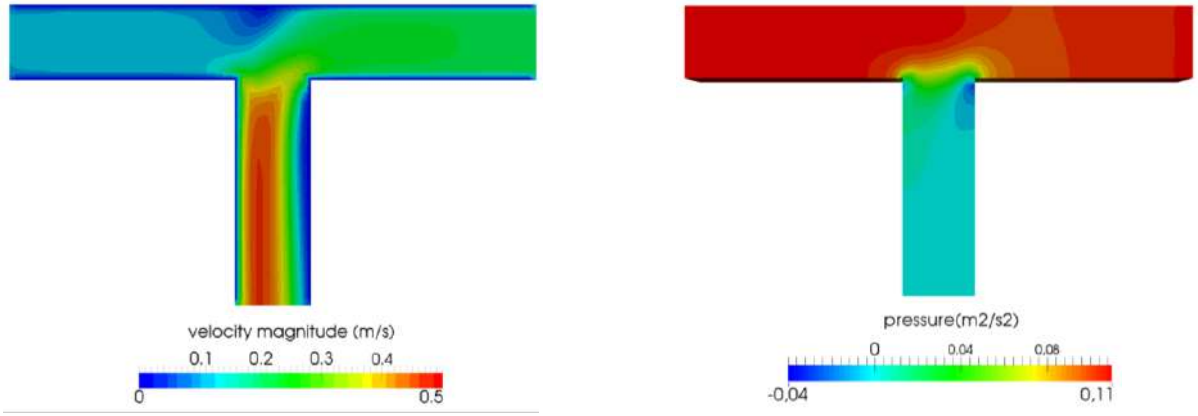
Mesh created using blockMesh

```
>simpleFoam
```

Wait for simulation to converge. After convergency check the results to be sure the solution is converged.

```
>foamToVTK
```

The simulation results are as follows (results are on the cut plane in the middle):



Simulation results after convergence (65 iterations)

3. RTD calculation

3.1. Copy tutorial

Copy following tutorial to your working directory:

```
$FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily
```

3.2. 0 directory

Delete the U file and replace it with the calculated velocity field from the first part of the tutorial (use the latest time step velocity field from previous part of simulation to calculate RTD for this geometry). There is no need to modify or change it. The solver will use this field to calculate the scalar transportation.

Update T (T will be used as an inert scalar in this simulation) file boundary conditions to match new simulation boundaries, to calculate RTD of the `inlet_one` set the `internalField` value to `0`, T value for `inlet_one` to `1.0` and T value for `inlet_two` to `0`.

3.3. system directory

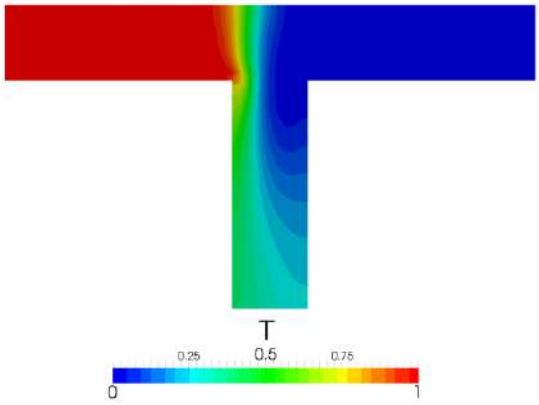
Replace the blockMeshDict file with the one from the first part of tutorial.

In the controlDict file change the `endTime` from `0.1` to `120` (approximately two times ideal resistance time) and also `deltaT` from `0.0001` to `0.1` (Courant number approximately 0.4).

4. Running Simulation

```
>blockMesh
>scalarTransportFoam
>foamToVTK
```

5. Post-processing



Contour plots scalar T at 120 s for inlet 1

5.1. Calculating RTD

To calculate RTD the average T value at the outlets should be calculated first. The "integrate variables function" of ParaView can be used for this purpose.

```
>foamToVTK
```

Load the outlet VTK file into paraview using following path:

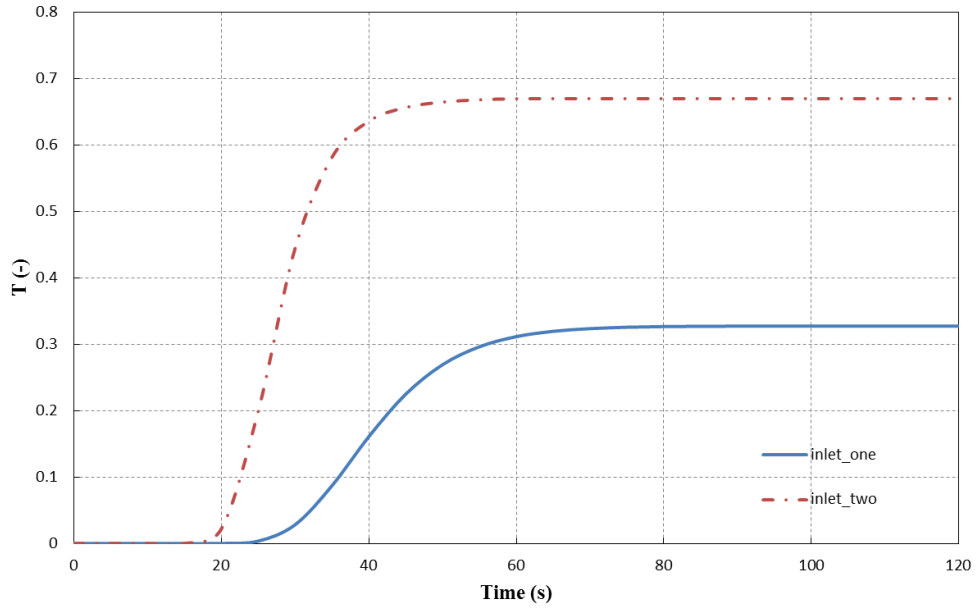
File > Open > VTK > outlet > outlet_..vtk > OK > Apply

Select T from variables menu, and then integrate the variables on the outlet:

Filters > Data Analysis > Integrate Variables > Apply

The values given in the opened window are integrated values in this specific time step. By changing the time step values for different time steps are displayed. As mentioned before, the average value of the property is needed. Therefore, these values should be divided by outlet area to get average values (1m × 1m).

The same procedure should be followed for calculating RTD of `inlet_two`, except T value for `inlet_one` should be `0` and for `inlet_two` it should be `1.0`.

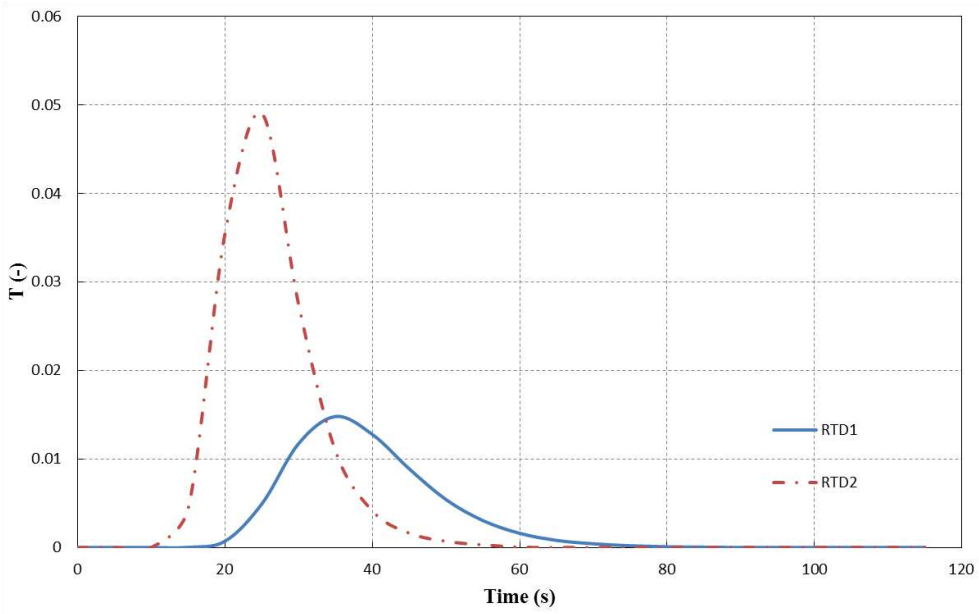


Average value of T on the outlet for two inlets versus time

The average value of T for each outlet approaches a certain constant value, which is the ratio of that scalar mass inlet to the whole mass inlet. For plotting data over time "Plot Selection Over Time" option in ParaView can be used, in the opened SpreadSheetView window (IntegrateVariables) select the set of data which you want to plot over time and then:

Filters > Data Analysis > Plot Selection Over Time > Apply

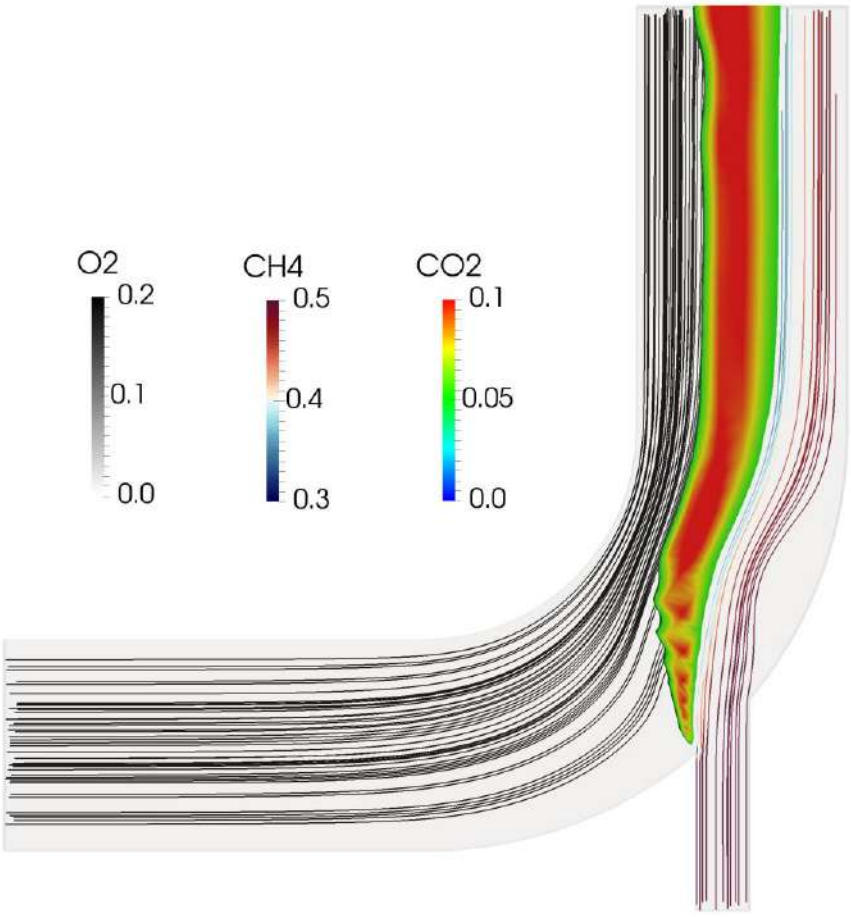
Next, to obtain the RTD plots, simply calculate the gradient of change in average value of T on the outlet from time 0 to 120s, export the data to Excel, and plot the results.



RTD of two inlets

Tutorial Eleven

Reaction



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:
- Bahram Haddadi
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

There are two common approaches in modeling reactions:

1. Partially stirred reactor (PaSR) Model

Partially stirred rector (PaSR) model is used to model thermodynamic and chemical reactions numerically, for example, combustion. In the PaSR approach, a computational cell is split into two different zones: a reacting zone and a non-reacting zone. The reacting zone is modeled as a perfectly stirred reactor (PSR), and all reactants are assumed to be perfectly mixed with each other.

For the reactor, we are interested in three concentrations, 1) mean concentration of key component in the feed, c_{in}; 2) mixture concentration in the reacting zone, c; 3) concentration at the reactor exitc_{exit}.

In the reacting zone, reaction occurs for a duration of τ_c, so the concentration of mixture changes from c_{in} to c. In the non-reacting zone, the reacted mixture is getting mixed up with the non-reacted mixture for a duration of τ_{mix}, resulting in the final exit concentration, c_{exit}.

A key parameter to be calculated in this model would be the reaction rate, and it is clear that the reaction rate is proportional to the ratio of the chemical reaction time to the total conversion time in the reactor (i.e. sum of reacting and mixing time), κ_k:

$$\kappa_k = \frac{\tau_c}{\tau_c + \tau_{mix}}$$

2. Eddy dissipation concept (EDC) Model

The Eddy Dissipation Concept (EDC) model looks at the interaction between reaction and turbulence, where the overall reaction rate is controlled by turbulent mixing. It is widely used for combustion modeling for a great variety of combustion environments with great success.

It is assumed in the model that most reaction takes place within fine turbulence structures, which are modeled as perfectly-mixed reactors. We need to know the reaction mass fraction and the mass transfer rate between the fine structures and its surrounding fluid.

The mass fraction occupied by the fine structures, γ^*, is expressed as:

$$\gamma^* = \left\{\frac{u^*}{u'}\right\}^2$$

Where u^* is the mass average fine structure velocity. The fine structures are located in regions with nearly constant turbulent kinetic energy given by u'^2.

The mass transfer rate between fine structure and surrounding fluid per unit of fluid and per unit of time is modeled as:

$$\dot{m} = 2 \cdot \frac{u^*}{L^*} \cdot \gamma^*$$

where L^* is the characteristic length of the fine structure.

reactingFoam – reactingElbow

Simulation

Use the reactingFoam solver, simulate combustion of CH_4 and O_2 in a mixing elbow:

- Use the two times finer Hex mesh from Example One
- Domain initially filled with N_2
- velocity-inlet-5:
  - Velocity: 1 m/s
  - Mass fractions: 23 % O_2, 77 % N_2
  - Temperature: 800 K
- velocity-inlet-6:
  - Velocity: 3 m/s
  - Mass fractions: 50 % CH_4, 50 % N_2
  - Temperature: 293 K
- Operating pressure: 10^5 Pa
- Operating temperature: 298 K
- Isolated walls

Objective

- Understanding multi-species and reaction modeling in OpenFOAM®

Data processing

Evaluate your results in ParaView.

1. Pre-processing

1.1. Copy tutorial

Copy the following tutorial to your working directory:

$FOAM_TUTORIALS/combustion/reactingFoam/laminar/counterFlowFlame2D

Copy the GAMBIT® mesh from Tutorial One (two times finer mesh) to the case main directory.

1.2. 0 directory

Update all the files in 0 directory with new boundary conditions, e.g. U:

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

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

internalField   uniform (0 0 0);

boundaryField
{
    wall-4
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    velocity-inlet-5
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    velocity-inlet-6
    {
        type            fixedValue;
        value           uniform (0 3 0);
    }

    pressure-outlet-7
    {
        type            zeroGradient;
    }

    wall-8
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

The reaction taking place in this simulation CH_4 combusting with O_2 creating CO_2 and H_2O. N_2 is the non-reacting species. The boundary condition and initial value of all species should be defined in the 0 directory. These values are mass fractions (between 0 and 1) and dimension less, e.g. CH_4:

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

dimensions      [0 0 0 0 0 0 0];
```

```
internalField   uniform 0.0;

boundaryField
{
    wall-4
    {
        type            zeroGradient;
    }

    velocity-inlet-5
    {
        type            fixedValue;
        value           uniform 0; //no CH4 at this inlet
    }

    velocity-inlet-6
    {
        type            fixedValue;
        value           uniform 0.5; //50% CH4 mass fraction at this inlet
    }

    pressure-outlet-7
    {
        type            zeroGradient;
    }

    wall-8
    {
        type            zeroGradient;
    }

    frontAndBackPlanes
    {
        type            empty;
    }
}

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

Note: If the file for a species does not exist in the 0 directory, the values from Ydefault will be used for that species.

