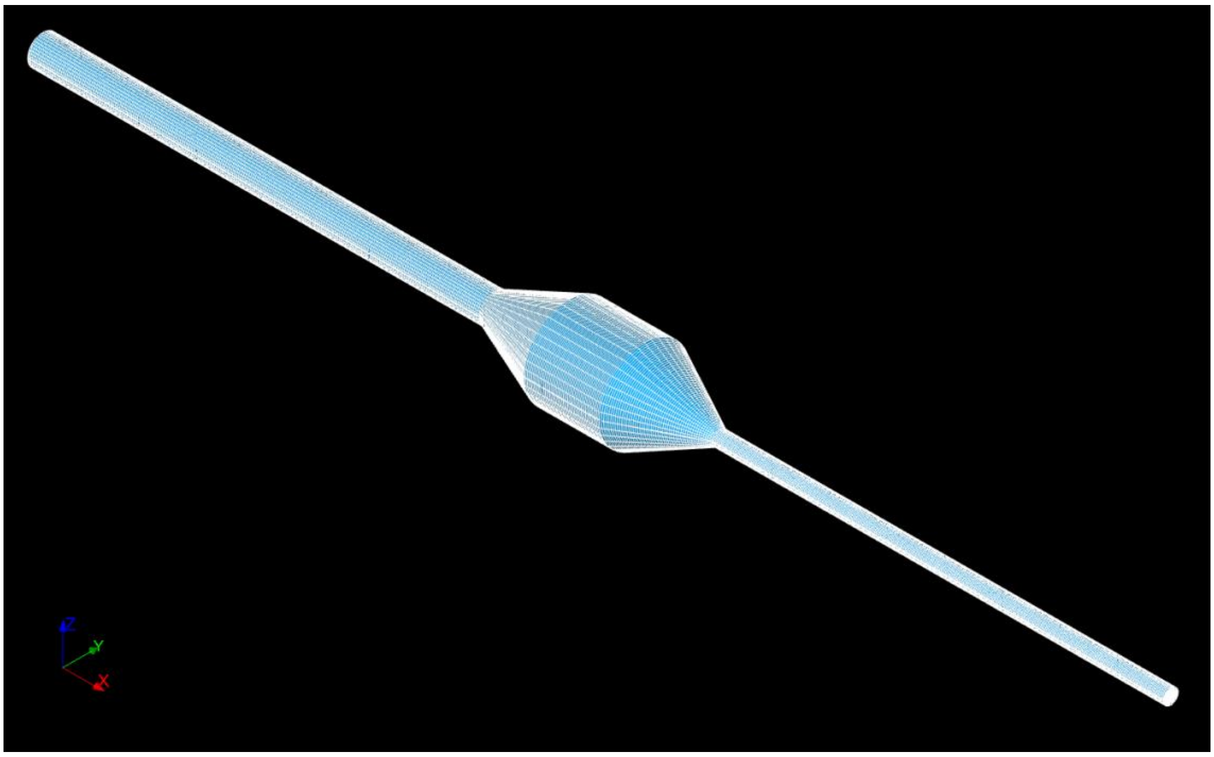
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2021

Process Technology

Complex Mesh Generation with OpenFOAM



Anne Marie Lande

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2021

Title: Complex Mesh generation with OpenFOAM

Number of pages: 125

Keywords: CFD, simpleFoam, Salome, diverging converging pipe with belly section, 3D, mesh generation software, tutorial, hexahedral mesh, OH-grid (butterfly grid)

Student: Anne Marie Lande

Supervisor: Joachim Lundberg and Knut Vågsæther

External partner: -

Summary:

In the course Computational Fluid Dynamics at the University of South-Eastern Norway there have been some requests for more advanced geometry and meshing tools than the lectured blockMesh dictionary and meshing tool in the open-source software OpenFOAM.

In this report, theory concerning mesh quality parameters and turbulence modeling is presented and an evaluation of different open-source meshing tools is made. Salome is evaluated to be the most user-friendly tool and is used to create and mesh a 3D model of a diverging converging pipe with belly section. 3 meshes with a hexahedral OH-grid and different wall treatments are constructed and imported to OpenFOAM. 4 different simulations are run with the steady-state turbulent incompressible flow solver simpleFoam, applying the turbulence models *standard* $k-\varepsilon$ or $k-\omega\ SST$ to model the air flowing through the pipe. A student tutorial describing the workflow of one of the cases is made.

An evaluation of basic fluid mechanics checkpoints, symmetry and residuals did not indicate numerical errors in the solutions. The simulation cases generally obtained results that agreed well with the analytical solutions, even though the complex flow field of turbulent, transitional and laminar flow was modeled with turbulence models. The adjusted wall functions gave slightly better results than the default wall functions for the $k-\varepsilon$ model. The $k-\varepsilon$ model on Mesh B obtained pressure results that agreed well with the analytical solution. The $k-\omega\ SST$ model generally predicted the velocity field best, and also showed consistent results in terms of pressure loss. Case A did not fulfill the wall function requirements and obtained diverging pressure loss results. The OH-grid topology was evaluated to give good mesh optimality for the full resolution approach and insufficient mesh optimality for the wall function approach.

Preface

This is a master's thesis written at the University of South-Eastern Norway. The master's thesis is part of the last course at the master's degree study in Process Technology and constitutes 30 credits.

I would like to thank my supervisor, associate professor in Process technology and laboratory teacher in CFD, Joachim Lundberg. His guidance, help and assistance have been invaluable throughout the work of this thesis.

Further, I would like to thank Olas Bil AS for letting me borrow a suitable computer for the work on this thesis and providing me with the paid commercial software VMWare.

Avaldsnes, 18th May 2021

Anne Marie Lande

Contents

Nomenclature

Latin symbols:

Symbol	**Description**	**Unit**
A	Cross-sectional area of pipe	$[m^2]$
C_f	Skin friction coefficient	$[-]$
C_μ	Turbulent viscosity constant	$[-]$
D	Pipe diameter	$[m]$
f	Pipe friction factor	$[-]$
f, g	Functions	$[-]$
g	Gravitational acceleration	$[m/s^2]$
in	Location of pipe inlet	$[-]$
K	Obstruction loss factor	$[-]$
k	Turbulence kinetic energy	$[m^2/s^2]$
L	Length of a pipe section	$[m]$
l	Turbulence length scale	$[m]$
out	Location of pipe outlet	$[-]$
P	First computational interior node	$[-]$
p	Static pressure	$[Pa]$
p_0	Stagnation pressure	$[Pa]$
p_k	Kinematic pressure	$[m^2/s^2]$
Q	Volume flow	$[m^3/s]$
Re	Reynolds number	$[-]$
Δt	Time step	$[s]$
T	Temperature	[°C]
T_i	Turbulence intensity	$[-]$
U	Velocity	$[m/s]$
U_{max}	Maximum velocity	$[m/s]$
u_τ	Friction velocity	$[m/s]$
u^+	u plus	$[-]$
Δx	Length of a grid cell	$[m]$
$x_{fd,h}$	Hydrodynamic entry length	$[m]$
Δy	Height of a grid cell	$[m]$
y^+	y plus	$[-]$
y	Distance from wall	$[m]$
Δy_P	Distance from surface to nearest node P	$[m]$
z	Elevation	$[m]$

Greek symbols:

Symbol	Description	Unit
α	Skewness angle	$[°]$
δ	Boundary layer thickness	$[m]$
ε	Turbulence dissipation rate	$[m^2/s^3]$
ϵ	Roughness height of the pipe wall	$[m]$
μ	Dynamic viscosity	$[Ns/m^2]$
μ_t	Turbulent viscosity	$[m^2/s\ ]$
ν	Kinematic viscosity	$[m^2/s]$
ρ	Density	$[kg/m^3]$
τ	Shear stress	$[Pa]$
τ_w	Wall shear stress	$[Pa]$
ω	Turbulence dissipation rate	$[s^{-1}]$

Abbreviations:

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
AR	Aspect ratio
CAD	Computer-aided design
CFD	Computational fluid dynamics
GUI	Graphical user interface
HRN	High-Reynolds-number
LRN	Low-Reynolds-number
RANS	Reynolds-averaged Navier-Stokes
RDT	Rapid Distortion Theory
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
SIMPLEC	Semi-Implicit Method for Pressure-Linked Equations-Consistent
SST	Shear Stress Transport
USN	University of South-Eastern Norway
WF	Wall functions

1 Introduction

Computational Fluid Dynamics (CFD) is a course taught at the University of South-Eastern Norway (USN). The CFD software used for the simulation part of the course is OpenFOAM and the typical working method is to define the geometry, create the mesh, solve and post-process. The students are taught how to program a simple 2D geometry and to generate a mesh with the blockMesh meshing tool. However, over the past few years, there have been requests from some students for a simple tool to mesh complex geometries for more applied cases.

In this thesis, different tools for creating and meshing a 3D geometry are assessed, and the user-friendliness of the different tools is evaluated. The recommended tool is used to create a student tutorial. The tutorial describes the full workflow of modeling the geometry, generating the mesh, importing the mesh to OpenFOAM, running a simulation and post-processing.

The thesis is divided into two parts: one theory section and one method section. The theory section explores different parameters for obtaining a good mesh, meshing tools and turbulence boundary layer and modeling theory. The method section presents the case and the results. The geometry, mesh and simulation cases are described, and the results of the simulations are presented and discussed.

A relatively complex flow geometry is chosen for the case. The geometry is a 3D model of a diverging converging pipe with belly section. The geometry is meshed with a hexahedral OH-grid (butterfly grid). 3 meshes with different wall treatments are created. The physics involved in the model is quite complex, because turbulent, transition and laminar flow occur. The air flowing through the pipe is assumed to be steady state and incompressible and is modeled with the turbulence models the $standard\ k-\varepsilon$ and the $k-\omega\ SST$ in the s*impleFoam* solver in OpenFOAM. 4 cases with different meshes and wall treatments are simulated.

The project topic description of the thesis is given in Appendix A.

1.1 Previous work

A literature review has been performed and reveals that CFD simulations are a topic that is extensively covered in the literature. However, the mesh generation stage is typically not described. This claim is supported by Hernandez-Perez et al. [1]. In general, only a basic description of the applied mesh is given, although sometimes additional information about the quality and characteristics of the mesh is presented.

An in dept description of the full workflow from creating and meshing a geometry, importing the mesh in OpenFOAM, to running a simulation has not been found in the literature. However, there are meshing tutorials available on meshing platforms, web pages and YouTube.

In a journal article Gao et. al [2] describes the steps of creating and meshing a neighborhood of buildings in Salome.

In his master's thesis Nour [3] develops a 3D hexahedral butterfly mesh for a bent pipe geometry in Gmsh. He describes the workflow of creating, programming and importing the mesh to OpenFOAM. However, he does not explain how to make the mesh executable in OpenFOAM by setting the correct boundaries and initial conditions in the case directory.

In her master's thesis Liestyarini [4] creates meshes for 3D straight, blind-tee and elbow pipes in Salome and Gmsh. She explains that the meshing stage takes 60 % of the project time, but does not present a description of the process. For the straight pipe geometry, the created mesh is unstructured in the center of the pipe and structured in the rest of the pipe. For the blind-tee and elbow geometries, the generated meshes are unstructured.

1.2 Organization of thesis

The thesis is divided into 5 chapters.

Chapter 1 presents an introduction to the thesis with background information, objectives, methods and scope. Additionally, a short literature review of similar work and the report structure info is provided.

Chapter 2 describes theory about mesh structures and meshing tools. Furthermore, theory about turbulence boundary layers and turbulence modeling are covered.

Chapter 3 presents the geometry, meshes, simulation cases and numerical setup for the CFD analysis. In addition, some information about the student tutorial is given.

Chapter 4 contains comparisons of simulation results with analytical solutions, along with a discussion of the results and models.

Chapter 5 concludes the thesis work. Further work is recommended.

Appendices provide additional information and figures that are not given in the report. The student tutorial is located in the Appendices.

2 Theory

In this section, theory concerning meshes, meshing tools, turbulence boundary layers and turbulence modeling are treated.

2.1 Mesh theory

The result of a CFD simulation is greatly influenced by the *mesh* or *grid*[1]. The quality of the mesh affects the convergence, numerical solution and stability of a simulation. The accuracy of the results and solver convergence is established by a good mesh quality [1]. A good mesh topology is essential for reducing discretization errors [5, p. 302]. Therefore, considerate attention should be given to the mesh generation stage. The grid can determine whether a simulation is a success or a failure. Consequently, research and advancement in the grid generation field are important [6, p. 125].

Jasak [7, p. 245] describes the difference between *mesh quality* and *mesh optimality* in the following manner: "[..] **Mesh quality** is a property of the mesh, specifying the amount of mesh-induced errors expected from a certain mesh. **Mesh optimality**, on the other hand, depends on the interaction between the mesh and the particular problem that is being solved."

2.1.1 Mesh structures

There are many possible mesh structures. Mesh topologies can generally be divided into two groups: *structured* (Figure 2.1) and *unstructured* (Figure 2.2) grids. A structured mesh typically consists of elements that follow a regular cartesian pattern. In 2D domains the cells usually consist of quadrilaterals, while in 3D domains hexahedral cells are typical. An example of a hexahedral cell (hexahedron) is illustrated in Figure 2.3 [8]. In the 2D and 3D domains of a structured mesh, the center cell is attached to four and six neighboring cells, respectively. This pattern makes the computing, coupling and solution algorithms straightforward.

[1] The terms *mesh* and *grid* are considered to have the same meaning.

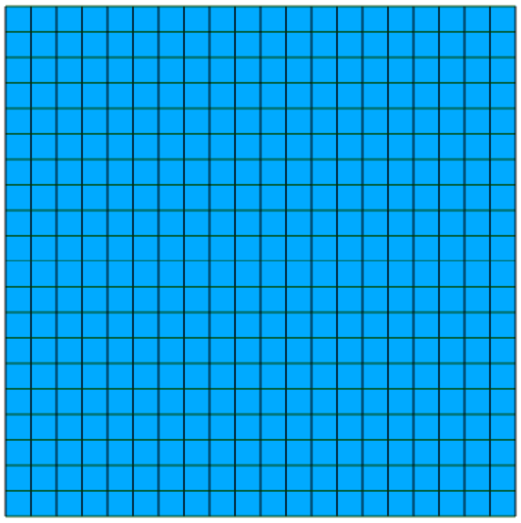


Figure 2.1: Structured mesh.

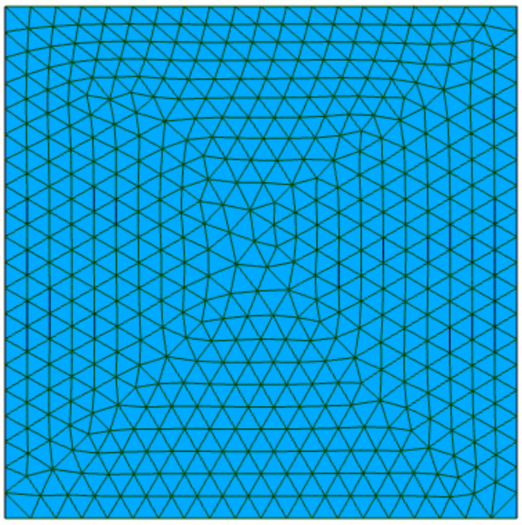


Figure 2.2: Unstructured mesh.

A *single-block* structured mesh is a structured mesh that is meshed as a single block. However, for complex geometries it may be difficult to create a high-quality structured grid configuration without applying additional modifications to the interior geometry. A single-block mesh for a complex geometry typically gives a low-quality mesh with high skewness and non-orthogonality.

Many complex geometries require a *block-structured* grid to ensure a high-quality mesh. In *block-structured* or *multiblock* meshes the geometry is divided into blocks or sub-regions that are meshed separately. Thus, the topology of meshes in sub-regions can differ from each other, and it is possible to refine the grid in desired areas, such as the near-wall area, and to improve the quality of the mesh. However, if a satisfactory block division of the geometry is not possible, the result may be various areas with highly skewed cells and bad mesh quality. Moreover, the creation of fitting interior blocks can be difficult and requires expertise and time. For even more complex geometries, a high-quality structured mesh may be impossible to achieve and an unstructured grid is preferred [5, pp. 305, 309-310, 342]–[6, pp. 136, 139-140].

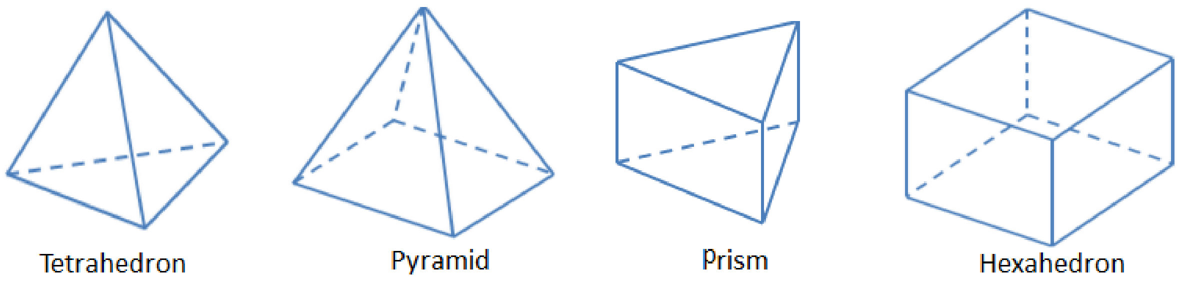


Figure 2.3: 3D shapes [8].

An *unstructured* mesh is defined as a mesh where the elements do not follow a regular pattern and every cell is considered to be a block. In 2D domains the cells usually consist of triangles, while in 3D domains tetrahedral cells are typical. An example of a tetrahedral cell (tetrahedron) is illustrated in Figure 2.3 [8]. Unlike the structured mesh, the center cell in an unstructured mesh is attached to an inconsistent number of neighboring cells. This complicates the solving of the numerical equations, such as the calculation of diffusion and convection fluxes. More complex solver algorithms may be required, which increases the required computing resources and time usage. In addition, difficulties may arise in generating a high-quality mesh along the wall. Triangle and tetrahedral elements close to the wall complicate the solving of the near-wall region and the solution of the boundary layers may be inaccurate.

The major advantage of an unstructured mesh is that it can be used for all types of geometries and is easy and fast to implement. Furthermore, it is easy to refine and adjust the unstructured mesh in specific regions. It is also considered to use the computational resources for complex flows in an efficient manner.

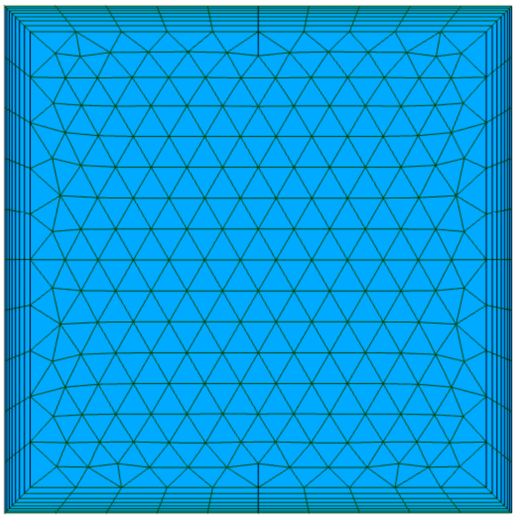


Figure 2.4: Hybrid mesh.

If the grid topology consists of a combination of different geometrical shapes, the grid is called a *hybrid mesh* (Figure 2.4). Figure 2.3 [8] shows examples of different shapes that can be used in various 3D mesh topologies. A hybrid mesh often consists of quadrilateral or hexahedral cells along the wall for better resolution of the boundary layer, whilst triangular, tetrahedral or polyhedral elements are generated on the rest of the geometry. This topology typically improves convergence and accuracy of results [5, pp. 305, 311-312,]–[6, pp. 139-140, 342].

2.1.2 Mesh quality aspects

A low-quality mesh causes discretization errors. Mesh resolution, grading, aspect ratio, skewness and orthogonality are important characteristics that affect the quality of a mesh and the numerical stability and accuracy of results [6, p. 139]–[7].

It is important to make sure that the mesh is not too coarse. The mesh resolution directly affects the accuracy of the solution, therefore the mesh must be constructed so that the solution can be modeled accurately. A sufficiently fine grid ensures that the geometrical shape and flow characteristics are captured. Too low mesh resolution may produce unphysical results.

In addition, the distribution of the grid cells is important. A uniform grid is not always the best arrangement for a given flow problem to ensure mesh optimality. Some regions of the flow domain may need special attention. Problem areas may include regions of rapid changes, high gradients, reattachment, separation or recirculation of the flow, as well as the flow in the near-wall region. In order to resolve all the details in these areas, the mesh can be refined locally. To accurately resolve the boundary layer towards the wall, a configuration of quadrilateral or hexahedral elements is preferred.

If a mesh is refined in certain regions, smooth grading between coarse and fine mesh areas is required to ensure the overall grid quality and accuracy of the solution [6, pp. 125, 133, 141-143]–[7]. Figure 2.5 [9] illustrates the difference between a smooth and steep transition.

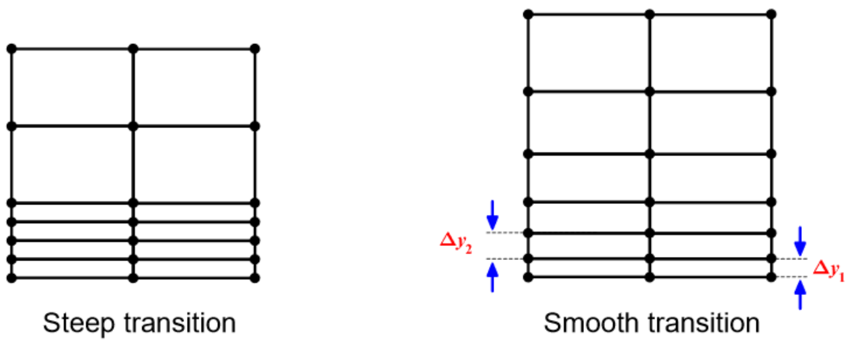


Figure 2.5: Smoothness [9, p. 311].