1.3. constant directory

In the thermophysicalProperties file the physical properties of the species can be set:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}

inertSpecie N2;

chemistryReader foamChemistryReader;

foamChemistryFile "$FOAM_CASE/constant/reactions";

foamChemistryThermoFile "$FOAM_CASE/constant/thermo.compressibleGas";
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

The mixture type is set to a reacting mixture for calculating the mixture properties and the heat capacities are calculated using "janaf polynomials".

N_2 is defines as `inertSpecie`. In reaction solvers in OpenFOAM® the inert specie is calculated explicitly using the mass balance equation (to satisfy mass conservation):

$$mass\ fraction\ of\ inert\ specie = 1 - \sum mass\ fraction\ of all\ other\ species$$

The species and the reactions are addressed using `foamChemistryFile`. In this simulation reactions and species are read from reactions file in the constant directory:

```
species
(
    O2
    H2O
    CH4
    CO2
    N2
);

reactions
{
    methaneReaction
    {
        type     irreversibleArrheniusReaction;
        reaction "CH4 + 2O2 = CO2 + 2H2O";
        A        5.2e16;
        beta     0;
        Ta       14906;
    }
}
```

OpenFOAM® v1712: Also the elements are listed in the reactions file and an element balance is performed in the calculations!

The species in this simulation are O_2, H_2O, CH_4, CO_2 and N_2. They are defined in the `species` sub-dictionary. In the `reactions` sub-dictionary, reactions are specified. The reaction of methane combustion is defined and it is of `type` irreversible Arrhenius reaction, `irreversibleArrheniusReaction`.

In the Tutorial Two it was explained that the coefficients for calculating gas mixture properties are defined in the `mixture` sub-dictionary because it was a homogeneous mixture. But in this example the mixture is not homogenous so coefficients for calculating properties of each species are needed separately to calculate mixture properties based on each cell composition. The coefficients of each species are defined in the foamChemistryThermoFile, which reads the file *thermos.compressibleGas* from the constant directory (this step is outlined in the thermophysicalProperties file). For example, the `O2` coefficients for each model are shown below:

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

O2
{
    specie
    {
        molWeight       31.9988;
    }
    thermodynamics
    {
```

```
        Tlow            200;
        Thigh           5000;
        Tcommon         1000;
        highCpCoeffs    ( 3.69758 0.00061352 -1.25884e-07 1.77528e-11 -
                          1.13644e-15 -1233.93 3.18917 );
        lowCpCoeffs     ( 3.21294 0.00112749 -5.75615e-07 1.31388e-09 -
                          8.76855e-13 -1005.25 6.03474 );
    }
    transport
    {
        As              1.67212e-06;
        Ts              170.672;
    }
}
…

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

OpenFOAM® v1712: Number of elements in the specie is listed in this file!

In the `thermodynamics` sub-dictionary the janaf polynomial model coefficients for calculating the heat capacity can be found and in `transport` the sutherland model coefficients for viscosity are stored.

1.4. system directory

By setting the `adjustTimeStep` to `yes` in the controlDict, the solver automatically ignores `deltaT`, and calculates the `deltaT` based on the maximum Courant number `maxCo` defined for it. Change the `endTime` to `120` (approximately one time the volumetric residence time based on velocity-inlet-5) and `writeTimeInterval` to `10`, to write every 10 s to case directory.

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

application     reactingFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         120;

deltaT          1e-6;

writeControl    adjustableRunTime;

writeInterval   10;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           0.4;
```

// *//

2. Running simulation

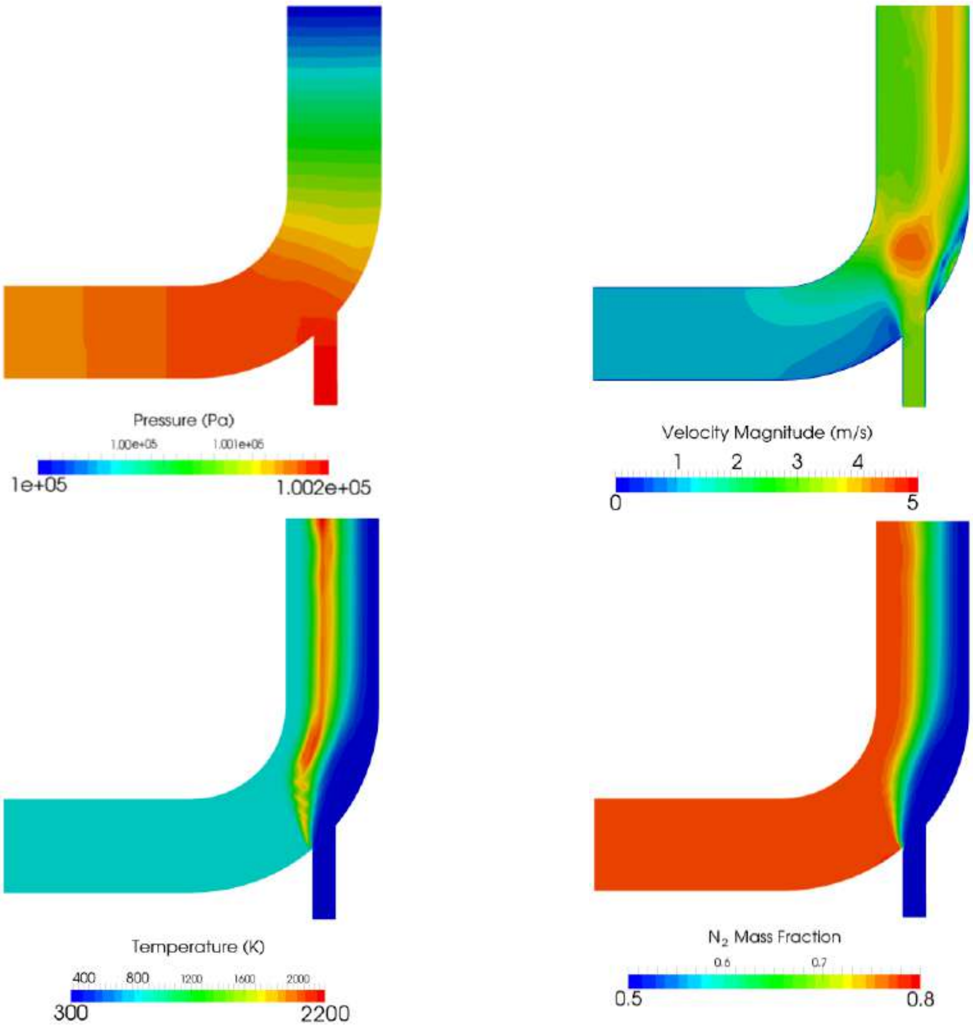
```
>fluentMeshToFoam fineHex.msh
```

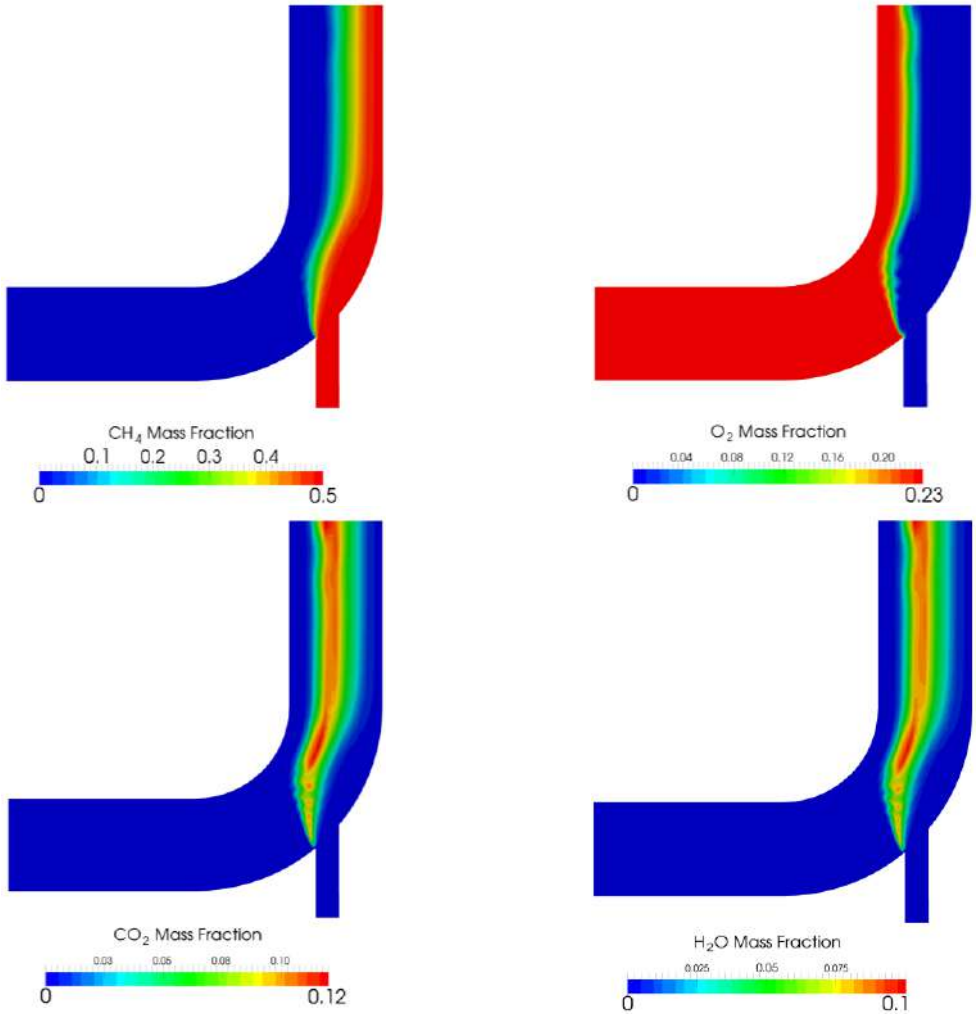
After converting the mesh, check the boundary file in the constant/polyMesh directory and change the `type` and `inGroups` of boundary `frontAndBackPlanes` from `wall` to `empty` (it is a 2D simulation).

```
>reactingFoam
>foamToVTK
```

3. Post-processing

The simulation results are as follows:

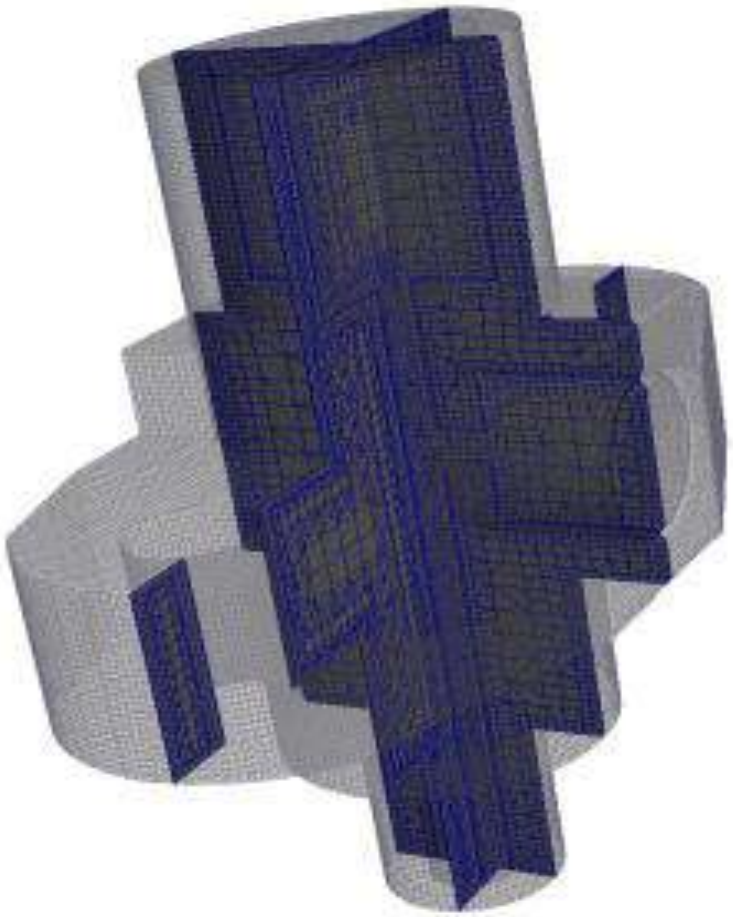




Simulation results after 120 s

Tutorial Twelve

snappyHexMesh – Single Region



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:
- Philipp Schretter
- Yitong Chen

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Philipp Schretter

For more tutorials visit: www.cfd.at

Background

In this tutorial, we will familiarize ourselves with the *snappyHexMesh* tool in OpenFOAM®. This utility generates 3D meshes containing hexahedra and split-hexahedra. We will also introduce different types of meshes with complex geometries and compare the *snappyHexMesh* tool with other mesh generation tools.

1. Meshes with complex geometries

So far we have only worked with meshes in Cartesian co-ordinates, however, many engineering problems involve complex geometries that do not fit exactly in Cartesian co-ordinates. In such cases, it would be much more advantageous to work with grids that can handle curvature and geometric complexity more naturally.

CFD methods for complex geometries are classified into two groups:

1) structured curvilinear grid arrangements

2) unstructured grid arrangements

In a **structured grid** arrangements:

- Cells center points are placed at the intersections of co-ordinates lines
- Cells have a fixed number of neighboring cells
- Cells center points can be mapped into a matrix based on their location in the grid
- Structure and position in the matrix is given by indices (*I*, *J* in two dimensions and *I*, *J*, *K* in three dimensions)

For the most complex geometries it may be necessary to sub-divide the flow domain into several different blocks, where each mesh cell is a block, this is known as **block-structured grids**. The next level of complexity is the **unstructured grids**. It gives unlimited geometric flexibility, here the limitations of structured grids do not apply – but at the cost of higher programming and computational efforts. Unstructured grids also allow the most efficient use of computing resources for complex flows, so this technique is now widely used in industrial CFD.

2. Mesh generation tools

There are a number of advanced meshing tools available, both commercial and free source. The major mesh generators are ANSYS GAMBIT®, ICEM, Salome, snappyHexMesh and cfMesh. Here we will learn about GAMBIT®, snappyHexMesh and cfMesh tools in detail.

2.1. GAMBIT®

GAMBIT® is a 3D unstructured tool, to specify the meshing scheme in it, two parameters must be specified:

- **Elements**
- **Type**

The Elements parameter defines the shape(s) of the elements that are used to mesh the object. The Type parameter defines the pattern of mesh elements on the object. It has a single graphical user interface which brings geometry creation and meshing together in one environment.

2.2. snappyHexMesh

In contrast to GAMBIT®, which incorporates both mesh generation and refinement, the *snappyHexMesh* tool built within OpenFOAM® requires an existing geometry base mesh to work with. The base mesh usually comes from the blockMesh tool. This utility has the following key features:

- allow parallel execution to speed up the process
- supports geometry data from STL/OBJ files
- addition of internal and wall layers
- zonal meshing

The key steps involved when running snappyHexMesh are:

- **Castellation**: The cells which are beyond a region set by a predefined point are deleted
- **Snapping**: Reconstructs the cells to move the edges from inside the region to the required boundary
- **Layering**: Creates additional layers in the boundary region.

The advantages of snappyHexMesh over the other mesh generation tools are as follows:

- No commercial software package is ultimately necessary. For the meshing, the OpenFOAM® environment is sufficient and no further software is necessary.
- The geometry can be created with any CAD program like CATIA®, FreeCAD, etc. As the geometry is to be only surface data, the files need to be in .stl, .nas or .obj. format.
- The meshing process can be run in parallel mode. If high computational capabilities are available, high quality meshes can be generated in little time.

2.3. cfMesh

cfMesh is an open-source library for mesh generation implemented within the OpenFOAM® framework (similar to *snappyHexMesh*). Currently cfMesh is capable of producing mesh of Cartesian type in both 2D and 3D, tetrahedral and polyhedral types.

The fundamental work-flow of the tool starts from a mesh template, then followed by a mesh modifier. The modifier allows for efficient parallelization using shared memory parallelization (SMP) and distributed memory parallelization using MPI.

snappyHexMesh – flange

Simulation

The procedure described in this tutorial is structured in the following order:

- Creation of the geometry data
- Meshing a geometry with one single region
- Run an OpenFOAM® simulation with the generated mesh using scalarTransportFoam

Objectives

- The aim of the tutorial is to give a basic introduction to single region meshing with the meshing tool *snappyHexMesh*
- Understanding the advantages of *snappyHexMesh*
- Understanding the three basic steps of *snappyHexMesh*

Data processing

Import your simulation to ParaView. Analyze the heat distribution in the flange.

1. Pre-processing

1.1. Copy tutorial

Copy the following tutorial to your working directory.