The *aspect ratio* (AR) is defined as the ratio between the longest and shortest length of a grid cell, as shown in Equation (2.1). Figure 2.6 [10] illustrates the aspect ratio lengths, while Figure 2.7 [11] demonstrates some examples of aspect ratios for different geometrical shapes.

$$AR = \frac{\Delta x}{\Delta y} \tag{2.1}$$

Where Δx $[m]$ is the width of the grid cell and Δy $[m]$ is the height of a grid cell.

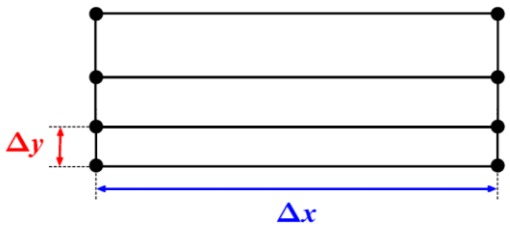


Figure 2.6: Aspect ratio [10, p. 310].

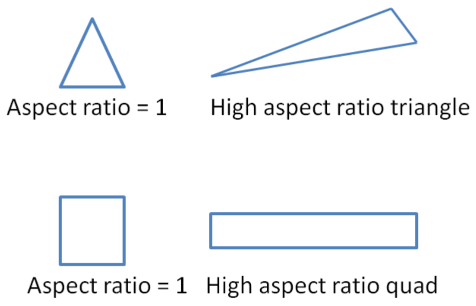


Figure 2.7: Examples of aspect ratios [11].

The value of the aspect ratios of a mesh should be kept as low as possible, especially in important areas of the flow geometry. High aspect ratios can increase the numerical diffusion, decrease the accuracy of the numerical solution and cause convergence issues or divergence. The erroneous effects are especially large if high gradients are present in the regions of high aspect ratios. However, solver settings can influence the effects on the results [6, p. 149], [12, p. 310].

Skewness or *distortion* describes the angle, α [°], between the lines of a cell. The optimal angle is 90°. A cell is considered to be highly skewed if the angle is smaller than 45° or larger than 135°. Figure 2.8 shows an example of a skewed cell. A mesh with highly skewed cells can cause stability issues and poor results. The calculation of the gradient and convective terms are influenced. Moreover, the accuracy of face integrals is reduced to first order, which creates a numerical diffusion error dependent on the degree of distortion. The introduction of *underrelaxation factors* to the solution algorithm may improve the results of a mesh with skewed cells [5, p. 308]–[6, p. 150]–[7], [12, p. 309].

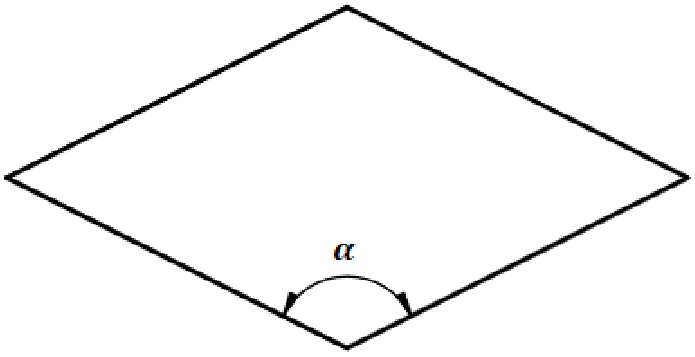


Figure 2.8: Example of a skewed cell.

One of the drawbacks of an unstructured mesh is the skewness of the elements. The unstructured mesh generally gives a zeroth-order of accuracy in OpenFOAM with the standard divergence theorem (Green-Gauss) schemes. It is recommended to use the least-squares gradient on highly skewed unstructured meshes to obtain second-order accuracy [13].

Orthogonality is another important mesh quality aspect. In an orthogonal mesh, the grid lines at the junctions form a 90° angle, as shown in Figure 2.9. In a *non-orthogonal* mesh, on the other hand, the angle of the grid lines at the junctions is not perpendicular. In non-orthogonal meshes, diagonal equality is violated. The boundedness of the solution is affected and the computational molecule increases in size. The calculation of the gradient and Laplacian terms is affected and cross-diffusion effects are generated. As a result, there is a higher risk of errors in the results. For highly non-orthogonal meshes, the introduction of *non-orthogonal correctors* may be necessary [5, pp. 305, 342], [12, p. 308], [7].

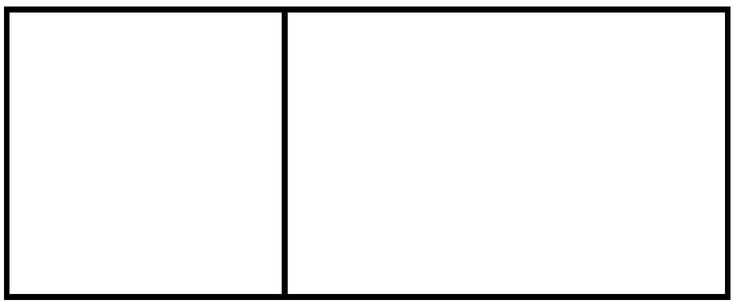


Figure 2.9: Example of an orthogonal mesh.

2.1.3 Mesh topologies for pipes

Many different mesh topologies can be applied to a circular pipe geometry. A circular pipe can be meshed with an unstructured or a structured grid. An example of an unstructured grid topology for a pipe is illustrated in Figure 2.10. As previously mentioned, one weakness of the unstructured mesh is that accurate solving of the boundary layers is difficult. A strength is that it is easy to implement this mesh in many complex pipe geometries, like bends, elbows, and pipes with expanding or contracting cross-sections.

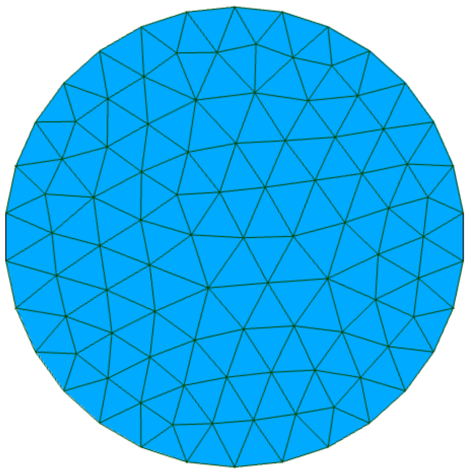


Figure 2.10: Tetrahedral mesh for a circular pipe.

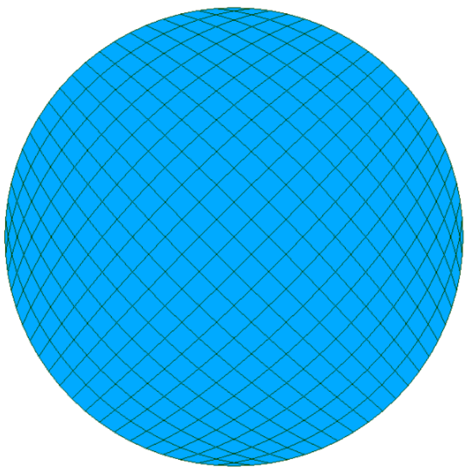


Figure 2.11: H-grid for a circular pipe.

For structured grids, there are many possible pipe topologies. A structured grid can be created with an *H-grid*, an *OH-grid*, an *O-grid* or an *unstructured pave grid*. All of these meshes have advantages and disadvantages. Strengths and weaknesses of the topologies are related to the complexity of the mesh-generation stage, expenditure and how the topology affects the simulation and results. Here, only the H-grid and the OH-grid are explored. Further details about O-grids and pave grids are available in [1].

Figure 2.11 shows an H-grid on a circular pipe geometry. The H-grid is a single-block mesh and consists of orthogonal grid lines. A drawback is that there are 4 vertices at the wall with highly skewed cells, as seen in Figure 2.11. If the number of cells is increased, the skewness at the vertices increases and the mesh quality reduces [1].

The structured OH-grid or *butterfly* topology (Figure 2.12) is described as the ideal configuration for hexahedral meshes in pipes [1]. It is a block-structured mesh where the center of the mesh consists of a cartesian mesh, while the outer mesh is cylindrical. The grid topology allows refinement of the mesh close to the wall. The high-quality hexahedral mesh in the middle of the pipe allows accurate resolution of the flow along the centerline. The topology is widely used in the literature. Some examples of publications with pipes modeled with the OH-grid are the work by Gopalakrishnan and Disimile [14], Hernandez Perez [15], Hernandez Perez et al. [1] and Zang et al. [16]. In [1] a comparison study of hexahedral mesh topologies for a simulation of gas-liquid flow in a pipe was done, and the OH-grid obtained the best results.

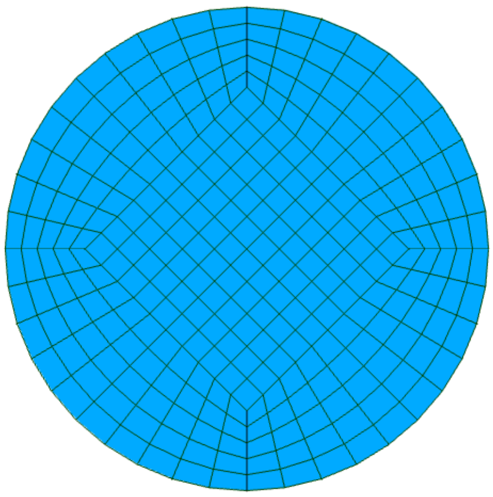


Figure 2.12: OH-grid for a circular pipe.

Of all the described structured grid topologies, the OH-grid employs the best mesh density, orthogonality, skewness and aspect ratio. The main disadvantage with this mesh is that it can be difficult and time-consuming to build, especially for inexperienced users. The geometry must be divided into blocks, which can be a complicated task, especially for geometries that are more complex than a straight pipe. However, some meshing tools have the ability to automate the generation process of such a mesh [1].

The unstructured mesh and the H-grid are the simplest and fastest to create and are suitable for complex geometries.

2.2 Meshing tools

This section explores different open-source meshing tools and compares the two meshing tools Salome and Gmsh.

2.2.1 Open-source meshing tools

blockMeshDict is a dictionary file in OpenFOAM where a geometry and mesh can be programmed. The mesh is generated with the blockMesh command. However, since there is no *graphical user interface* (GUI) and only a basic hexahedral mesh algorithm, the use of this utility is limited to simple geometries and meshes.

There exist many open-source meshing tools that are more advanced than blockMesh. Some of the tools exist within OpenFOAM, like *snappyHexMesh*, some have a GUI, and others have both a geometry and meshing module.