```
$FOAM_TUTORIALS/mesh/snappyHexMesh/flange
```

1.2. Set-up of stl files

Normally the .stl files are created using CAD software, such as CATIA® and freeCAD. stl files contain information about the solid geometry. However, in this tutorial the stl files are available to be copied from the OpenFOAM® tutorials folder. To do this, copy the stl files from the below location to the constant/triSurface of your running case directory.

```
$FOAM_TUTORIALS/resources/geometry/flange.stl.gz
```

1.3. constant directory

The *constant* directory must initially have the following folder:

- **triSurface***:*
  The folder **triSurface** should contain a file with the geometry data to be meshed (stl, nas, obj). The file name is to be used as a reference pointer in later stages.

Note: The stl file should be in ascii format. All the stl files (different boundaries stl files) should form a closed geometry together.

1.4. system directory

For creating a mesh using snappyHexMesh the following files should be present in system directory:

- **blockMeshDict**

  For meshing using snappyHexMesh a background mesh is needed, which should surround the geometry surface (e.g. stl file) file. The background mesh will be refined based on the settings in the snappyHexMeshDict and the extra parts will be removed. Usually the background mesh is crated using blockMesh. Here we define a base mesh.
  Note: To ensure that the sharp edges are refined properly, it is very important to create perfect cube cells in the background mesh using blockMesh utility.

- **decomposeParDict**

  The meshing using snappyHexMesh can be also performed in parallel mode. If the mesh is to be run in parallel using the *decomposePar* utility, this file defines the parameters for distributed processors

- **meshQualityDict**

  Parameters to be checked for mesh quality and their values are defined in this file.

- **surfaceFeatureExtractDict:**

Using surfaceFeatureExtract utility prior to meshing with snappyHexMesh helps to extract the sharp edges and have a better mesh with snappyHexMesh on these edges. All edges are marked, whose adjacent surfaces normal are at an angle less than the angle specified in `includedAngle`. For the included angels, a special level of refinement can be set in the `features` sub-dictionary in the *snappyHexMeshDict*.

```
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    flange.stl
    {
        extractionMethod    extractFromSurface;

        extractFromSurfaceCoeffs
        {
            includedAngle   150;
        }
            writeObj                yes;
    }
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

- **snappyHexMeshDict:**

This file includes the settings for running the snappyHexMesh. As mentioned in the Background section meshing using this tool has three steps:
1 – Castellating
2 – Snapping
3 – Layering
In the first section of this file you can set `castellatedMesh`, `snap`, `addLayers` to true or false depending on the stages required.

```
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
        castellatedMesh          true;
        snap                     true;
        addLayers                true;
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

The `Geometry` sub-dictionary lists all surfaces used by *snappyHexMeshDict*, except the blockMesh geometry, and defines a name for each of them to be used as a reference.

Then we specify a region of the domain that we want to refine. The refined region is given an arbitrary name; in this case, it is `refineHole` which is a sphere with its center and radius defined.

```
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    geometry
    {
        flange
        {
            type triSurfaceMesh;
            file "flange.stl";
        }
        refineHole
        {
            type searchableSphere;
            centre (0 0 -0.012);
            radius 0.003;
        }
    };
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

OpenFOAM® v1712: The geometry section of the snappyHexMeshDict looks as following:

```
//  * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
      geometry
      {
          flange.stl
          {
              type triSurfaceMesh;
              name flange;
          }
          …
      };
  //  * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

CASTELLATING

The `castellatedMeshControls` sub-dictionary is used for user-defined edge refinement. It uses the eMesh files, created with the surfaceFeatureExtract command. These eMesh files are extracted parts from the stl surfaces, specified in the *surfaceFeatureExtractDict*.

`features` allows explicit feature edge refinement and a level is chosen for any cell intersected by its edges.

`refinementSurfaces` are for surface based refinement. Every surface is specified with two levels. The first level is the minimum level that every cell intersecting the surface gets refined up to. The second level is the maximum level of refinement.

`resolveFeatureAngle` is an important setting. Edges, whose adjacent surfaces normal are at an angle higher than the value set, are resolved. The lower the value, the better the resolution at sharp edges.

`refinementRegions`: The refinement level of the previous specified `refinementHole` is defined. The cells in this region are split up to the number of cells defined, or until the max number of cells is reached.

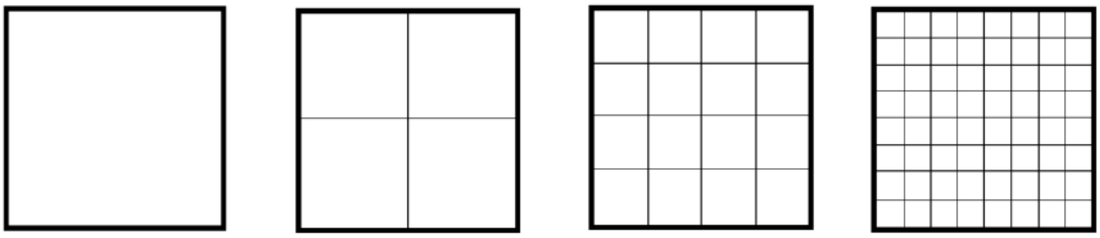
`locationInMesh`: Important coordinate for single region cases, to define a region which is to be kept from the blockMesh.

```
//  * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    castellatedMeshControls
    {
        maxLocalCells 100000;
        maxGlobalCells 2000000;
        minRefinementCells 0;
        nCellsBetweenLevels 1;

        features
        (
            {
                file "flange.extendedFeatureEdgeMesh";
                level 0;
            }
        );

        refinementSurfaces
        {
```

```
            flange
            {
            level (2 2);
            }
        }

        resolveFeatureAngle 30;

        refinementRegions
        {
            refineHole
            {
               mode inside;
               levels ((1E15 3));
        }
        locationInMesh (-9.23149e-05 -0.0025 -0.0025);
        allowFreeStandingZoneFaces true;
    }
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

In the first step, the `castellatedMeshControls`, the initial mesh can be refined with levels. The level depends on the refinement set in the blockMesh, and the required refinement on the surfaces. Therefore, levels can be set in the sub-directories `features`, `refinementSurfaces` and `refinementRegions` in the *snappyHexMeshDict*.

Level 0 stands for no refinement and each subsequent level splits the cell into 4 separate cells.



Refinement level 0, level 1, level 2, level 3

SNAPPING

Important parameters are number of mesh displacement iterations, `nSolveIter` and the number of feature edge snapping iterations, `nFeatureSnapIter`.

```
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    snapControls
    {
        nSmoothPatch 3;
        tolerance 1.0;
        nSolveIter 300;
        nRelaxIter 5;
            nFeatureSnapIter 10;
            implicitFeatureSnap false;
            explicitFeatureSnap true;
            multiRegionFeatureSnap true;
    }
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

LAYERING

The label for the layering is equal to the labeling of the Boundary surface in the boundary file in the constant/polyMesh folder.

- `nSurfaceLayers` defines the number of surface layers

- `expansionRatio` defines the expansion ratio of the surface layers
- `finalLayerThickness` and `minThickness` define the min and the final thickness of the surface layers
- `nLayerIter`: if not snapped smoothly enough, the max number of layer addition iteration can be increased.

```
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    addLayersControls
    {
        relativeSizes true;
        layers
        {
            "flange_.*"
            {
                nSurfaceLayers  3;
            }
        }
        expansionRatio 1.005;

        finalLayerThickness 0.3;
        minThickness 0.25;
        nGrow 0;
        featureAngle 30;
        nRelaxIter 5;
        nSmoothSurfaceNormals 1;
        nSmoothNormals 3;
        nSmoothThickness 10;
        maxFaceThicknessRatio 0.5;
        maxThicknessToMedialRatio 0.3;
        minMedianAxisAngle 90;
        nBufferCellsNoExtrude 0;
        nLayerIter 50;
        nRelaxedIter 20;
    }
    meshQualityControls
    {
        #include "meshQualityDict"
        relaxed
        {
            maxNonOrtho 75;
        }

        nSmoothScale 4;
        errorReduction 0.75;
    }
    writeFlags
    (
        scalarLevels
        layerSets
        layerFields
    );
    mergeTolerance 1e-6;
// * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Note: Only the relevant changes, which were used in the sample flange case, are commented in the snappyHexMeshDict.

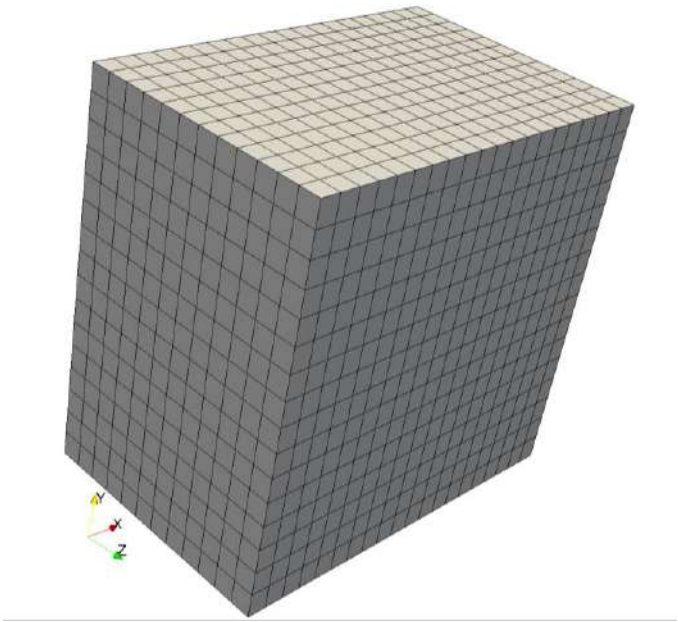
2. Running snappyHexMesh

The background mesh is created with the following command:

```
>blockMesh
```

According to the settings in the *blockMeshDict*, the mesh was created with 20 cells in x direction,

16 cells in y direction and with 12 cells in z direction.



Block mesh for flange

The command `surfaceFeatureExtract` creates the **eMesh** files from the stl files and creates the folder *extendedFeatureEdgeMesh* in the constant directory. Refining certain parts of the surface geometry with the `surfaceFeatureExtract` tool is optional.

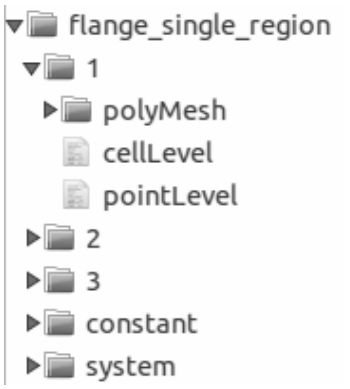
```
>surfaceFeatureExtract
```

The command to mesh the flange geometry on one processor is

```
>snappyHexMesh
```

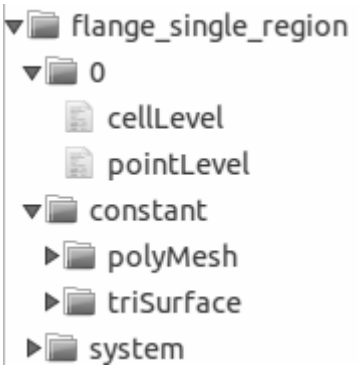
Note: The meshing process with snappyHexMesh can also be run in on several processors in parallel. To run the command on several processors, refer to Tutorial Eight for more information.

The command `snappyHexMesh` creates a folder with the mesh files for each mesh step. If, for example, in the **snappyHexMeshDict**, only *castellatedMesh* is set to *true* and *snap* and *addLayers* are set to *false*, only one folder is created. If also *snap* is set to *true*, 2 folders are created and if also *addLayers* is set to *true*, 3 folders with 3 polyMesh folders are created.



Folders structure after running snappyHexMesh

In order to avoid the creation of these folders and only keep the final mesh, the following command can be used to overwrite the previous meshing steps. In this case, only one polyMesh folder exits in the /constant directory.



Folders structure after using -overwrite flag

```
>snappyHexMesh -overwrite
```

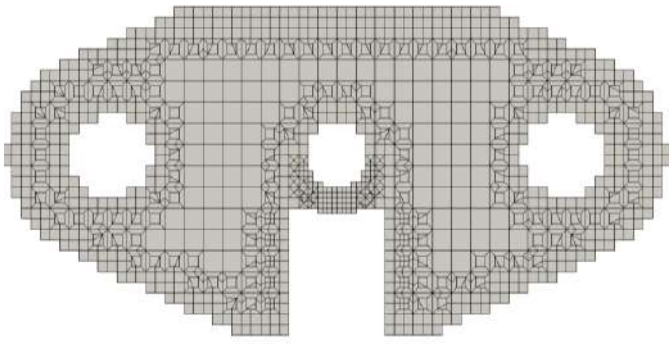
However, sometimes it is useful to run snappyHexMesh without the overwrite option, as it allows the user to make changes to a specific time step without having to run all the other steps again, thus reducing computational time.