The meshing tool chosen for the student tutorial should be an approved and user-friendly tool. Some essential features of the software to be used in the student tutorial are:

- GUI
- Geometry module

The two criteria are set to ensure the user-friendliness and simplicity of the tutorial. A GUI is desired because it eliminates the need for programming and makes the tool accessible. A geometry module is favorable because it enables geometry and mesh generation in the same software. If a meshing tool does not have a geometry module, the geometry file (e.g., STEP/STL) must first be created and exported from a CAD software.

Different open-source meshing tools have been examined and their features have been considered. Table 2.1 gives an overview of the assessment of the necessary features. As shown in Table 2.1, only Salome, Gmsh and Netgen/NgSolve include both a geometry module and a GUI, and are therefore the only software relevant for the student tutorial.

Table 2.1: Assessment of essential features available in open-source meshing tools.

Meshing tool	**Geometry module**	**GUI**
Salome	X	X
Gmsh	X	X
snappyHexMesh		
cfMesh		
Netgen/NgSolve	X	X
enGrid (with Netgen)		X
Tetgen		

2.2.2 Comparison of Salome and Gmsh

Salome, Gmsh and Netgen/NGSolve are determined to be the only meshing tool candidates for the tutorial. However, Netgen/NgSolve is not evaluated further because Salome and Gmsh are considered to be more popular and widespread software.

To determine the meshing software for the student tutorial, further evaluations of Salome and Gmsh have been conducted. The same geometry[2] was modeled and meshed with an unstructured tetrahedral mesh algorithm in each software. A comparison of the software's user-friendliness and features was performed. The comparison key findings are summarized in Table 2.2. The quality of the generated meshes was not evaluated and compared.

[2] A diverging converging pipe with belly section, same geometry as in the tutorial.

Table 2.2: Comparison of Salome and Gmsh.

	Salome-CFD 8.5	**Gmsh 3.0.6**
GUI	yes	yes
Object browser	yes	no
Complex geometry module	yes	no
Import of CAD-file	yes	yes
Edit geometry in GUI	no	no
Delete objects in GUI	yes	no
Undo command	no	yes[3]
Save as command	yes	no
Must change/choose settings	no	yes
Automatically saves file in correct format	yes	no
File extension	.py	.geo
Programming language	python	Gmsh's own
Easy modification of script file from GUI	no	yes
User forum	yes	no[4]
Easy to use advanced meshing tool	yes	no
Tetrahedral mesh algorithm	Netgen	Gmsh

As shown in Table 2.2, both software have advantages and disadvantages, and there is no clear winner. However, after some evaluation, it was decided to use Salome in the tutorial. Salome has a more user-friendly GUI with little need for simultaneous editing of the script, because it has an object browser that lists all the created objects. Gmsh, on the other hand, does not have an object browser, so there is no possibility to show or delete objects in the GUI. As a consequence, simultaneous editing of the script is unavoidable.

[3] Modules → Geometry → Remove last script.

[4] It is possible to post issues to the developers at https://gitlab.onelab.info/Gmsh/Gmsh/-/issues.

Another advantage is that Salome uses the Python scripting language, which the students are already familiar with, while Gmsh uses its own. In addition, Salome has a more advanced geometry module that resembles a CAD software. The implementation of other geometries of more applied cases is considered to be easier in Salome. Moreover, Salome has multiple meshing algorithms and advanced meshing options.

However, for even more complex geometries, the Salome geometry module may not be suitable. In these cases, the geometry can be drawn in an open-source drawing software, such as FreeCAD or Blender, and exported to Salome.

2.3 Turbulence boundary layer theory

This section describes turbulence boundary layers, estimation of y^+ and wall treatment.

Turbulent flow within solid smooth walls behaves significantly different than free stream turbulent flow.

Equation (2.2) shows the *law of the wall*, which describes how the velocity of the flow $U\ [m/s]$ is dependent on the distance from the wall, $y\ [m]$, the *wall shear stress*, $\tau_w\ [Pa]$, the *dynamic viscosity*, $\mu\ [Ns/m^2]$ and the *density*, $\rho\ [kg/m^3]$. It also describes the relationship between the dimensionless groups y^+ and u^+.

$$u^+ = \frac{U}{u_\tau} = f\left(\frac{\rho u_\tau y}{\mu}\right) = f(y^+) \tag{2.2}$$

Where f symbolizes a function. The *friction velocity*, $u_\tau\ [m/s]$, is given by Equation (2.3):

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{2.3}$$

The law of the wall is illustrated in Figure 2.13 [17].

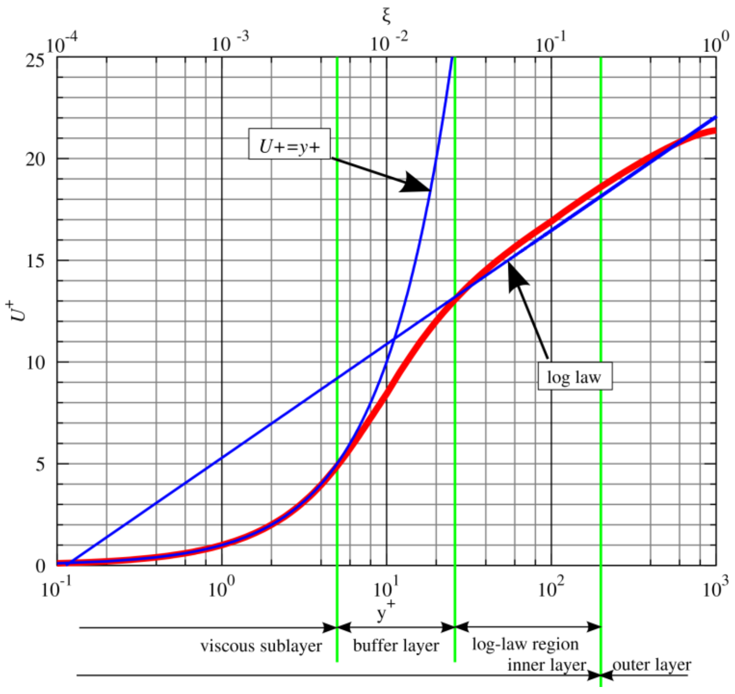


Figure 2.13: Law of the wall [17].

The fluid velocity is zero at the wall. Closest to the wall, there is a thin layer where the fluid flow is dominated by viscous effects. This region is called the *viscous sub-layer* or the *linear sub-layer* and in this layer the values of $y^+ < 5$. In this layer it is assumed that the wall shear stress and the *shear stress*, τ $[Pa]$ are constant and equal.

Equation (2.4) shows that the viscous sub-layer has a linear relationship between the distance from the wall and the velocity.

$$U = \frac{\tau_w y}{\mu} \tag{2.4}$$

Of this relationship follows Equation (2.5):

$$y^+ = u^+ \tag{2.5}$$

The *buffer layer* is the region outside the viscous sub-layer where $5 < y^+ < 30$. In this area, the viscous and turbulent stresses are considered to be equal.

The *log-law* layer is located outside the viscous sub-layer and the buffer layer. It is defined as the region where $30 < y^+ < 500$. In this region there is a logarithmic relation between u^+ and y^+. Both turbulent and viscous effects influence the flow, but the turbulent stresses are predominating.

In the outer layer, far from the wall, where $y^+ > 500$, inertia forces dominate the flow and the viscous forces are negligible. In this region the *velocity-defect law* or the *law of the wake*, Equation (2.6), is valid.

$$\frac{U_{max} - U}{u_\tau} = g\left(\frac{y}{\delta}\right) \tag{2.6}$$

Where δ $[m]$ is the boundary layer thickness, U_{max} $[m/s]$ is the maximum velocity at the pipe centerline and g symbolizes a function [5, pp. 57-59].

2.3.1 Estimation of y^+

y^+ is given by Equation (2.7) [5, p. 275]:

$$y^+ = \frac{\Delta y_P}{\nu} u_\tau = \frac{\Delta y_P}{\nu}\sqrt{\frac{\tau_w}{\rho}} \tag{2.7}$$

Where Δy_P $[m]$ is the distance from the nearest node P to the wall and ν $[m^2/s]$ is the *kinematic viscosity*.

If $y^+ \leq 11.63$ the flow near the wall is considered to be entirely dominated by viscous effects and is assumed to be laminar. If $y^+ \geq 11.63$ the flow is considered turbulent and wall functions (WF) must be used to model the flow. The intersection between the linear profile and the log law curve occurs at the value of $y^+ = 11.63$, as illustrated in Figure 2.13 [5, p. 275].

The wall shear stress, τ_w, is defined by Equation (2.8) [18, p. 362]:

$$\tau_w = \frac{C_f \rho U^2}{2} \tag{2.8}$$

Where C_f is the *skin friction coefficient*. Several different formulas for the approximation of the skin friction coefficient based on boundary layer theory for flat plates are available in the literature. White [19, p. 473] proposes Equation (2.9):

$$C_f = \frac{0.027}{Re^{\frac{1}{7}}} \tag{2.9}$$

Where Re $[-]$ is the *Reynolds number*.

Since the friction velocity is not known before running a simulation, Equation (2.7) only approximates the y^+. The actual y^+ must be determined after running a simulation [12, p. 855].

2.3.2 Wall treatment

Wall functions are used to model the flow in the near-wall area. The first wall function was formulated by Spalding in 1961 [20]. In the literature, wall functions are covered by among others Kalitzin et al. [21], Bredberg [22], Liu [23] and Liu [24].

The standard wall functions are based on the law of the wall and are valid in the log-law region. Thus, wall functions should be used when the node closest to the wall is placed in the log-law region, where $30 < y^+ < 500$. However, a value of $y^+ > 11.63$ might be acceptable [5, p. 283]. Some advantages of the wall function approach are that it allows for a coarse grid, which reduces the requirements of the computer and the computational cost, along with improved convergence and numerical stability.

The wall function approach is expected to give less accurate results, as it may not model the flow adjacent to the wall adequately. The wall function method requires a *high-Reynolds-number* (HRN) turbulence model.

To accurately resolve the flow in the boundary layer, the mesh near the wall must be very fine. The node closest to the wall must be placed in the viscous region where $y^+ < 5$. However, a value of $y^+ \leq 1$ is recommended. In the viscous layer, the log-law is not valid, so wall functions are inaccurate and should not be used. The appropriate turbulence model for this condition is a *low-Reynolds-number* (LRN) model. An LRN-model is a model that can be integrated towards the wall. The disadvantage with this approach is that the requirements of the computer and the computational cost increase with a finer mesh.

In the buffer layer, there are rapid fluctuations in the turbulent source terms, and it is challenging to predict and model the flow behavior. If the first node is located in the buffer layer, neither the wall function approach nor the full resolution approach is fitting. Therefore, it is recommended not to place the first node in the buffer layer. The general practice is to either position the first node in the viscous sub-layer and use an LRN-model, or place it in the log-law layer and use an HRN-model [5, pp. 85, 106, 275-276, 283], [22].