3. Examining the meshes

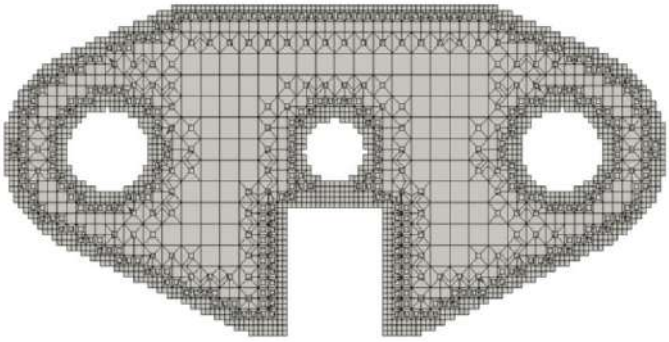
To examine, what each of the steps in the *snappyHexMeshDict* really does, we need to turn off the overwrite feature in snappyHexMesh command and generate VTK files to be opened in ParaView.

```
>foamToVTK
```

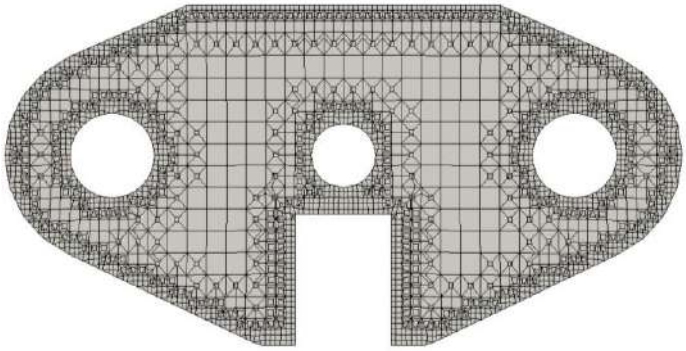
Simply change the time in Paraview to see the effect of *snappyHexMesh* steps on the mesh, i.e. time 1 corresponds to the mesh after castellating step, time 2 for the mesh after snapping step, time 3 for the mesh after the layering step.



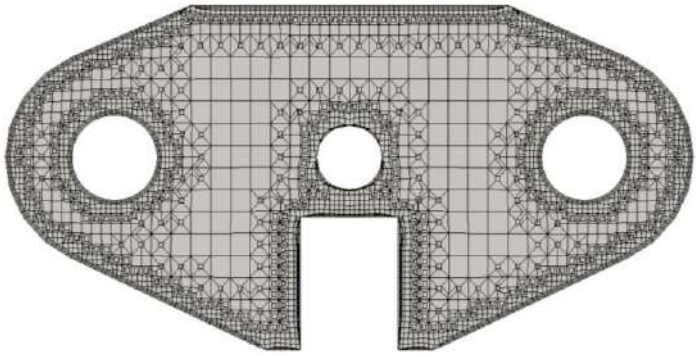
Flange mesh for step castellate with surface refinement level 2



Flange mesh for step castellate with surface refinement level 3

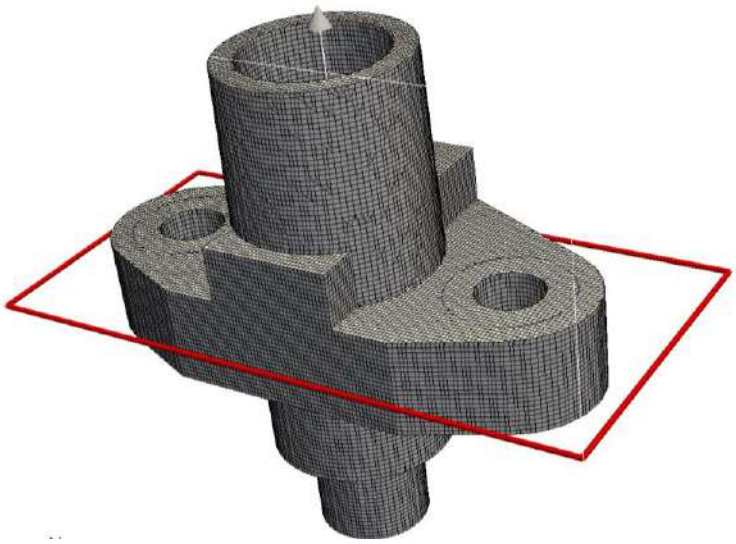


Flange mesh for step snap with surface refinement level 3



Flange mesh for step addlayers with surface refinement level 3

The slice views taken with ParaView from the center of the flange. The slices are depicted by the red plain in the following figure:



Flange with sectional plain

You can review the mesh quality with the tool *checkMesh*.

```
>checkMesh
```

4. Running simulation

4.1. Copy tutorial

Now with the new mesh ready, let's run some simulation on it! Here scalarTransportFoam solver is chosen for the simulation. To set up the case, copy the following tutorial file into your working directory:

```
$FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily
```

The flange mesh files need to be transferred to the running case directory. To achieve this, copy the polyMesh folder from the latest time step file of the flange folder into the constant directory of the pitzDaily folder. If the overwrite function is activated when running snappyHexMesh, copy the polyMesh folder from constant directory of the flange folder.

4.2. Case set-up

The following changes need to be made to set up the case:

- Update the T file in the 0 directory, so that the flange has an initial temperature of 293K but is heated up from the inlet at 350K

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

internalField   uniform 293;

boundaryField
{
   flange_patch1
    {
        type            fixedValue;
        value           uniform 350;
    }
    flange_patch2
    {
        type            fixedValue;
```

```
        value           uniform 293;
    }
    flange_patch3
    {
        type            fixedValue;
        value           uniform 293;
    }
    flange_patch4
    {
        type            fixedValue;
        value           uniform 350;
    }
}
```

- Update the U file in the 0 directory so that the velocity in the entire flange domain and at the boundaries is zero

- Update the *controlDict* file in the system directory by changing the `endTime` to 0.0005, `deltaT` to 0.000001 and `writeInterval` to 100.

4.3. Running solver

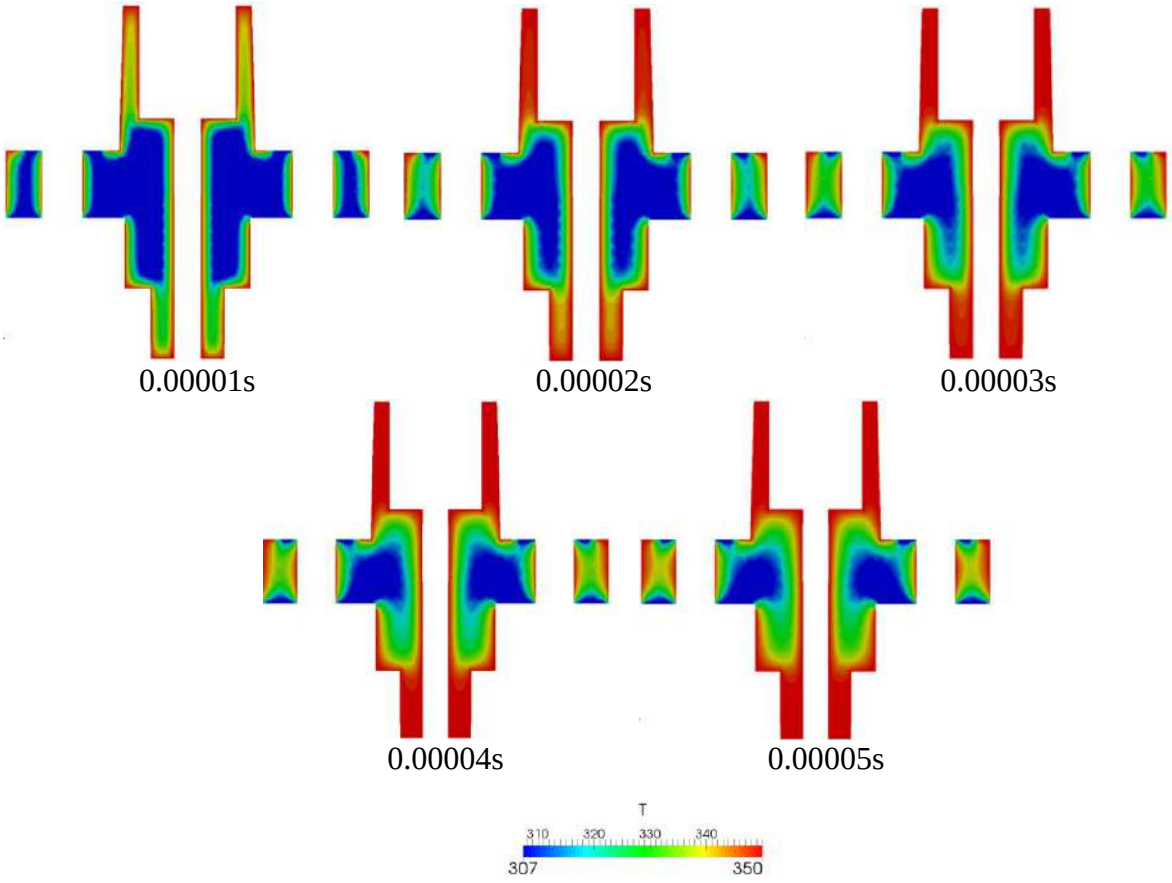
Run the solver with the command

```
>scalarTransportFoam
```

4.4. Results

Convert the results to VTK files with

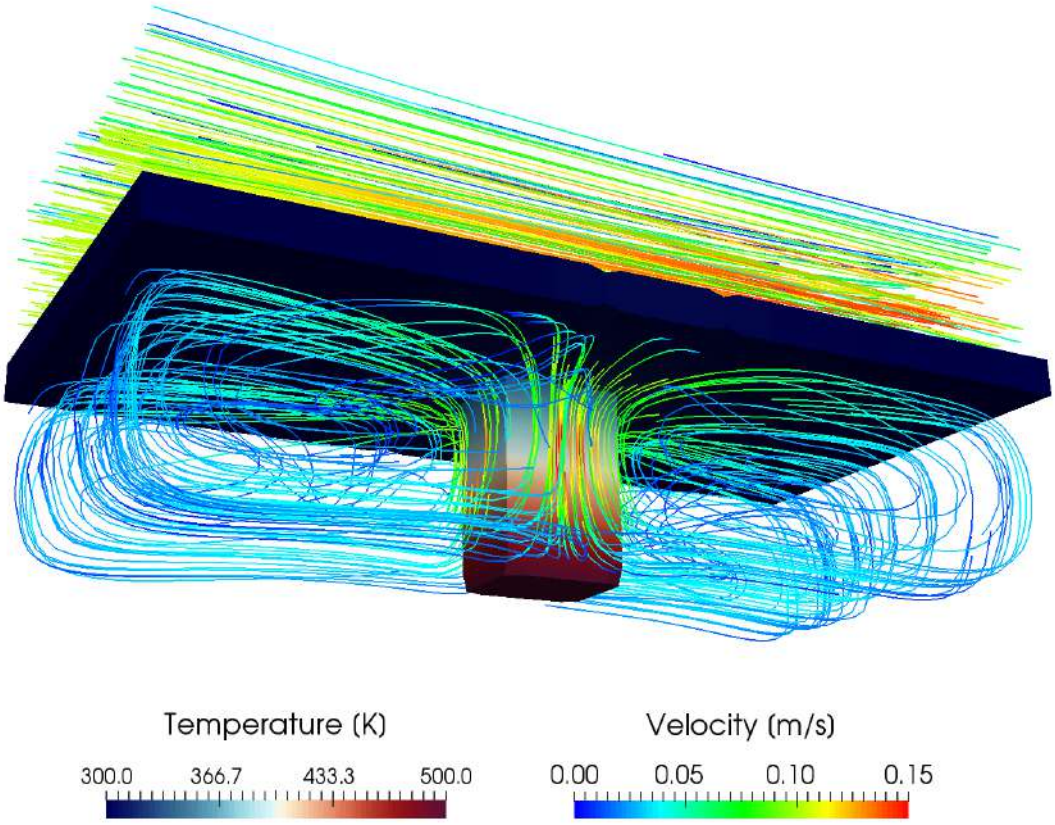
```
>foamToVTK
```



Heating of the flange from 0.01 to 0.05s

Tutorial Thirteen

snappyHexMesh – Multi-Region



4th edition, Jan. 2018

Editorial board:
- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:
- Bahram Haddadi
- Philipp Schretter
- Yitong Chen

Compatibility:
- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:
- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Multi-region modeling & why do we need it?

In multi-region modeling, the entire computational domain is divided into individual regions, with each region representing a coherent continuum of the same phase. The key feature of this type of modeling is that separate governing transport equations are solved for each region.

In general, two different approaches have been adopted in the past to solve multi-region problems:

- Monolithic: solve using a single coupled matrix equation system
- Partitioned: solve using separate matrix equation systems

In this OpenFOAM® tutorial, we are focusing on the partitioned approach. The fundamental steps in this approach are outlined below:

1. Define the whole mesh domain and the separate regions within it. Assign cells into each region
2. Specify field variables in each region
3. Solve the transport equation in each individual region
4. Multiregional coupling at the interface between different regions
5. Iteration to achieve fully/coupled solution

2. chtMultiRegionFoam solver

This solver is developed to solve heat transfer problems between multiple regions.

snappyHexMesh - snappyMultiRegionHeater

Simulation

The procedure described in this tutorial is structured in the following order:

- Creation of the geometry data
- Tutorial on Meshing a geometry with more than one region
- Run an OpenFOAM® simulation with the generated mesh using chtMultiRegionFOAM

Objectives

- Understanding multi region meshing with the meshing tool *snappyHexMesh*

Data processing

Import your simulation to ParaView. Analyze the flow field through the flange and the heat distribution in the flange.

1. Pre-processing

1.1. Copy tutorial

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

`$FOAM_TUTORIALS/mesh/snappyHexMesh/snappyMultiRegionHeater`

1.2. 0 directory

Unlike the single region simulations in the 0 directory an individual folder per region exist which stores the files including initial and boundary conditions for that region. Also in the 0 directory some files exists which are just dummy files that will not be used in the simulations. The initial and boundary conditions for each region are changed and updated using the *changeDictionary* utility which will be explained later.

1.3. constant directory

Also in the constant directory exist a folder per region; in this case, the domain is split into the following regions: bottom air, heater, left solid, right solid and top air. Within the designated folder, there are relevant dictionaries that describe the physical properties, turbulence or radiation behavior of each region, e.g. radiationProperties, turbulenceProperties and thermophysicalProperties.
The polyMesh directory in the constant folder includes the original mesh while the polyMesh directories in each region folder include the split mesh for that region with the new boundaries between regions.
Unlike polyMesh directories there exist just one triSurface folder which stores all the stl files for mesh creation using snappyHexMesh.
In the regionProperties file, the physical phase of each region is specified. As you can see, bottom and top air regions are fluid, whereas heater, left and right solid are in solid phase.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
    regions
    (
            fluid                   (bottomAir   topAir)
            solid                   (heater   leftSolid   rightSolid)
    );
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

1.4. system directory

Like constant directory also in system directory a folder per region can be found and all the settings for that region are stored in the corresponding folder, e.g. fvSolution, fvSchemes and decomposeParDict. The fvSchemes file in the system directory is a dummy file while the fvSolution includes the number of outer correctors setting for PIMPLE algorithm. There is also just one controlDict file and it is in main system folder.

Note: For running the simulations in parallel the decomposeParDict files for all the regions should have the same settings as the main one in the system directory.This is not valid for parallel meshing using snappyHexMesh while it just uses the decomposeParDict file in the main system directory.

The files needed for creating a multi-region mesh are the same as the mesh for single-region, except for slight differences in snappyHexMeshDict file:

`locationInMesh`: In a multi-region mesh this point is not used but it should be defined just as a place holder.

`refinementSurfaces`: Different regions are defined in here. E.g. for the region BottomAir all the faces and cells inside the bottomAir stl (each region stl should be a closed volume) file are marked with `bottomAir` flag (in `faceZone` and `cellZone`).

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
castellatedMeshControls
{
    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 10;
    nCellsBetweenLevels 2;

    features
    (
        {
            file "bottomAir.eMesh";
            level 1;
        }
    …
        {
            file "topAir.eMesh";
            level 1;
        }
    );

    refinementSurfaces
    {
        bottomAir
        {
                level (1 1);
                faceZone bottomAir;
                cellZone bottomAir;
                cellZoneInside inside;
            }
    …
          rightSolid
            {
                level (1 1);
                faceZone rightSolid;
                cellZone rightSolid;
                cellZoneInside inside;
            }
        }

        resolveFeatureAngle 30;

        refinementRegions
        {
        }
        locationInMesh (0.01 0.01 0.01);
        allowFreeStandingZoneFaces false;
    }
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

After creation of the mesh and splitting to different regions the initial and boundary conditions for each region can be manually set in the relevant region folders in 0 directory. This process can be also automated usind the changeDictionary utility. The dictionary file for this utility for each region is in the relevant region folder in the system directory: changeDictionaryDict.

See below the changeDictionaryDict file for the heater region. In the `boundary` sub-dictionary type of boundaries for `minY`, `MinZ` and `maxZ` are set to patch. Then for `T` the internal field will be overwritten with `uniform 300`. In the next step all the boundaries in the T file for heater

region will be set to `zeroGradient` (".*" means all the boundaries with any name) and after that the bounadries with the name "`heater_to_.*`" will be changed to `turbulentTemperatureCoupledBaffleMixed` and `minY` will be changed to `fixedValue`.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    boundary
    {
        minY
        {
            type            patch;
        }
        minZ
        {
            type            patch;
        }
        maxZ
        {
            type            patch;
        }
    }

    T
    {
        internalField   uniform 300;

        boundaryField
        {
            ".*"
            {
               type               zeroGradient;
               value              uniform 300;
            }
            "heater_to_.*"
            {
               type               compressible::turbulentTemperatureCoupledBaffleMixed;
               Tnbr               T;
               knappaMethod       solidThermo;
               value              uniform 300;
            }
            minY
            {
               type               fixedValue;
               value              uniform 500;
            }
    }
 }
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

2. Mesh creation and running simulation

The background mesh is created with blockMesh.