If $y^+ < 30$ and wall functions are used, the flow predicted by the wall functions may be inaccurate or erroneous [5, p. 279]. However, in complex flows and geometries, it may be challenging to fulfill the y^+ requirement at certain locations in the domain. As a consequence, the flow downstream of a y^+ violation can be affected by an inaccurate solution upstream[5, p. 292].

In the design phase of a mesh it is important to keep in mind the desired turbulence model, wall treatment and y^+. The equations in Chapter 2.3.1 can give an estimate of the y^+ and the required distance to the first computational interior node.

2.4 Turbulence modeling

This section describes turbulent flow characteristics, the turbulence models $k - \varepsilon$ and $k - \omega\ SST$, simulation time step and pressure loss.

2.4.1 Turbulent flow characteristics

The volume flow, $Q\ [m^3/s]$, is given by Equation (2.10):

$$Q = UA \tag{2.10}$$

Where $A\ [m^2]$ is the cross-sectional area of the pipe and is calculated as shown in Equation (2.11):

$$A = \frac{\pi D^2}{4} \tag{2.11}$$

The *continuity equation* for an incompressible fluid flowing through a pipe is given by Equation (2.12) [18, p. 109]:

$$Q = U_{in}A_{in} = U_{out}A_{out} \tag{2.12}$$

Where the subscripts in is the location of the inlet and out is the location of the outlet.

The critical Reynolds number for a circular tube is approximately 2300. The flow is laminar for Reynolds numbers below this value. For Reynolds numbers between 2300 and 4000 the flow is transitional. The Reynolds number must be 4000 to achieve fully developed turbulent flow [25].

The Reynolds number for a pipe is given by Equation (2.13):

$$Re = \frac{UD}{\nu} \tag{2.13}$$

Where D $[m]$ is the diameter.

The kinematic viscosity is calculated by dividing the dynamic viscosity with the fluid density, as displayed in Equation (2.14):

$$\nu = \frac{\mu}{\rho} \tag{2.14}$$

Fully developed turbulent flow occurs when the velocity profile does not change in the direction of the flow. The hydrodynamic entry length, $x_{fd,h}$ $[m]$, is defined as the length from the inlet until fully developed flow occurs. The hydrodynamic entry length is dependent on many factors and an approximation is shown in Equation (2.15) [26]:

$$10 \lesssim \frac{x_{fd,h}}{D} \gtrsim 60 \tag{2.15}$$

The maximum velocity occurs at the centerline of the pipe and is a function of the mean velocity. The maximum velocity for laminar flow, $U_{max,lam,}$ is given by Equation (2.16) [18, p. 264]:

$$U_{max,lam} = 2U \tag{2.16}$$

The maximum velocity for turbulent flow, $U_{max,turb,}$ is given by Equation (2.17) [18, pp. 277-278]:

$$U_{max,turb} = \left(1 + 1.44\sqrt{f_{turb}}\,\right)U \tag{2.17}$$

Where f_{turb} $[-]$ is the turbulent friction factor.

The turbulent velocity profile is predicted by Equation (2.18) [18, pp. 277-278]:

$$U = \left(1 + 1.44\sqrt{f_{turb}}\,\right)U - 2.15\sqrt{f_{turb}}U \cdot log_{10}\frac{r_0}{r_0 - r} \tag{2.18}$$

Where r_0 $[m]$ is the radius of the pipe and r $[m]$ is any radius.

2.4.2 $k - \varepsilon$ turbulence model

The $k - \varepsilon$ model is the most widespread and verified turbulence model and is extensively used in engineering applications in the industry [5, pp. 78, 97]. It is a *Reynolds-averaged Navier-Stokes (RANS)* turbulence model. The $k - \varepsilon$ model in OpenFOAM (kEpsilon) is based on the standard model by Launder and Spalding [27] and on the *Rapid Distortion Theory* (RDT) compression term by El Tahry [28]. It is a two-equation model, with one equation for k and one for ε. The standard $k - \varepsilon$ model is an HRN-model and needs wall functions. Some disadvantages with the model are that it predicts too high quantities of shear stresses, as well as turbulence in stagnation locations. Some advantages are that it is computationally efficient and gives good results for many industrial flow problems [1], [5, pp. 88, 97], [29].

In a CFD simulation inlet values of k and ε must be provided as boundary conditions. Measurements of these values for a given CFD case are frequently not available in the literature. However, formulas based on the *turbulence length scale*, l, and the *turbulence intensity*, T_i, can give an approximation [5, p. 76].

The turbulence length scale, l $[m]$, for a circular pipe is given by Equation (2.19):

$$l = 0.07D \tag{2.19}$$

The turbulence intensity, T_i $[-]$, is given by Equation (2.20) [30, Eq. (6.68)].

$$T_i = 0.16Re^{-\frac{1}{8}} \tag{2.20}$$

The *turbulence kinetic energy*, k $[m^2/s^2]$, and *turbulence dissipation rate*, ε $[m^2/s^3]$, are given by Equations (2.21) - (2.22):

$$k = \frac{3}{2}(T_i U)^2 \tag{2.21}$$

$$\varepsilon = \frac{C_\mu^{\frac{3}{4}} k^{\frac{3}{2}}}{l} \tag{2.22}$$

Where C_μ is a constant equal to 0.09 [5, pp. 76-77].

2.4.3 $k - \omega$ SST turbulence model

The $k - \omega$ turbulence model is a RANS model and is the most common alternative to the $k - \varepsilon$ model. The $k - \omega$ *Shear Stress Transport* (SST) turbulence model in OpenFOAM (kOmegaSST) is based on the model by Menter and Esch [31] and the updated model by Menter et al. [32]. It is a two-equation model, with one equation for k and one for ω. It is a hybrid model; in the near-wall region, the $k - \omega$ model is used and in the fully turbulent region $k - \varepsilon$ is used. Advantages of the hybrid model are that the $k - \omega$ model performs better than $k - \varepsilon$ in the near-wall area, while the $k - \varepsilon$ model is more robust in the fully turbulent region. As a result, the solution is numerically stable and robust. The model also has the capacity to model flow separation. The model can solve the flow in the near-wall area and can be used without wall functions in LRN applications. One disadvantage is that the model does not model the interactions between the mean flow and turbulent stresses accurately [5, pp. 90-92], [33].

The *turbulence dissipation rate*, ω $[s^{-1}]$ is given by Equation (2.23) [33].

$$\omega = \frac{\sqrt{k}}{C_\mu^{\frac{1}{4}} l} \tag{2.23}$$

The turbulence kinetic energy, $k,$ is calculated as for the $k - \varepsilon$ model, by Equation (2.21).

2.4.4 Time step

The appropriate time step of a simulation can be estimated with the *Courant number*. The courant number expresses how much information is transferred in one time-step through one cell. An appropriate courant number helps ensure numerical stability and convergence of the simulation. A courant number of 1 is often considered to give a good result. The definition of the Courant number in 1D is shown in Equation (2.24):

$$Courant\ number = \frac{U\Delta t}{\Delta x} \tag{2.24}$$

Where $\Delta t\ [s]$ is the time step and $\Delta x\ [m]$ is the length of a grid cell [12, pp. 616-618].

The courant number is of significant importance in transient simulations. However, in steady state simulations, the courant number should not notably affect the simulation, and the time step is considered to be an iteration counter [34].

2.4.5 Pressure loss

The s*tatic pressure*, $p\ [Pa]$, is defined as the fluid pressure in a moving fluid.

The pressure results in OpenFOAM for incompressible solvers like simpleFoam, are given as the *kinematic pressure*, $p_k\ [m^2/s^2]$. The static pressure is calculated by multiplying the kinematic pressure with the fluid density, as shown in Equation (2.25).

$$p = p_k \cdot \rho \tag{2.25}$$

Stagnation pressure, $p_0\ [Pa]$*,* is defined as the sum of the static pressure and the *dynamic pressure*, $\rho\frac{U^2}{2}\ [Pa]$, as shown in Equation (2.26), and is valid for incompressible fluids [18, p. 139].

$$p_0 = p + \rho\frac{U^2}{2} \tag{2.26}$$

The steady flow energy balance for an incompressible fluid along a streamline is given by Equation (2.27) [18, p. 145]:

$$\frac{1}{2}\rho U_{in}^2 + p_{in} + \rho g z_{in} - p_{loss} = \frac{1}{2}\rho U_{out}^2 + p_{out} + \rho g z_{out} \tag{2.27}$$

Where $p\ [Pa]$ is the gage pressure, $z\ [m]$ is the elevation, $g\ [m/s^2]$ is the gravitational acceleration and $p_{loss}\ [Pa]$ is the total pressure loss.

For a pipe with no change in elevation between inlet and outlet, $z_{in} = z_{out}$. When the pressure at the discharge is the same as that of the surrounding pressure, the gage pressure at the outlet is zero, $p_{out} = 0$. Then, Equation (2.27) reduces to Equation (2.28):

$$\frac{1}{2}\rho U_{in}^2 + p_{in} - p_{loss} = \frac{1}{2}\rho U_{out}^2 \tag{2.28}$$

The total pressure loss is the sum of the pressure loss due to friction, $p_{friction}$, and the pressure loss due to obstruction, $p_{obstruction}$, and is given by Equation (2.29):

$$p_{loss} = \sum p_{friction} + \sum p_{obstruction} \tag{2.29}$$

The pressure loss due to friction is given by Equation (2.30) [18, p. 261]:

$$p_{friction} = f\frac{L}{D}\frac{1}{2}\rho U^2 \tag{2.30}$$

Where $L\ [m]$ is the length of the pipe section.

For laminar flow in a pipe, the friction factor, $f_{lam}\ [-]$, is given by Equation (2.31) [18, p. 265]:

$$f_{lam} = \frac{64}{Re} \tag{2.31}$$

For turbulent flow in a pipe, the friction factor, f_{turb} $[-]$, can be approximated by the *Swamee-Jain equation,* Equation (2.32) [35, Eq. 19], which is based on the *Darcy-Weisbach equation* and the *Colebrook-White equation*.

$$f_{turb} = \frac{0.25}{\left[\log_{10}\left(\frac{1}{3.7\left(\frac{D}{\epsilon}\right)} + \frac{5.74}{Re^{0.9}}\right)\right]^2} \tag{2.32}$$

Where ϵ $[m]$ is the *roughness height of the pipe wall*. Equation (2.32) is valid for $5000 < Re < 10^8$ and $10^{-6} < \frac{\epsilon}{D} < 10^{-2}$.

For smooth walls, the roughness height of the pipe wall is zero [18, p. 284]. The friction factor for smooth walls can be approximated by Equation (2.33) by Blasius [36]:

$$f_{turb} = \frac{0.316}{Re^{0.25}} \tag{2.33}$$

Equation (2.33) is valid for $3000 < Re < 10^5$.