```
>blockMesh
```

Equal to the single region case, the command `surfaceFeatureExtract` creates the **eMesh** files from the stl files with the geometry data. Also the folder *extendedFeatureEdgeMesh* is created in the constant directory. The creation of eMesh files with the command `surfaceFeatureExtract` is not obligatory. This step is only necessary, if certain edges need to be refined.

```
>surfaceFeatureExtract
```

For performing the meshing in parallel, the geometry needs to be decomposed prior to running *snappyHexMesh*. Depending on the number of subdomains, defined in the *decomposeParDict*, the processor folders are created accordingly.

```
>decomposePar
```

*Note: It is **recommended**, not to use the scotch method to decompose the region. Rather, the hierarchical or the simple method should be used. In case of scotch method, errors can occur while executing snappyHexMesh or while reconstructing the mesh.*

In order to prevent the creation of the folders 1, 2 (castellation and snapping features are turned on while layering is turned off) and only keep the final time step folder with the final mesh, the command *-overwrite* can be added after *snappyHexMesh.* In this case, only one folder, *0,* is created with the files **pointLevel** and **cellLevel**. The mesh data in this case is located in constant/polyMesh.

```
>mpirun -np 4 snappyHexMesh -parallel -overwrite
```

*Note: If castellatedMesh and snap are set on true in the **snappyHexMeshDict**, only the snapped mesh is stored, whereas the intermediate step castellatedMesh is overwritten. If castellatedMesh, snap and addLayers are set on true in the **snappyHexMeshDict**, only the layered mesh is stored and the previous intermediate steps castellatedMesh and snap are overwritten.*

In this case only the steps *castellatedMesh* and *snap* are set to *true*, as these steps are applied to the whole mesh. The following command reconstructs the final mesh:

```
>reconstructParMesh -constant
```

After running the command `reconstructParMesh`, the following message appears in the terminal window, which can be ignored:

This is an experimental tool, which tries to merge individual processor meshes back into one master mesh. ...
Not well tested & use at your own risk!

After this step all the regions are meshed but the meshes are connected and needs to be split. In the meshing step each region cells are marked with a flag and this flag will be used in the next step to split the mesh. Mesh regions can be split using the following command which split the mesh based on the flagged cellZones and overwrite the old meshes in the polyMesh directories in the region folders (if any exist):

```
>splitMeshRegions -cellZones -overwrite
```

With the mesh ready, the next step is to apply appropriate field values to each region, according to the *changeDictionaryDict*. This command needs to be repeated for each region, with the name of the region specified after the prefix `'-region'`.

```
>changeDictionary -region heater
…
>changeDictionary -region topAir
```

Now the solver chtMultiRegionFoam is ready to be run.

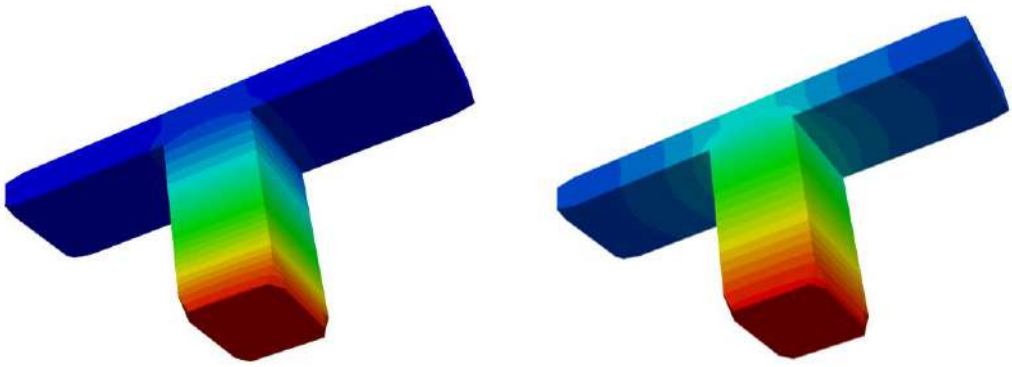
```
>chtMultiRegionFoam
```

Note: chtMultiRegionFoam can also be run on several processors.

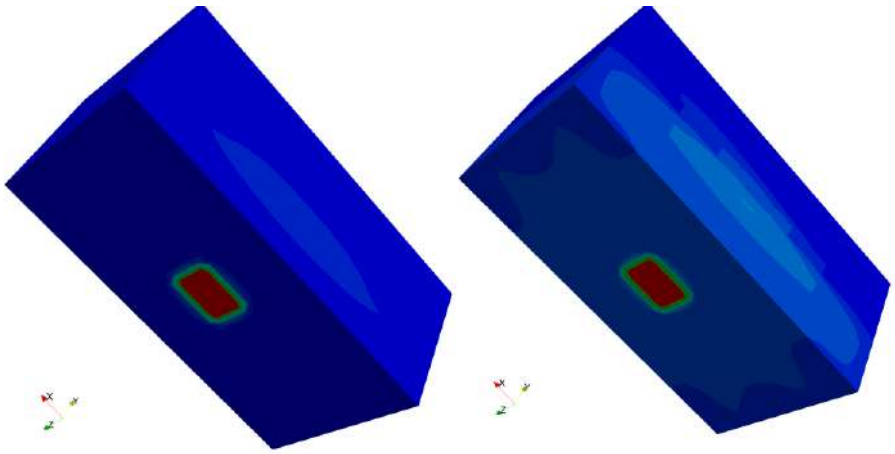
3. Post-processing

The results need to be converted to VTK files for each region with flag `-region`.

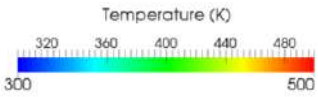
```
>foamToVTK -region heater
…
>foamToVTK -region topAir
```



Temperature profile of heater region at time 15s and 75s

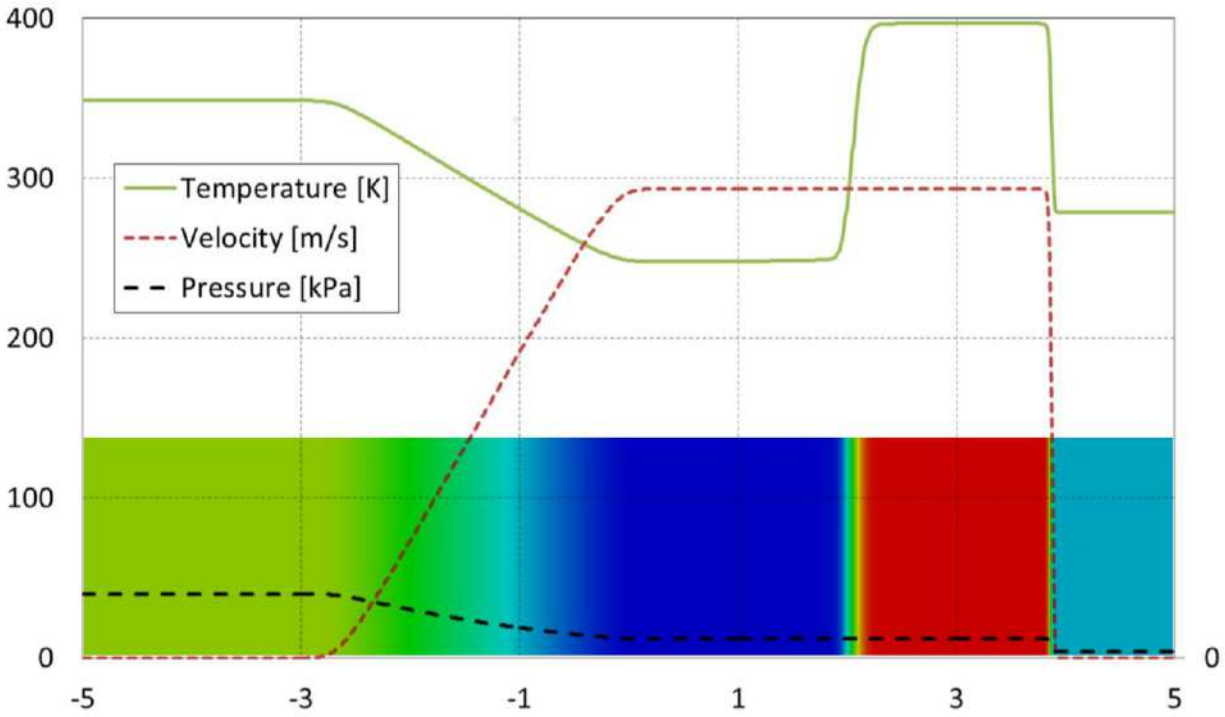


Temperature profile of entire mesh at time 15s and 75s



Tutorial Fourteen

Sampling



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:

- Bahram Haddadi
- Benjamin Piribauer
- Yitong Chen

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:

- Bahram Haddadi

For more tutorials visit: www.cfd.at

Background

1. Introduction to sampling

This tutorial is an introduction to the OpenFOAM® sampling utility. With this utility one can extract sample data from certain selected surfaces or points in ones simulation after or while running the simulation. If data is sampled while running one can use sampling to observe the progress of the simulation and the correctness of the solution without even reaching a write-interval.

Using a *sample* file in the system directory data can be sampled after the simulation or by adding the needed functions to the *controlDict* it can be done while running a simulation.

At the beginning of this tutorial the implementation for sampling using the *sample* and the *controlDict* will be introduced and afterwards the different sampling options in OpenFOAM® will be discussed.

sonicFoam – shockTube

Tutorial outline

Simulate the flow along a shock tube for 0.007 s and use OpenFOAM® sampling utility for extracting the data along a line during the simulation and after the simulation.

Objectives

- Understand the function of sampling and how to use the sampling utility

Data processing

Import your simulation to ParaView to visualize it and analyze the extracted data with sampling tool.

1. Pre-processing

1.1.Copying tutorial

To test the sampling feature, we will use the shockTube tutorial covered in Tutorial Three and extract data over a line between (-5 0 0) and (5 0 0).

`$FOAM_TUTORIALS/compressible/sonicFoam/laminar/shockTube/system`

1.2.system directory

1.2.1.sample dictionary

The *sample* file can be found in the system directory.

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

type sets;
libs          ("libsampling.so")

interpolationScheme      cellPoint;

setFormat   raw;

sets
(
    data
    {
        type            uniform;
        axis            x;
        start           (-4.995 0 0);
        end             (4.995 0 0);
        nPoints         1000;
    }
);

fields                  (T mag(U) p);

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

In the `type` the type of data to be sampled is defined, e.g. `sets` or `surfaces`. The different options for `interpolationScheme` and `setFormat` will be discussed in a later section.

In the `sets` sub-dictionary each set of data should be given a name, which is freely chosen by the user, in this case the name is simply '`data`'. In the bracket for the set of data, we need to specify the following criteria:

- `type`: specifies the method of sampling. Here uniform was chosen to make a sample on a line with equally distributed number of points.
- `axis`: to define how the point coordinates are written. In this case, x means that only the x coordinate for each point will be written.
- `Start/end`: the endpoints of the line-sample are defined
- `nPoints`: number of points on our line

Outside of the data and sets bracket in the `fields` we have to define which fields we want to sample.

1.2.2.controlDict

To have the option to sample for each time step instead of each write-interval or sampling while the solver is running; instead of the *sample* dictionary additions in the *controlDict* are needed.

In this part one will change the *controlDict* of the shockTube tutorial so that our line-sampling from before will be executed while running, and per time step.

Add the following code to the end of the `function` sub-dictionary in the *controlDict*.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
…

functions
{
    #includeFunc mag(U)

    linesample
    {
        type                        sets;
        functionObjectLibs          ("libsampling.so");
        writeControl                timeStep;
        outputInterval              1;

        interpolationScheme        cellPoint;

        setFormat    raw;

        sets
        (
           data
            {
                type               uniform;
                axis              x;
                start             (-4.995 0 0);
                end               (4.995 0 0);
                nPoints           1000;
            }
       );

       fields                    (T mag(U) p);
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
```

`linesample` sub-dictionary includes the settings for the sampling tool. Any arbitrary name can be chosen instead of `linesample`. The chosen name will be the name of the folder in the *postProcessing* directory after running the solver.

Inside our linesample sub-dictionary:

- `type`: `sets` or `surfaces` can be chosen. More types will be covered in a later section.

- `functionObjectLibs`: provides the operations needed for the sampling tasks.

- `writeControl`: specifies the intervals in which sampling data should be collected in the case of timeStep, depending on the `outputInterval`, sampling data will get saved in dependence of the `timeStep`. In the case of

`outputInterval` being equal to 1, every time step sampling data will be saved. Changing the interval to 2 means that data will be saved every 2 time steps.