The pressure loss due to obstruction is given by Equation (2.34) [37, p. 69]:

$$p_{obstruction} = K \cdot \frac{\rho}{2} U^2 \tag{2.34}$$

Where K $[-]$ is the obstruction loss factor.

A table with obstruction loss factors for different obstructions is available in [37, Tab. 2.1]. Some relevant obstruction loss factors are listed in Table 2.3.

Table 2.3: Obstruction loss factors [37, Tab. 2.1].

Obstruction		**K**
Tank exit		0.5
Tank entry		1.0
Conical enlargement: (total included angle)	6°	0.13
	10°	0.16
	15°	0.30
	25°	0.55
Sudden contractions: Area ratio (A2/A1)	0.2	0.41
	0.4	0.30
	0.6	0.18
	0.8	0.06

3 Case

In this section the flow geometry, meshes, simulation cases, numerical setup and student tutorial are presented. The software used in the cases is listed in Appendix B.

3.1 Geometry

A relatively complex flow geometry is chosen for the case. The geometry is a 3D model of a diverging converging pipe with belly section. Figure 3.1 shows the geometry with measurements. The inlet section has a diameter of 20 mm, the diameter of the belly section is 60 mm, and the diameter of the outlet section is 10 mm. The length of the inlet section is 300 mm ($15D$), the diverging, belly and converging sections have a length of 50 mm each, and the outlet section is 300 mm ($30D$) long.

A technical drawing of the pipe is available in Appendix C.

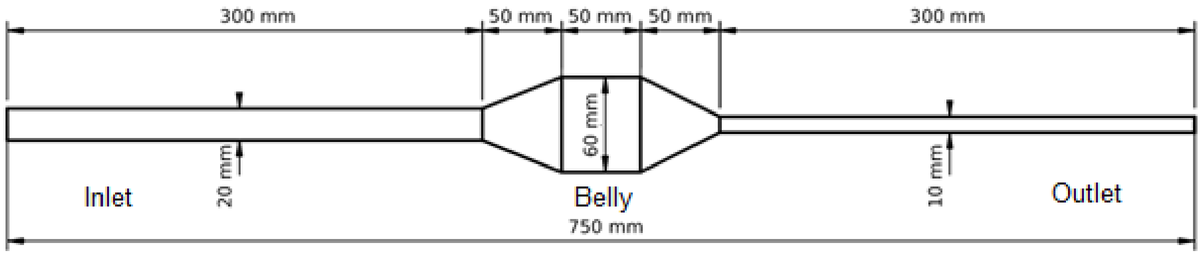


Figure 3.1: Geometry.

The geometry is modeled in the geometry module in Salome. Based on the theory presented in Section 2.1.3, it is decided to create an OH-grid, and the interior geometry is designed to enable the meshing of this grid. The pipe consists of a divided disk with five quadrangles extruded along the pipe length, as shown in Figure 3.2. The quadrilateral design is a template for the hexahedral OH-grid. When the pipe is divided into quadrangles, the quadrangle mapping algorithm in Salome can create a fully hexahedral mesh without highly skewed cells.

Another option for creating a block-structured mesh is to use the *HexaBlock* module in Salome. This module creates the geometry in the form of blocks, which makes a basis for hexahedral mesh construction in the mesh module [38].

The method of creating the geometry is described in the student tutorial in Appendix N.

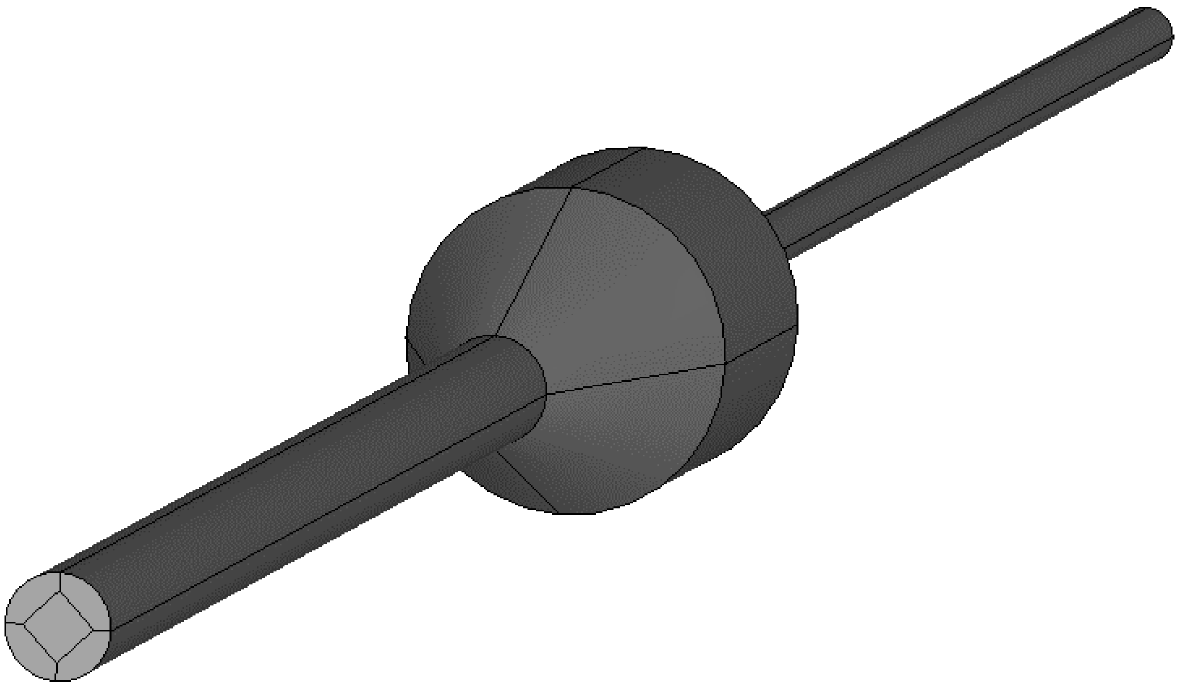


Figure 3.2: Sub-regions for the OH-grid.

3.2 Meshes

The meshes are created in the mesh module in Salome. The mesh type is *hexahedral*, created with the *3D Hexahedron, 2D Quadrangle Mapping* and *1D Wire Discretization* algorithms. The 1D algorithm discretizes lines in the domain, the 2D algorithm creates 2D surfaces and domains, and the 3D algorithm creates volume meshes from the 2D meshes.

Furthermore, sub-meshes in the axial direction are created to ensure the desired distribution and number of cells. For some of the meshes, sub-meshes with grading in the radial direction are created on the outer quadrangles to control the distance from the wall to the first interior node. In addition, the mesh groups *inlet, outlet* and *wall* are created for the simulation in OpenFOAM.

The result is an OH-grid with quadrangle surfaces and hexahedron volumes, as shown in Figure 3.3. The OH-grid is extruded along the pipe length; the grid expands radially outwards in the diverging section of the pipe and contracts radially inwards in the converging section of the pipe.

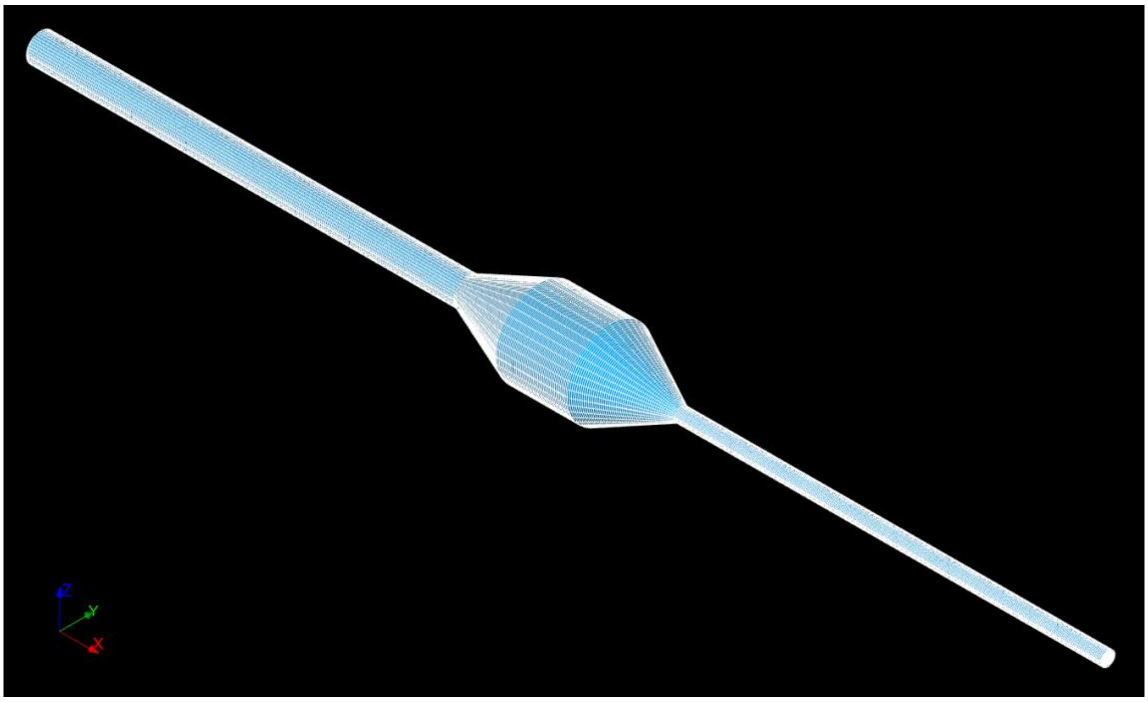


Figure 3.3: Mesh.

3 different meshes are created for the pipe geometry to account for different wall treatments. The meshes are built with the same pattern in the axial direction and in the center quadrangle, but differ in the four quadrangles towards the wall. The python script for the meshes is available in Appendix D.

Descriptions of the meshes are given in Table 3.1.

Mesh A (Figure 3.4) is the original mesh and was created without considering the value of y^+, only the general mesh quality. Mesh B (Figure 3.5) is created to be used with wall functions and aims for a y^+ of 30 at the inlet. Mesh C (Figure 3.6) is created to be used with the full resolution approach and aims for a y^+ less than 1. Mesh B and Mesh C are graded towards the wall.

The equations presented in Chapter 2.3.1 are used for the calculations of y^+.

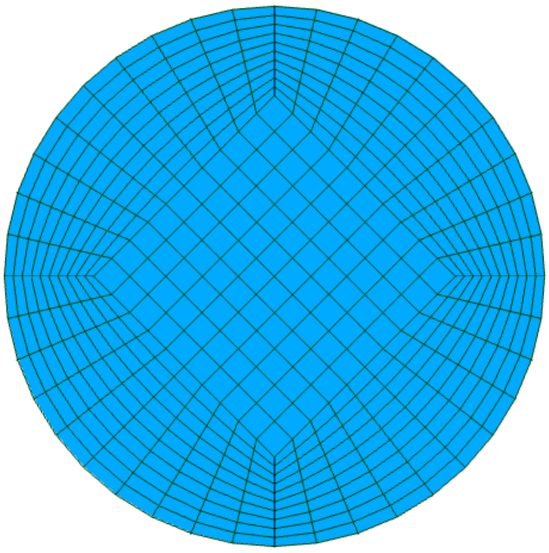


Figure 3.4: Mesh A.