2. Running simulation

To run the Tutorial go to your case directory in the terminal and use the following commands:

```
>blockMesh
>setFields
>sonicFoam
```

3. Post-processing

After sonicFoam solver finishes running, based on your sampling approach the following steps should be performed:

3.1.sample dictionary

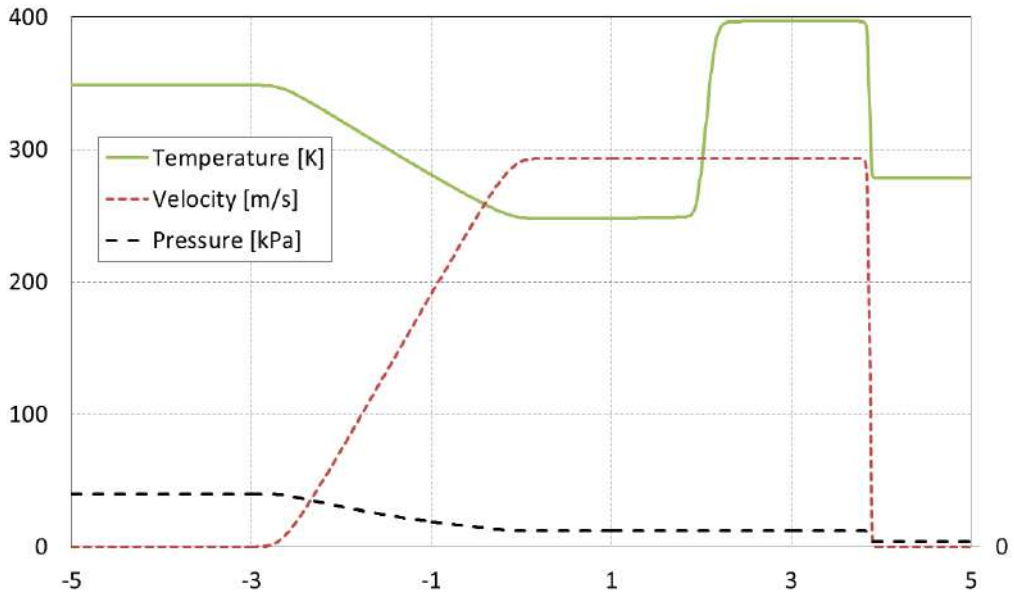
use the sample command to extract your sample-data.

```
>postProcess -func sample
```

A new folder will appear in your case directory named *postProcessing* and in it a folder named *sample*. In this folder all the sampling data will be stored in separate folders for each write-interval.

3.2.controlDict

The *postProcessing* directory and all its subdirectories have been generated after the first time step. Now it can be seen that for every time step a folder is generated instead of only every write interval.



Extracted data using sampling tool after 0.007 s

4. Types of sampling

There are 2 main types of sampling. The sets type, which was used in our example above, and the surfaces type.

In the sets type of sampling different kinds of point samplings, like the line sampling we used before, or some kind of cloud sampling are included. In the surface type whole surfaces are sampled, like near a patch, or on a plane defined by a point and a normal vector.

Let's discuss the similarities between the set and surface types. If the sampling happens in the *controlDict* the 4 entries discussed in the *controlDict* section of this tutorial need to be included for both types. On top of that both types need an interpolation scheme. Here only two of the schemes: cell and `cellPoint` will be discussed. The cell scheme assumes that the cell centre value as constant in the whole cell. The `cellPoint` scheme will carry out linear interpolation between the cell centre and vertex values. Lastly the field bracket looks the same for both cases.

4.1. sets

All sets need a `setFormat`, for example csv which needs to be included after the `interpolationScheme`.
After that the sets sub-dictionary begins where a bracket with an arbitrary name begins in which the sets sampling type and the geometrical location of the sampling points will be chosen. In the following a few of sampling types will be discussed.

4.1.1. uniform

This one was used in the above example. A line from a start point to an end point with a fixed number of points evenly distributed along it.

`axis` determines what is written for the point coordinate in the output file. `distance` means it will only write the distance between sampled point and start point in the file.

```
lineX1
    {
        type                    uniform;
        axis                    distance;

        start                   (0.0201  0.05101  0.00501);
        end                     (0.0601  0.05101  0.00501);
        nPoints                 10;
    }
```

4.1.2. face

This type also samples along a line from a defined start to endpoint, but only writes in the log file for every face the line cuts.

```
lineX2
    {
        type                    face;
        axis                    x;

        start                   (0.0001  0.0525  0.00501);
```

```
        end                         (0.0999  0.0525  0.00501);
    }
```

4.1.3. cloud

The cloud type samples data at specific points defined in the points bracket.

```
somePoints
    {
        type                        cloud;
        axis                        xyz;
        points                      ((0.049 0.049 0.00501)(0.051 0.049 0.00501));
    }
```

4.1.4. patchSeed

The `patchSeed` sampling type is used for sampling patches of the type wall. One can for example sample the surface adsorption on a wall with this type.

```
patchSeed
    {
        type        patchSeed;
        axis        xyz;
        patches     (".*Wall.*");
        maxPoints   100;
    }
```

Please note that for patches only a patch of type wall can be used. If you choose a wrong type, nothing will be sampled and you receive no error message.

4.2. surfaces

All surfaces need a `surfaceFormat` specified. Practical formats would be the vtk format which can be opened with paraview and the raw format which can be used for gnuplots. Instead of the `sets` bracket now a `surfaces` bracket must be used and the `type` is of course also `surfaces`. Inside the surfaces brackets one can now sample an arbitrary number of surfaces, each in its own inner brackets. The different types of surface sampling like the `plane` in the example below will be discussed in the next sections.

```
        type                surfaces;

        interpolationScheme  cellPoint;
        surfaceFormat        vtk;

         fields
         (
             U
         );

         surfaces
         (
               yoursurfacename
               {
                type          plane;
                basePoint     (0.1 0.1 0.1);
                normalVector  (0.1 0 0);
                triangulate   false;
               }
         );
```

4.2.1. plane

The type plane creates a flat plane with an origin in the `basePoint` normal to the `normalVector`. This `normalvector` starts in the origin, not in the `basePoint`. To turn the triangulation of the surface off one has to add `triangulate false`.

```
constantPlane
    {
        type            plane;    // always triangulated
        basePoint       (0.0501 0.0501 0.005);
        normalVector    (0.1 0.1 1);

        //- Optional: restrict to a particular zone
        // zone        zone1;

        //- Optional: do not triangulate (only for surfaceFormats that support
        //            polygons)
        //triangulate     false;
        //interpolate     true;
    }
```

One can also set a new origin for the `basePoint` and `normalVector` with

```
coordinateSystem
        {
            origin      (0.0501 0.0501 0.005);
        }
```

4.2.2. patch

A sampling of type `patch` can sample data on a whole patch. Please note that while the syntax looks the same as in the `patchSeed` case, also patches that are not of type wall work. Default option will triangulate the surface, this can be turned off with `triangulate false`.

```
walls_interpolated
    {
        type            patch;
        patches         ( ".*Wall.*" );
        //interpolate     true;
        // Optional: whether to leave as faces (=default) or triangulate
        // triangulate     false;
    }
```

4.2.3. patchInternalField

Similar to the `patch` type, this type needs a patch on which it samples. It will sample data that's a certain distance away in normal direction (`offsetMode normal`). There is also the option to define the distance in other ways seen in the commented section of the code.

Note: While the sampling happens not on the patch but a distance away from it, the geometric position of the sampled values in the output file will be written as the position of the patch.

Once again the default triangulation can be turned off with `triangulate false`.

```
nearWalls_interpolated
    {
        // Sample cell values off patch.
        // Does not need to be the near-wall
        // cell, can be arbitrarily far away.
        type            patchInternalField;
        patches         ( ".*Wall.*" );
        interpolate     true;


        // Optional: specify how to obtain
        // sampling points from the patch
        // face centres (default is 'normal')
```

```
        //
        //  //- Specify distance to
        // offset in normal direction
        offsetMode  normal;
        distance    0.1;
        //
        //  //- Specify single uniform offset
        //  offsetMode  uniform;
        //  offset      (0 0 0.0001);
        //
        //  //- Specify offset per patch face
        //  offsetMode  nonuniform;
        //  offsets     ((0 0 0.0001) (0 0 0.0002));


        // Optional: whether to leave
        // as faces (=default) or triangulate
        // triangulate     false;
    }
```

4.2.4. triSurfaceSampling

With the `triSurfaceSampling` type data can be sampled in planes which are provided as a trisurface stl file. To create such a file one can use the command below. The command will generate a .stl file of one (or more) of your patches.

```
>surfaceMeshTriangulate name.stl -patches "(yourpatch)"
```

Here your patch needs to be replaced with the name of one of your patches defined in the constant/polyMesh/boundary file. Starting the command without the patches option will generate a stl file of your whole mesh boundary. Next make a directory in the constant folder named triSurface if it doesn't already exist and copy the .stl file there. In the code you now have to specify your stl file as the surface. For the source the use of `boundaryFaces` seems to be a good option of the stl file is one of your patches.

```
triSurfaceSampling
    {
        // Sampling on triSurface
        type        sampledTriSurfaceMesh;
        surface     integrationPlane.stl;
        source      boundaryFaces;
        // What to sample: cells (nearest cell)
        // insideCells (only triangles inside cell)
        // boundaryFaces (nearest boundary face)
        interpolate true;
    }
```

Note: Most CAD software can export the surface of 3D drawings as stl files.

4.2.5. isoSurface

The `isoSurface` sampling type is quite different to what was discussed before in this tutorial. Until now all the sampling types had a constant position in space and changing field values at that position were extracted. With the `isoSurface` sampling one tracks the position of a defined value in space. The example below can be copied into the shocktube tutorials *sample file* (of course it needs all the other options needed for surface type sampling).

Using `vtk` for the `surfaceFormat` one can visualize the moving shockwave in space. Note that both the vtk of the sampling and the whole shocktube case can be opened together in paraview to compare the results.
Note that the `isoField` needs to be a `scalarfield`.

```
interpolatedIso
    {
        // Iso surface for interpolated values only
        type            isoSurface;
        // always triangulated
        isoField        p;
        isoValue        9e4;
        interpolate     true;

        //zone            ABC;
        // Optional: zone only
        //exposedPatchName fixedWalls;
        // Optional: zone only

        // regularise      false;
        // Optional: do not simplify

        // mergeTol        1e-10;
        // Optional: fraction of mesh bounding box
        // to merge points (default=1e-6)
    }
```

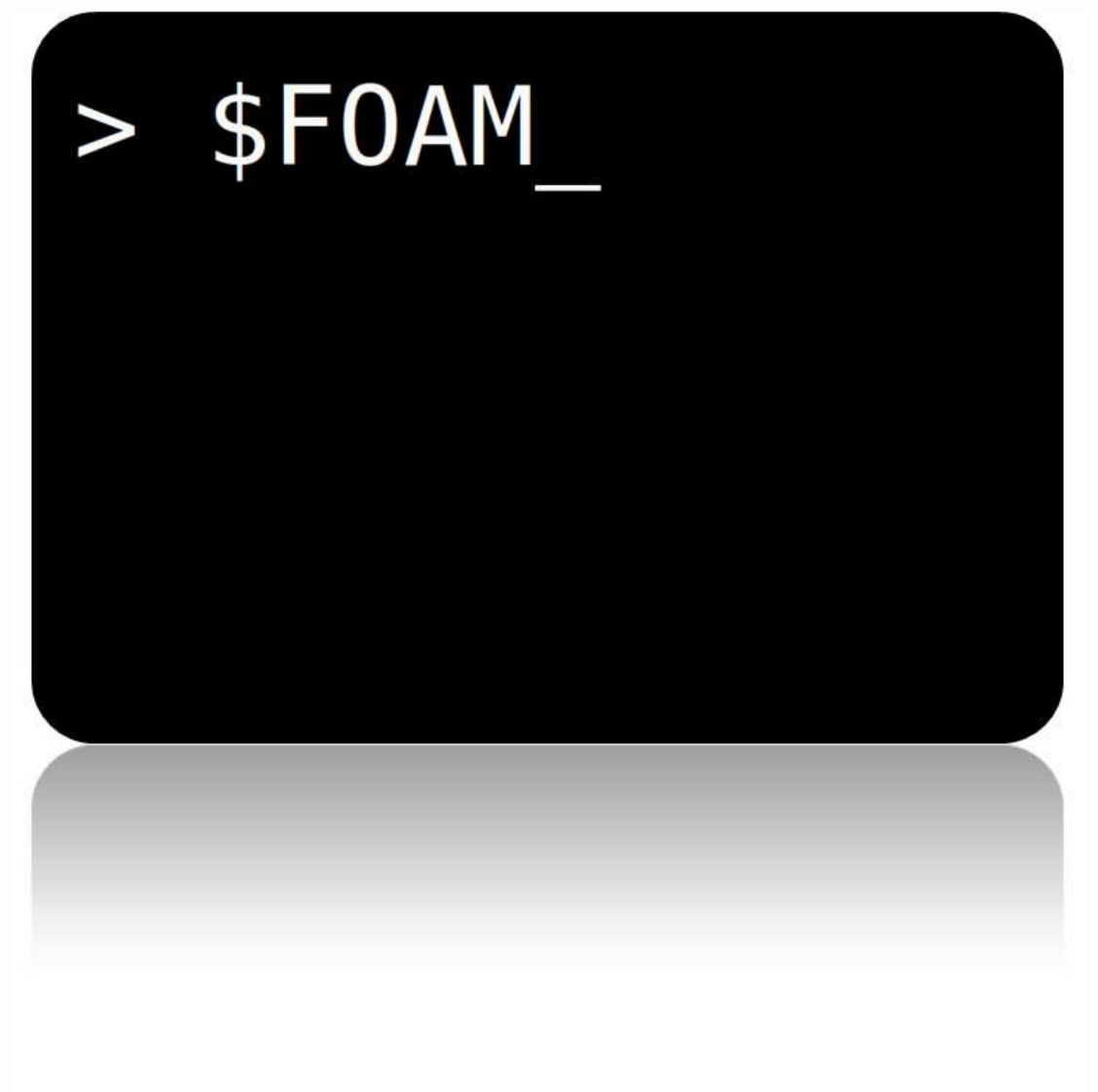
4.2.6. isoSurfaceCell

The `isoSurfaceCell` type is very similar to the one we discussed before, but this one doesn't cross any cell with its surface and doesn't interpolate values.

```
constantIso
    {
        // Iso surface for constant values.
        // Triangles guaranteed not to cross cells.
        type            isoSurfaceCell;
        // always triangulated
        isoField        rho;
        isoValue        0.5;
        interpolate     false;
        regularise      false;
        // do not simplify
        // mergeTol        1e-10;
        // Optional: fraction of mesh bounding box
        // to merge points (default=1e-6)
    }
```

Appendix A

Linux Commands



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- Unix (Linux, Alpha Unix …)

Contributors:

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

Cover picture from:

- Bahram Haddadi

For more tutorials visit: www.cfd.at

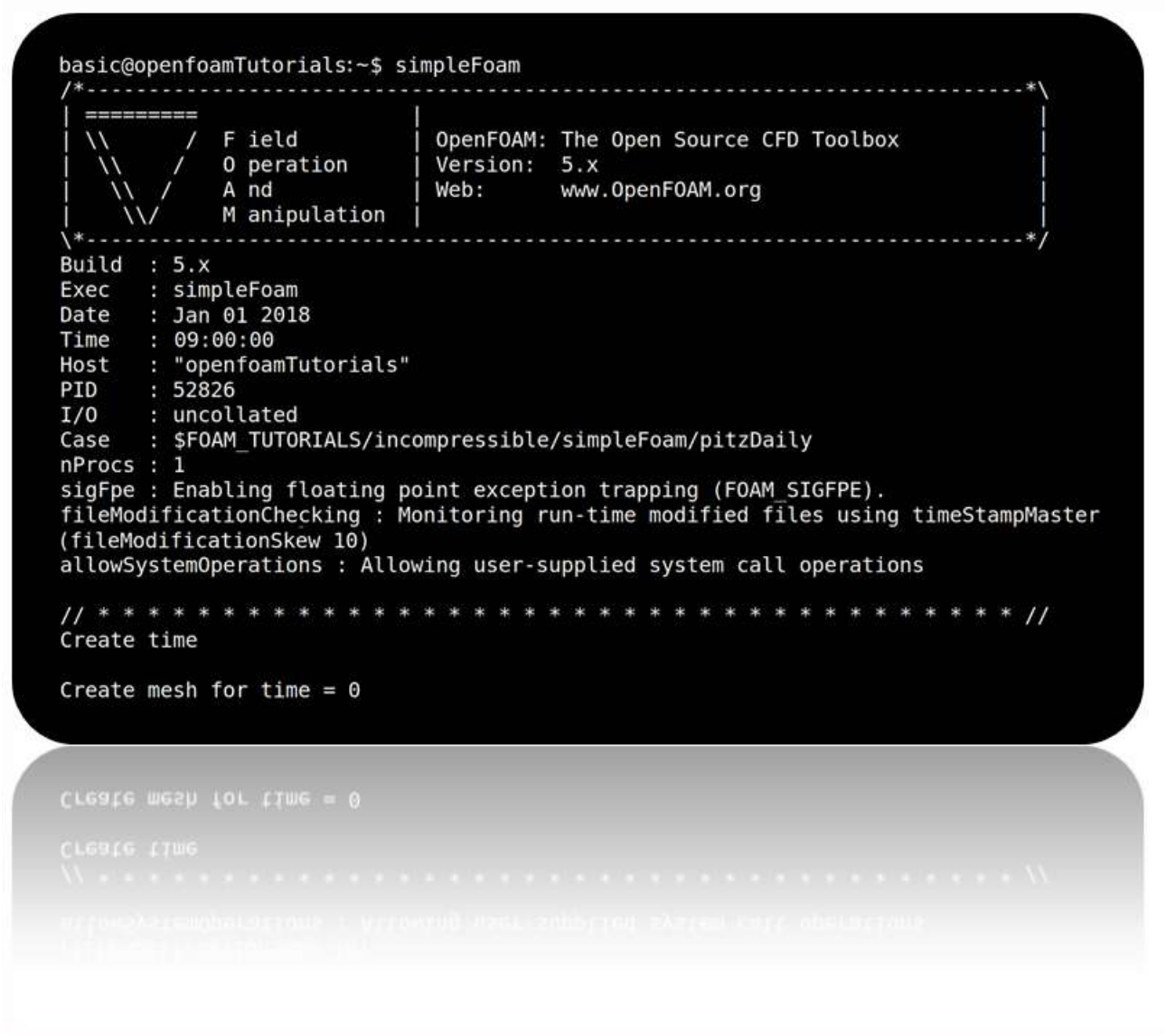
cat, more, less	File viewer with pure read function - in order of ease of operation. In ***less*** with ***pagedown***/***pageup*** you can navigate within the file, with **/** and ***?*** can look for strings, ***q*** can be used for closing ***less***. ***cat*** is back for universally available on Unix.
cd, cd ..	Changing the directory, ***cd ..*** goes one directory up and ***cd ~*** moves to home directory. Important to note is the space between ***cd*** and **..** as opposed to DOS!
cp, cp -r	Copying files or entire directory trees (with **-r** option). Caution: There is no warning or prompt when overwriting existing files! The important thing is that a target has to be always given, at least one **.** which means, copy to the current directory.
ctrl+r	Reverse search, for searching an already typed command in a terminal window.
du, du -s, du -k	Calculates the amount of space consumed in a directory. For safety reasons you should use the ***-k*** option (output in kilobytes), since some systems provide the space in blocks that include only 512 bytes ...
exit	Closing connection (terminal window).
gedit	Text editor with graphical user interface. When working with ***gedit*** some temporary files (originalFileName~) are created, they can be deleted after saving.
grep	Search command for plain-text data sets for lines matching a regular expression.
gzip, gunzip	Compression/decompression program for individual files (as opposed to ***zip***/***unzip***, this can also work on directories or file lists). The great advantage of ***gzip***: Fluent® and OpenFOAM® are able to read and write ***gz*** files directly, which saves about 30-90% space.
kill, kill -9	Stopping processes. For this the process ID is required, which can be found with ***top*** or ***ps***. The ***Exit*** is irrevocable course - but you cannot shoot processes, if you are not the "owner".
ls, ls –la	Lists the contents of a directory, with option ***-la*** also hidden files are displayed, as well as the file size and characteristics.

mc	Program that enables navigation in the text window, esc-keys, may be necessary: ***mc -c***, for navigating through mc use function keys or ***esc+[number]*** combination, e.g. ***F9*** or ***esc+9*** for moving to the menus at the top.
mkdir	Creates a new directory.
mv	Moving or renaming files and directories. Caution: There is no prompt when overwriting existing files!
Nano, pico	The command to run the ***nano*** text editor, a terminal based text editor.
passwd	The command to change the login password.
|	It is known as pipe and is used for merging two commands, redirecting one command as input to another, e.g. ***less|grep*** searches a specified word in the output of file opened with less.
ps, ps –A ps waux	Lists all the processes that were started in the respective command window with the options are all running processes on the system display.
pwd	Shows the current working directory.
rm, CAUTION: rm -fr	Deletes files. The option ***-r*** will also remove directories and files recursively and delete directories, ***f*** (force) prevents any further inquiry. *- Incorrectly applied, this command can lead to irreversible loss of all (private) data. There is no undelete or undo!*
rmdir	Deletes an empty directory.
scp	The copy command over the network - as secure FTP replacement. Also dominates the ***-r*** (recursive) option. Usage: ***scp*** source file destination file with source and the destination format can be USERNAME@ COMPUTER.DOMAIN:PATH/TO/FILE. Source or target can of course also be created locally, then (your) user name and computer are not required.
ssh	Telnet replacement with encryption. On Windows, for example, implemented with putty.

tail, tail -f	File viewer, the default outputs the last 10 lines of a file. With option ***-n XX*** can spend the last XX lines, with the ***-f*** option, the command is running from those lines, which are attached to a file. The command is therefore perfect for watching log files.
top	Displays a constantly updated list of all running processes, with process ID, memory and CPU usage. For processes of one user ***top [username]*** should be used, and for quitting ***q*** or ***ctrl+c*** should be applied.
vi, vim	File editor. For forward searching use /, for backward searching use ***?***. For exiting ***esc+:x***. ***nano*** or ***pico*** are recommended for beginners, which are easier to handle.

Appendix B

Running OpenFOAM®



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:

- Bahram Haddadi

Contributors:

- Christian Jordan
- Bahram Haddadi
- Jozsef Nagy
- Sylvia Zibuschka
- Yitong Chen

For more tutorials visit: www.cfd.at

Part A) Running OpenFOAM® on a Local Linux PC (or virtual machine):

- Open a terminal
- Go to the OpenFOAM® installation directory (e.g. /opt/openfoam5) in the opened terminal
- Change to the etc directory in the OpenFOAM® installation directory
- Run the following command:

  ```
  >. ./bashrc
  ```

- If a new terminal is opened, the same procedure should be repeated in that in order to activate OpenFOAM® in here.

Part B) Running OpenFOAM® on Remote Computers via SSH (e.g. server):

B-1) Windows:

- Run PuTTY (search for: PuTTY windows).
- Set the following:

  `Category>Session`

  Host name: openhost.university.edu

  Connection type: SSH

  `Category>Connection>SSH>Tunnels`

  Source port: 5901

  Destination: localhost:59**[1]

- Do not forget to press *Add*!

  Please make sure that display is not used by others.

  `Category>Connection>Data`

  Auto-login username: openFoamUser[2]

  `Category>Session`

  Saved Sessions: openFoamUser

- Press *Save.*
- Now choose from "saved sessions" your session (*openFoamUser*) and press *Open*. In the opened Command (Prompt) window, it prompts for your

[1] Display number
[2] Session ID

password. The password is not echoed to the screen and the passwords are case sensitive.

- Immediately after entering your password, your computer will attempt to establish a connection to your server. If it is your first time connecting to that server, you will see a message asking you to confirm the identity of the machine. Make sure you entered the address properly, and type *yes*, followed by the *return* key, to proceed.

- To log out use whatever command is used to logout from the server you are logged into (typically ctrl + d).

B-2) Mac OS X and Linux:

- Open your Terminal application. You will see a window with a $ or > symbol and a blinking cursor. From here, you may issue the following command to establish the SSH connection to your server (be careful about upper case 'L' in the -gL).

```
>ssh -gL 5901:localhost:59** openFoamUser@university.edu
```

- Immediately after issuing this command, your computer will attempt to establish a connection to your server. If it is your first time connecting to that server, you will see a message asking you to confirm the identity of the machine. Make sure you have entered the address properly, and type *yes*, followed by the return key, to proceed.

- You will then be prompted to enter your password. Type or copy/paste your SSH user password into the Terminal. You will not see the cursor moving while entering your password. This is normal. Once you are finished inputting your password, press return on your keyboard. At this point, you will be connected to your server remotely through SSH.

Part C) Running OpenFOAM® in Graphical Interface (VNC):

- Connect to remote machine via SSH connection using part B.

- Make sure VNC Server is installed on the remote machine and it is started (ask administrator for display number, port and other information, for starting VNC Server check FAQ)

- Install the appropriate VNC Viewer and run it (search for: vnc viewer):

  VNC Server: localhost:01

- Press *Connect*

- Press *Continue*

- Enter your password

- Press *Ok*

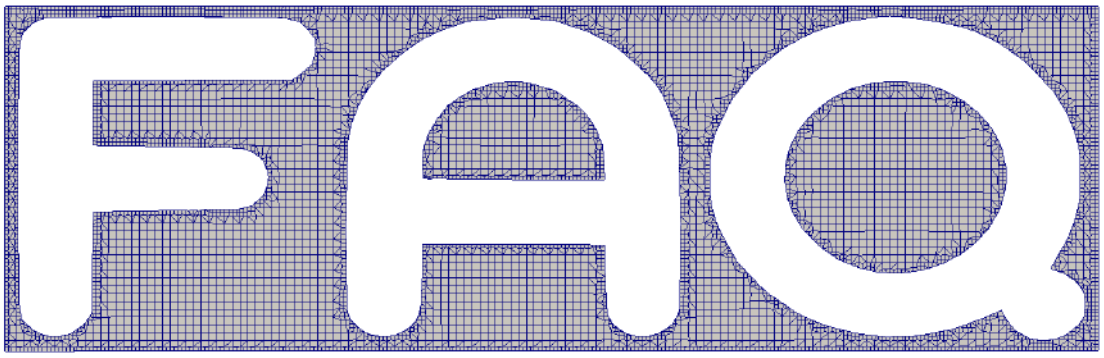
- On VNC desktop open a terminal
- Change to etc directory in OpenFOAM® installation directory
- Execute the following command:

```
>. ./bashrc
```

- If a new terminal in the VNC desktop is opened, the last two steps should be done in that to activate OpenFOAM® in there.

Appendix C

Frequently Asked Questions



How the mesh has been created?!

4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712
- Unix (IRIX, Alpha Unix …)

Cover picture from:

- Bahram Haddadi

Contributors:

- Bahram Haddadi
- Christian Jordan
- Jozsef Nagy
- Sylvia Zibuschka
- Yitong Chen

Q - What should I do in case of a GAMBIT failure?

A - e.g. Program stops responding:

- Type "ps" in the command window, search for the GAMBIT process number.
- "kill -9 PROCESS NUMBER" Enter

GAMBIT creates lock files, which must also be deleted, otherwise it is not possible to open of the affected files:

- "rm *. lok" Enter

Furthermore, "junk" (temporary files from GAMBIT) should be disposed of:

- "rm -fr GAMBIT.xxx" erases the complete directory, xxx again is the process number.
- If you have forgotten, to save before the crash, you should copy the file "jou" (it contains all the commands that have been executed and can be processed automatically in GAMBIT) from the directory, to resume its status before the crash.

Q - How can I prevent typing long commands in the terminal for couple of times?

A - Using curser keys to navigate line by line.

Type beginning of the command and use Tab (auto completion).

By using reverse search, use ctrl+r to search for previous commands typed in the terminal, e.g. typing a part of command show the suggestions and you can navigate through them.

Q - My VNC is not responding from server side?

A - First you should kill your VNC server:

```
>vncserver -kill :[YOUR DISPLAY NUMBER]
```

Restart your VNC server (according to SSH forward):

```
>vncserver:[YOUR DISPLAY NUMBER] -geometry 1600x800 -depth
24
```

Q - I have deleted some of my files accidently. What should I do?

A - Sorry, no recycling or undelete in Linux

Q - Why can I not connect to the server?

A - Check to see if you have an IP address for your network card.

Q - How can I start VNC Viewer from my Linux computer terminal?

A - Use command:

```
>vncviewer :[NUMBER OF LOCAL PORT, e.g. 1 or 2]
```

Q - Error “command not found”?

A - Make sure OpenFOAM® and ParaView are installed correctly. Check Appendix B for starting OpenFOAM®.

Q - Does foamToVTK command not work for chtMultiRegionFoam?

A - Use command:

```
>foamToVTK -region[REGION NAME]
```

Q - Is it possible to export animations from ParaView?

A - Yes, by choosing .ogv file format from "file/save animation" menu. The output will be a video file with .ogv format. In the new ParaView versions (4.3.0) the animation can also be saved using .avi format.

Q - Is there any tool in Linux to convert series of ParaView pictures to video?

A - Yes, command line tool ffmpeg:

```
>ffmpeg -r [FRAME PER SECOND RATE] -f image2 -i [images names, e.g. rho.%4d.jpg] [OUTPUT FILE NAME].[OUTPUT FILE FORMAT, e.g avi]
```

Q - How can complex geometries be patched?

A - During creating the geometry in the preprocessing software, e.g. GAMBIT, create volume zones, which you will need to patch later (see software user manual for creating regions in each software). For converting the mesh to the OpenFOAM® mesh use the appropriate tool with "-writeZones" flag to import zones to OpenFOAM®, e.g.:

```
>fluentMeshToFoam -writeZones <your mesh>
```

then in the setFieldDict change it like this:

```
regions
(
    zoneToCell
    {
        name air; // region name which you assigned in gambit
        fieldValues
        (
            volScalarFieldValue alpha.water 0 // the value of property
                                              //which you want to patch
        );
    }
);
```

Then after running setFields tool, it will assign the values to that region.