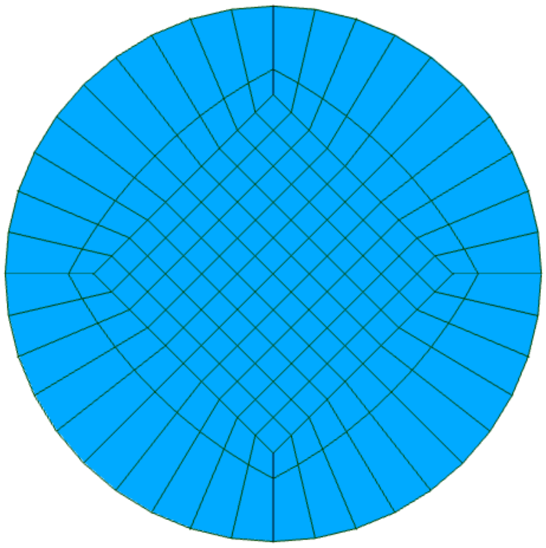


Figure 3.5: Mesh B.

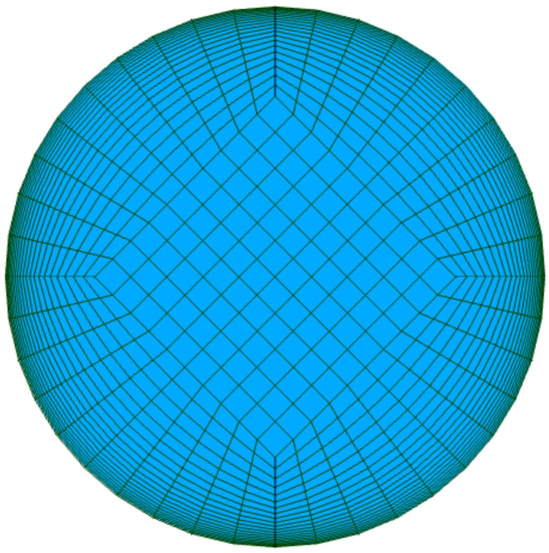


Figure 3.6: Mesh C.

Table 3.1 shows that the non-orthogonality of the three meshes is quite similar. The maximum non-orthogonality appears in Mesh A. Likewise; the maximum skewness also appears in Mesh A. However, the maximum aspect ratio is significantly larger in Mesh C than in the other meshes because of the thin layers of cells near the wall. Mesh B has the lowest non-orthogonality, skewness and aspect ratio.

Table 3.1: Mesh cases.

Mesh	**A**	**B**	**C**
Δy_P $[mm]$	0.33	2.36	0.041
Calculated y^+ at inlet	4.24	30.33	0.53
Appropriate turbulence model	LRN[5]	HRN	LRN
Mesh description	original	$y^+ \approx 30$	$y^+ \leq 1$
Number of cells	375 000	135 000	675 000
Max aspect ratio	7.04	6.63	44.41
Mesh non-orthogonality:			
Max	35.00	28.36	33.92
Average	8.47	7.31	7.75
Max skewness	2.04	1.26	1.62
Desired turbulence model	kEpsilon	kEpsilon	kOmegaSST
Wall treatment	Wall functions	Wall functions	Full resolution

Visualizations of the aspect ratios and skewness of the circular cross-sections are available in Appendix F. Mesh A has the highest aspect ratio just outside the corners of the middle quadrangle. The highest aspect ratios of Mesh B occur in the layer of cells closest to the wall and from the outer corners of the middle quadrangle towards the wall. Mesh C has the highest aspect ratios in the thin layers of cells closest to the wall, although the figure does not display the colors visibly. The maximum skewness in Mesh A and Mesh C occur in the cells in the regions outside the corners of the middle quadrangle. The cells of Mesh B have the largest skewness in the corner areas in the layer outside the middle quadrangle.

The procedure to create Mesh B is described in the student tutorial in Appendix N. The procedure to create Mesh A and Mesh C is explained in Appendix M.

The method to import a mesh created in Salome to OpenFOAM is described in Appendix E.

[5] $y^+ < 5$ places the first node in the viscous region, but for LRN models the value should ideally be $y^+ \leq 1$.

The *checkMesh* command did not detect any errors in any of the meshes.

3.3 Simulation cases

The case is a complex internal pipe flow problem. Air enters the pipe inlet at a velocity of 3 m/s at a Reynolds number of 4000, thus fully turbulent flow. The physics involved in the model is quite complex, because turbulent, transition and laminar flow occur. When the cross-section of the pipe expands to the belly section, the flow transitions to laminar flow. When the pipe contracts to the narrower outlet section, the flow transitions back to turbulent. The turbulent air flow continues through the outlet section at and leaves the pipe at the outlet.

The flow medium is air at $20\ ℃$. The flow is 3D, isothermal and without gravity forces. The flow is modeled with the RANS turbulence models *standard* $k-\varepsilon$ and $k-\omega\ SST$. The s*impleFoam* solver in OpenFOAM is used to calculate the pipe flow. simpleFoam is a solver for steady-state turbulent incompressible flow, which uses the *semi-implicit method for pressure-linked equations (SIMPLE)* algorithm or the *semi-implicit method for pressure-linked equations-consistent (SIMPLEC)* for the solution of the momentum and continuity equations.

To ensure the user-friendliness of the student tutorial, the case is based on a tutorial available in OpenFOAM[6], *pitzDaily* [39]. The contents of the pitzDaily directory can be copied to a new working directory and only small changes need to be made.

The initial and boundary conditions are adjusted to fit the current case and the boundaries and patches are changed to correspond to the imported mesh. Furthermore, the turbulent properties $k,\ \varepsilon$ and ω are calculated with the equations given in Section 2.4.2 and 2.4.3 and the kinematic viscosity of air is adjusted. Constants and boundary conditions are listed in Appendix H.

Table 3.2 summarizes the numerical setup of the 4 simulation cases.

[6] Path: /opt/openfoam8/tutorials/incompressible/simpleFoam/pitzDaily

Table 3.2: Numerical setup for the simulation cases.

Case	**A**	**B1**	**B2**	**C**
Mesh	A	B	B	C
Turbulence model	kEpsilon with default WF	kEpsilon with default WF	kEpsilon with adjusted WF	kOmegaSST
Solution algorithm	SIMPLE	SIMPLEC	SIMPLEC	SIMPLEC
Number of iterations	688	476	390	1472
Simulation time[7] [min]	11	3	3	129
residualControl:				
p	1E-05	1E-03	1E-03	1E-05
U	1E-05	1E-03	1E-03	1E-05
k|epsilon|omega	1E-05	1E-04	1E-04	1E-05
relaxationFactors:				
p	0.6	0.8	0.7	0.7
U	0.7	0.8	0.7	0.5
k|epsilon|omega	0.7	0.8	0.7	0.7

The relaxation factors, tolerances and algorithms are chosen based on simulation tests and residual plots to assess the best factors. The values of the solution tolerances and residual controls are decreased from default to increase the accuracy of the solution. The relaxation factors are decreased to ensure numerical stability and convergence. SIMPLEC showed better stability for Cases B1-C. The numerical schemes remain default.

The time step is decreased to 0.0001. The timestep is calculated by setting the Courant number to 1, velocity to 10 m/s and Δx to 1 mm. This gives an approximate value for the whole flow domain, but is not considered to notably influence the simulation in steady state.

[7] Simulation time is not fixed, but depends on available computer power.

Case A is the original case and was made without considering y^+. Case B1 and Case B2 aim for a y^+ of 30, and an HRN-turbulence model, the $k - \varepsilon$ with wall functions. Case C aims for a y^+ less than 1, and an LRN-turbulence model, the $k - \omega \; SST$.

An overview of different wall functions in OpenFOAM and their properties is given in Appendix G.

Table 3.3 shows the wall functions used in the $k - \varepsilon$ models. Case A and Case B1 use the default wall functions in the pitzDaily tutorial. The ε wall function is valid in both the viscous and log-law layers. The wall functions for k and the *turbulent viscosity*, $\mu_t \; [m^2/s\,]$, (nut) are only valid in the log-law layer.

Case B2 utilizes other wall functions for k and μ_t , specifically the *kLowReWallFunction* and the *nutUSpaldingWallFunction.* The kLowReWallFunction is chosen because it is valid in both the viscous and log-law layers. The nutUSpaldingWallFunction is chosen because it is valid in all layers. The intention is that these wall functions can give better accuracy in areas where $y^+ < 30$.

Table 3.3: Wall functions $k - \varepsilon$ models.

Case	**k**	**epsilon**	**nut**
A	kqRWallFunction	epsilonWallFunction	nutkWallFunction
B1	kqRWallFunction	epsilonWallFunction	nutkWallFunction
B2	kLowReWallFunction	epsilonWallFunction	nutUSpaldingWallFunction

Table 3.4 shows the wall treatment used in the $k - \omega \; SST$ model. The wall treatment is chosen based on the recommended wall treatment for resolved boundary layer models in [12, p. 853] and [40].

Table 3.4: Wall treatment $k - \omega \; SST$ model.

Case	**k**	**omega**	**nut**
C	0	omegaWallFunction	nutLowReWallFunction

3.4 Student tutorial

A focus of the student tutorial has been to describe a user-friendly, accessible and general approach to creating and meshing a geometry.

Case B1 is the case chosen for the student tutorial based on the following points:

- Differs the least from the pitzDaily tutorial.
- $k - \varepsilon$ is the most validated turbulence model.
- Implements wall functions → short simulation time.
- Demonstrates how to adjust the mesh in the boundary layer region.
- Shows consistent results.

The student tutorial is available in Appendix N.

4 Results and discussion

This section presents the results. Basic fluid mechanics checkpoints, symmetry and residuals are evaluated, and the simulated pressure loss and velocity field are compared to the analytical solution. Uncertainties and weaknesses of the models are discussed.

4.1 Evaluation of results

In a given CFD simulation there is a risk of errors and uncertainties. Possible errors include numerical errors caused by errors in roundoff, iterative convergence or discretization. Additionally, errors can be of the type coding errors and user errors. Uncertainties can be caused by wrong assumptions, approximations, simplifications and boundary conditions, or uncertainties related to the physical model [5, pp. 286, 292]. Consequently, it is important to evaluate the results of a simulation.