Q - How can I create a bash scripting file for executing couple of command in series?

A - Instead of typing command sequences one by one after each other and executing them. It is possible to put all those commands in a file and execute that file to run them. This is known as "bash scripting".
Bash scripting is typically used in the cases when the same simulation should be run with identical settings a couple of times, but with a few changes. For bash scripting create an empty file (e.g. using nano editor creating text file "go"):

```
> nano go
```

Add the commands to this file (e.g. commands for running blockMesh,

setFields, decomposePar, compressibleInterFoam in parallel mode and reconstructPar):

```
blockMesh
setFields
decomposePar
mpirun -np 4 compressibleInterFoam -parallel >log
reconstructPar
```

Exit the editor and save the file (ctrl+x , y, enter for nano editor).
For changing this file to an executable file, file permissions should be set. By using this command file permissions are displayed:

```
>ls -la go
```

```
-rw-r--r-- 1 e166**** E020D166 73 Aug 23 9:15 go
```

The first 'r' shows that this text file can be read by user, the 'w' shows that user has the permission to write this file, but the '–' sign shows that this file is not executable by the user. To change this permissions execute following command:

```
>chmod u+x go
```

Now this file is executable:

```
>ls -la go
```

```
-rwxr--r-- 1 e166**** E020D166 73 Aug 23 9:15 go
```

Now you can run the simulation by this executable text file:

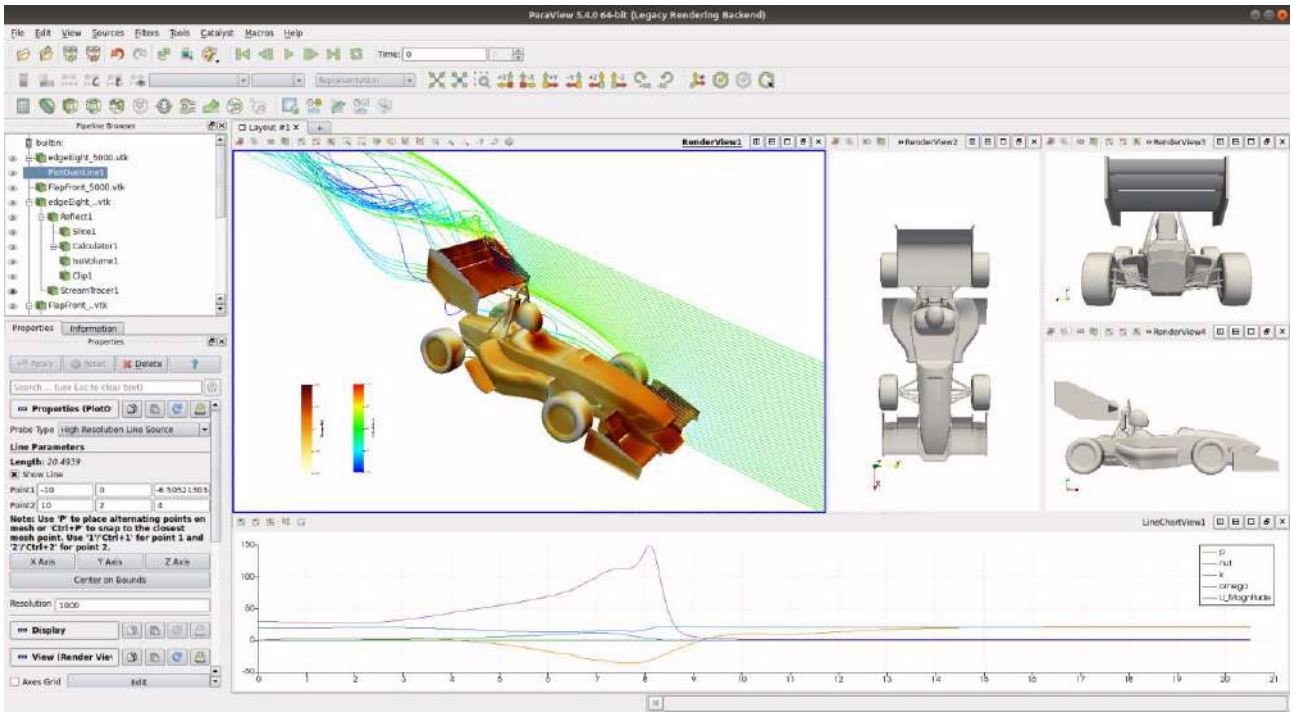
```
>./go
```

After executing the file, the commands added to the file will be executed one by one. In most of the OpenFOAM® tutorials there are **Allrun** and **Allclean** files, which are bash scripts for running the case and cleaning a case, respectively.

Q - How the cover mesh has been created?
A - Error: invalid question!

Appendix D

ParaView



4th edition, Jan. 2018

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- Paraview 5.0.1

Cover picture from:

- Bahram Haddadi

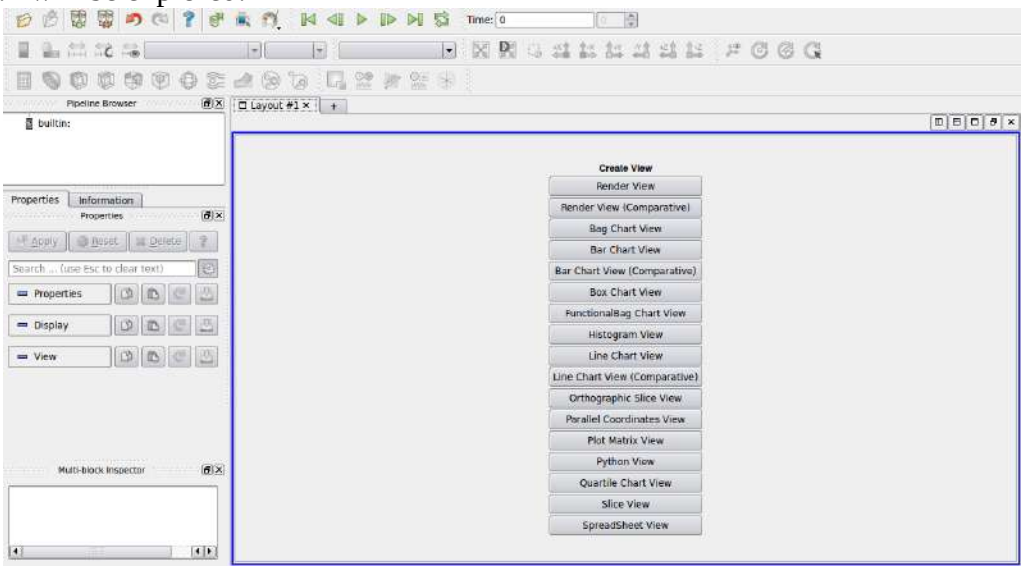
Contributors:

- Bahram Haddadi
- Jozsef Nagy
- Sylvia Zibuschka
- Yitong Chen

For more tutorials visit: www.cfd.at

1. Introduction to ParaView

The post-processing application for OpenFOAM® is ParaView, which is a free, open source program. In this tutorial, different features and tools available in ParaView 5.0.1 will be explored.



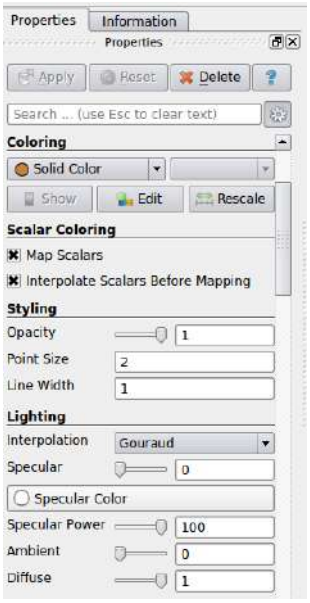
ParaView Interface

The tree structure ("pipeline") of ParaView helps the user to easily choose and display suitable sub-models for creating the desired image or animation. Adding a mesh or velocity vectors to a contour plot of pressure is an example of this functionality.

For generation operations, use the OpenFOAM® command *foamToVTK* to convert OpenFOAM® files into readable formats for ParaView. Then open the .vtk file and press the green *Apply* button in the Properties panel. The reset button is used for resetting the window and deleting the selected operation.

2. ParaView Interface

2.1. Properties Panel



Properties panel

- **Colouring**

  The drop-down menu for solid colour allows different field variables to be chosen and viewed, for example, pressure and velocity magnitude.
  The Rescale button allows the data range to be adjusted to fit the data, as sometimes the max/min data range are not updated automatically.

- **Scalar Coloring**

  The '*Map Scalars'* option allows the scalar values to be mapped to a specific colour using a lookup table.
  Turning the option *'Interpolate Scalars Before Mapping'* on or off will affect the way the scalar data is visualized with colours. According to the ParaView documentation, if it is turned on, scalars will be interpolated within polygons and colour mapping will happen on a per-pixel basis; if off, color mapping occurs at polygon points and colors are interpolated, which is generally less accurate[1].

- **Styling**

  The opacity of the image can be set (1 = solid, 0 = invisible) in the *Opacity* option.

- **Lighting**

  There are two options for *Interpolation*, *Gouraud* or *Flat*. With *Gouraud* shading enabled, normals are defined only per point and no face normal is needed. If the *Interpolation* is changed to *Flat*, only the face normals will be computed and used for lighting, note that this option is not suitable for objects with smooth surfaces [2].

- **Backface Styling**

  This is an advanced feature in ParaView that enables the backface style of a wire frame object to be changed.

- **Transforming**

  The Transform filter allows you to translate, rotate and change the size and the origin of the data sets.

- **Miscellaneous**

  By default ParaView triangulate the cells and shows them as triangles. For disabling this uncheck the "*Triangulate*" option in the Miscellaneous section of the Properties panel.

- **Glyph Parameters**

  The Glyth Parameters filter generates a glyph, which can be arrow, cone, box, cylinder, line, sphere or a 2D glyph. The glyth is generated at each point in the input dataset[3]. Depending on the type of glyph chosen, different options are available to orientate, scale and size the glyph.

- **Orientation Axes**

  The Orientation Axes feature controls an axes icon in the image window (e.g. to set the color of the axes labels x, y and z).

- **Lights**

  The lighting controls options appear when clicking on the *Edit* button. For producing images with strong bright colors (e.g. isosurface) Headlight of strength 1 is appropriate.

- **Background**

  The background color of the layout can be chosen from the drop-down menu, with types *Single color, Gradient* and *Image* available.

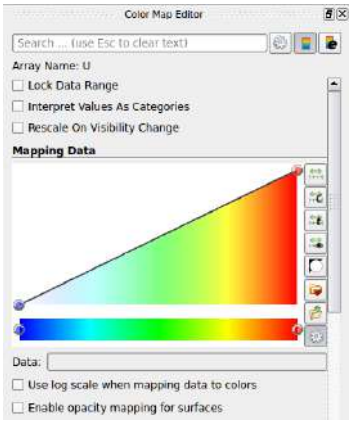
2.2. Button toolbars

Button toolbars

Pull-down menus at the top of the main window and the major panels, in the toolbars below the main pull-down menus increase the functionality of ParaView. The function of each button can be easily understood by its icon, also any button description can be found in the Help menu (keeping the mouse over an icon without clicking on it will also give a short explanation on its functionality).

A feature worth mentioning is the drop-down menu next to the *Reset* button, this provides the options of the different ways of presenting the mesh. To see the structure of the mesh, use *Surface with Edges*; and to see both the cell structure and the interior of the mesh, use *Wireframe*.

2.3. Color Map Editor Panel



Color map editor

The *Choose preset* button allows the color scheme of the scale to be chosen, a common color scheme used is Blue to Red Rainbow. The *Rescale to custom range* button allows the maximum and minimum values of the color scale to be freely chosen by the user.

Another important feature can be used by clicking the button *Edit color legend properties* on the top right of the panel, this allows the scale title, font style to be changed.

3. Manipulating the view

3.1. Contour plots

Clicking on the *Contour* button in the Button Toolbars creates a contour plot. The contour filter operates on any type of data set, but requires the input to have at least one point-centered scalar (single-component) array. The output of this filter is polygonal.

The chosen scalar field can be selected from a pull down menu. If the case is a 3D case module, the contours will be a set of 2D surfaces that represent a constant value. The *Isosurfaces* list in the Properties panel allows the user to specify the values at which the isosurfaces are computed.

3.2. Introducing a cutting plane

Creating contour plots across a plane is more convenient than isosurfaces. Cutting planes are the tools which can be used for this purpose, to create surfaces. This can be done by clicking on the Slice button in the Button Toolbars. A cutting plate can be manipulated and repositioned. In a similar way, the contour lines can also be derived out of planes.

By default ParaView triangulate the cells and shows them as triangles. For disabling this uncheck “triangulate the slice” option in the Properties panel of the slice.

3.3. Streamlines

To create tracer lines, click on the Stream Tracer button in the Button Toolbars. Tracer points can be along a line or points, and this can be chosen in the Seed Type drop-down menu in the Seeds section of the Properties panel. Usually, some trial and error is needed for achieving the desired streamlines. The length of steps tracer takes can be changed in the Streamline Parameters section of the Properties panel. A smaller length increases calculation time but increases smoothness. For having high quality images Tubes filter can be used after tracer lines have been created. There are different types of tubes, not only cylindrical.

3.4. Vector plots

The Glyph filter is used for creating vector plots. Scale Mode menu in the properties panel is used for:

- Setting the length of a vector, weather to be proportional to vector magnitude or not, all with the same length (Vector).

- Controlling the base length of the glyphs (set Scale Factor).

4. Data Analysis

4.1. Plot over time

This option is available by clicking the *Plot Selection Over Time* button in the Button Toolbars. This allows the data at one point to be plotted over the entire time range.

4.2. Plot over line

This option allows the data points to be plotted along a line at a specific time step. Click on the *Plot Over Line* button. The Cartesian coordinates of the beginning and ending points of the line can be specified in the Properties panel. Several variables

can be plotted at the same time, to turn each variable on or off and to change its legend name, use the Series Parameters section in the Properties panel.

4.3. Integrate Variables

The *Integrate Variables* option is selected from the Filters menu. This tool integrates point and cell data over lines and surfaces. It also computes length of lines, area of surface, or volume[4]. Different data types available are *Point Data*, *Cell Data*, or *Field Data*; this can be chosen in the *Field Association* section in the Properties panel.

5. Exporting Data

5.1. Image Output

For creating a screenshot of the graphs the easiest way is Save Screenshot from File menu. After selecting it in the opened window the picture resolution can be set, and by locking the aspect ratio, changing image resolution in one direction cause change in its resolution in the other direction respectively. For high quality images a resolution of more than 1000 pixels is a good choice.

5.2. Animation Output

Some animations can be saved in ParaView by selecting the *Save animation* option in the File menu. The resolution and number of frames per time step can be specified. You can save your animation by assigning a name and choosing the file format. The most suitable file format is *.ogv*.

5.3. Data Output

The field values of a chosen variable (e.g. temperature or pressure) can be exported into ExCel using the Save Data option in the File menu. The precision of the writer can be chosen and there is an option to export data from all time steps.