Versteeg and Malalasekera [5, pp. 301-302] list some checkpoints for evaluating the results of a simulation which include:

1. *Fluid flows from high to low pressure (in pressure-driven flows)*
2. *Static pressure decreases when velocity increases (Bernoulli's theorem for inviscid flows)*
3. *Friction losses cause a decrease of total pressure in the direction of flow (viscous flow) [...]*
4. *The speed of a fluid near a stationary wall is smaller than the speed further away from the wall (boundary layer formation)*
5. *Flow adopts a fully developed state after a sufficiently long distance in a straight duct with constant cross-section [...]*
6. *A flow emerging into a large expanse of fluid from a small hole generally forms a jet [...]*

An evaluation of each checkpoint is given below:

1. **Yes**. For all the simulation cases, the outlet static pressure is 0, as seen in the cut plane visualizations in Appendix I.
2. **Yes**. If one compares the velocity and static pressure plots in Appendix I, one can see that they show opposite behavior. The inlet and mid sections show high static pressure and low velocity, while the outlet section show low static pressure and high velocity.
3. **Yes**. The total pressure is decreasing in the direction of the flow, as seen in the total pressure visualizations in Appendix I.
4. **Yes**. The velocity near the wall is small in all simulation cases, as seen in velocity distribution visualizations in Appendix I.
5. **Yes**. As seen in Figure 4.4, all the simulations show signs of fully developed flow at the end of the outlet section which is $30D$ long.

6. **Yes**. The velocity distributions shown in Appendix I show that in all the cases, the flow emerging from the smaller inlet section to the wider belly section forms a jet.

The simulation results in this report are considered to satisfy all of the listed checkpoints.

4.1.1 Symmetry

In all the cases, the geometry, inlet boundary conditions and flow are symmetrical about the centerline of the pipe. Thus, in the steady-state model, the simulation results are expected to be symmetrical. Consequently, unsymmetrical results are a clear indication of numerical errors.

The solution of the velocity, stagnation pressure (*total pressure* in OpenFOAM) and static pressure fields for all the cases are visualized in ParaView with a 2D slice on the Z normal. The visualizations are shown in Appendix I. The results appear to be symmetrical and indicate that the simulation cases show symmetrical solutions about the centerline of the pipe.

4.1.2 Residuals

Residuals can be monitored live during the simulation as described in [41, p. 202] or they can be created after the simulation as described in [42] and demonstrated in the student tutorial in Appendix N. For both methods, a *log* file with the terminal output must be created during the simulations to enable the creation of plots.

The generated residual plots for the simulation cases are shown in Appendix J. All the plots show relatively stable residuals which decrease or stabilize along a straight line, showing signs of monotonic convergence. Ergo, steady behavior is indicated, even though some of the quantities show small oscillations.

In Case B1, B2 and C the pressure is generally the most oscillating quantity. Case A gives the most stable plot for all the quantities. The fact that Case A shows the most stable residuals and small solution tolerances demonstrates that a good residual plot does not guarantee a consistent solution. Nor is a convergent solution necessarily an accurate solution. An unstable and fluctuating residual plot may indicate that the solution is amiss, but a stable plot does not subsequently indicate the opposite.

4.2 Boundary layer

The y^+ values of a simulation are a direct measure of **mesh optimality**. In OpenFOAM, y^+ is computed with the post-processing utility. The y^+ field is calculated by typing the following command in the terminal:

```
simpleFoam -postProcess -func yPlus
```

The computed minimum, maximum and average values of y^+ are displayed in the terminal, and the entire y^+ field can be visualized in ParaView.

Table 4.1 shows the computed y^+ values from the simulations.

Visualizations of the y^+ distributions of the cases are available in Appendix K and they show that the y^+distributions of the respective cases do not follow the same pattern. Case A and Case B1 have the lowest values along the inlet and diverging sections. Case B2 has the lowest values along the mid section (the diverging, belly and converging sections). Case C has the lowest values of y^+ along the diverging section, at the intersection of the belly and converging section, and at the middle part of the converging section.

Case C has a maximum y^+ value of 1.88. The average value is 0.55 and the y^+ visualization shows that y^+ is mostly lower than 1. Regardless, a y^+ of 1.88 places the first node in the viscous sub-layer. Thus, Mesh C is considered to be successful in the y^+ distribution for this flow case and gives good **mesh optimality**.

Table 4.1: Computed y^+ values.

Case	**A**	**B1**	**B2**	**C**
Turbulence model	kEpsilon (default WF)	kEpsilon (default WF)	kEpsilon (adjusted WF)	kOmegaSST (full resolution)
Required y^+	$y^+ > 30$	$y^+ > 30$	$y^+ > 30$	$y^+ \leq 1$
Calculated y^+	4.24	30.33	30.33	0.53
Simulation y^+:				
average	3.7	29	27	0.55
max	11	58	50	1.88
min	0.7	8.6	1.6	0.05
Section where low y^+ occur:				
inlet	X	X		
div.	X	X	X	X
belly			X	
conv.			X	X
outlet				

When Mesh A was originally constructed, y^+ was not taken into consideration. However, it was already decided to use the standard $k - \varepsilon$ model. The estimated and computed average y^+ for Case A of less than 5, places the first interior node in the viscous sub-layer. Consequently, the utilization of wall functions and the standard $k - \varepsilon$ model, an HRN-model, are assumed to give erroneous results.

Case B1 has an average value of y^+ of 29. However, the minimum value of 8.6 places the first computational node in the buffer layer, where the log-law is not valid. The visualization of the y^+ distribution shows that the lowest values occur in the region closest to the inlet. Along the inlet and diverging section the values of y^+ are mostly lower than 30.

Case B2 has an average y^+ value of 27. However, the minimum value of 1.6 places the first near-wall node in the viscous layer. The minimum y^+ value of Case B2 is considerably smaller than for Case B1. Along the mid section, the lowest y^+ values occur and are mostly below 20. The y^+ distribution places the first interior nodes in all the sub-layers: viscous, buffer, log-law.

The intersection between the linear profile and the log law curve occurs at the value of $y^+ = 11.63$. According to Versteeg and Malalasekera [5, p. 275], one can consider the flow as turbulent and use wall functions when $y^+ \geq 11.63$. However, wall functions should preferably be used when $y^+ > 30$ [5, p. 283].

The y^+ and wall function dependency does not seem to be covered in the OpenFOAM documentation. In his master's thesis, Furbo [43] investigates the $k - \varepsilon$ model both with and without wall functions for HRN and LRN applications for many different meshes with y^+ ranging from around 1 to above 30. The simulations gave varying results. He also points out that the pitzDaily tutorial in OpenFOAM uses a mesh with y^+ in the range 0.7 to 26, while using wall functions, thus not fulfilling the wall function requirement.

Since cases B1 and B2 have a large part of the y^+ values larger than 11.63, the wall functions may give acceptable results. Case A, on the other hand, has a maximum y^+ value of 11, therefore unsatisfactory results can be expected from the wall function approach.

The y^+ computations confirm that the y^+ estimates must be checked after a simulation. Moreover, y^+ estimations limited to only one geometrical flow domain (inlet section in this case) can fail to predict the entire y^+ distribution.

4.2.1 Increasing y^+

Some mesh and simulation tests were carried out to investigate if it was possible to increase the y^+ values in Case B1. Reducing the number of mesh cells in the outer quadrangles to only 1 in each quadrangle gives a Δy_P of 3.3 mm and is demonstrated in Figure 4.1. Nonetheless, the visualization of the y^+ distribution showed that the y^+ values along the inlet section do not reach 30.

Consequently, Δy_P must be even longer to increase the y^+ to 30. However, it is not possible to increase the Δy_P more, because the pipe is divided into 5 quadrangles by the divided disk utility in Salome. An alternative is to create the quadrangles manually.

Therefore, no further attempts to increase Δy_P were made. For this specific geometry and flow problem, increasing the y^+ even more is considered to give a too coarse and bad-quality mesh. It would be expected to merely capture basic characteristics of the flow and give low accuracy of the solution.

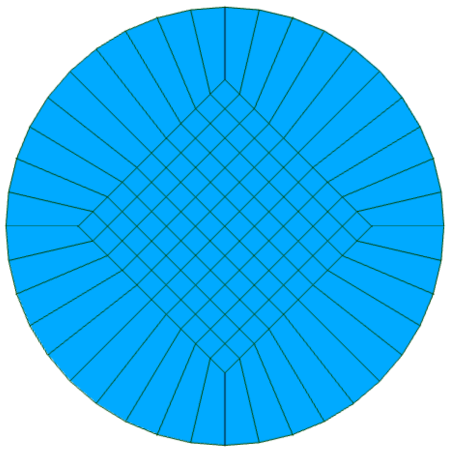


Figure 4.1: Coarser mesh, $\Delta y_P = 3.3\ mm$.

4.3 Pressure loss

The best way to test a CFD computation is to compare the computed results with experimental results. The model is recognized as validated if the difference in results is satisfactorily small [5]. However, for this specific work there is no experimental data available, therefore the solution is compared with the analytical solution.

The analytical solution is calculated with the equations and coefficients given in Section 2.4. The pressure friction losses are calculated for turbulent flow in the inlet and outlet sections, and laminar flow for the belly section. In the diverging and converging sections, the appropriate friction factor is chosen based on the calculated Reynolds number at each point.

The obstruction pressure losses are calculated for the diverging and converging sections of the pipe. The angle of the divergent section is determined to be 22°. The corresponding value of the obstruction loss factor for *conical enlargement* is calculated with linear interpolation.

The converging section is regarded as a *sudden contraction.* However, the area ratio of 0.03 is considerably smaller than the ratios listed in [37, Tab. 2.1], as shown in Table 2.3, so interpolation is not considered to give a credible result. As a result, the value of the obstruction factor is instead approximated to be that of a *tank exit.*

However, the table values for the obstruction loss factor do not take into account special flow phenomena, but consider a uniform and homogenous inflow into a given geometrical domain. The velocity field in the converging section is not uniform, but resembles the shape of a jet; it has a higher velocity in the area along the centerline than in the area closer to the wall. In addition, the pressure field is not homogeneous and the flow is transitioning from laminar to turbulent flow, which further complicate the flow field. These flow phenomena are assumed to increase the influence of *vena contracta*[8] and the pressure drop. Due to these factors, an additional 0.5 is added to the obstruction loss factor. An increased obstruction loss factor also agrees better with the simulation data.

For the friction loss calculations, the correct value of the wall roughness height must be determined. However, roughness information is not found in the OpenFOAM documentation. Lipej et al. [44] claim that most CFD analyses are based on hydraulically smooth walls. Furthermore, in the analyses in [45], [46] and [47] it is assumed that the default wall roughness in OpenFOAM is smooth. Moreover, on the SimScale platform which uses OpenFOAM, and in ANSYS, the default wall roughness is smooth [44], [48]. In addition, there are specific wall functions available in OpenFOAM which account for wall roughness,

[8] More information about vena contracta is available in [18, pp. 506-507].

namely the *nutkRoughWallFunction* and the *nutURoughWallFunction*. Thus, it is assumed that the default roughness in OpenFOAM is smooth.

The simulated cross-sectional average stagnation pressure loss for the simulation cases, along with the analytical solution is shown in Figure 4.2.

The method to compute the average stagnation pressure in OpenFOAM is described in Appendix L.

The values and results of the calculations are shown in Appendix L.

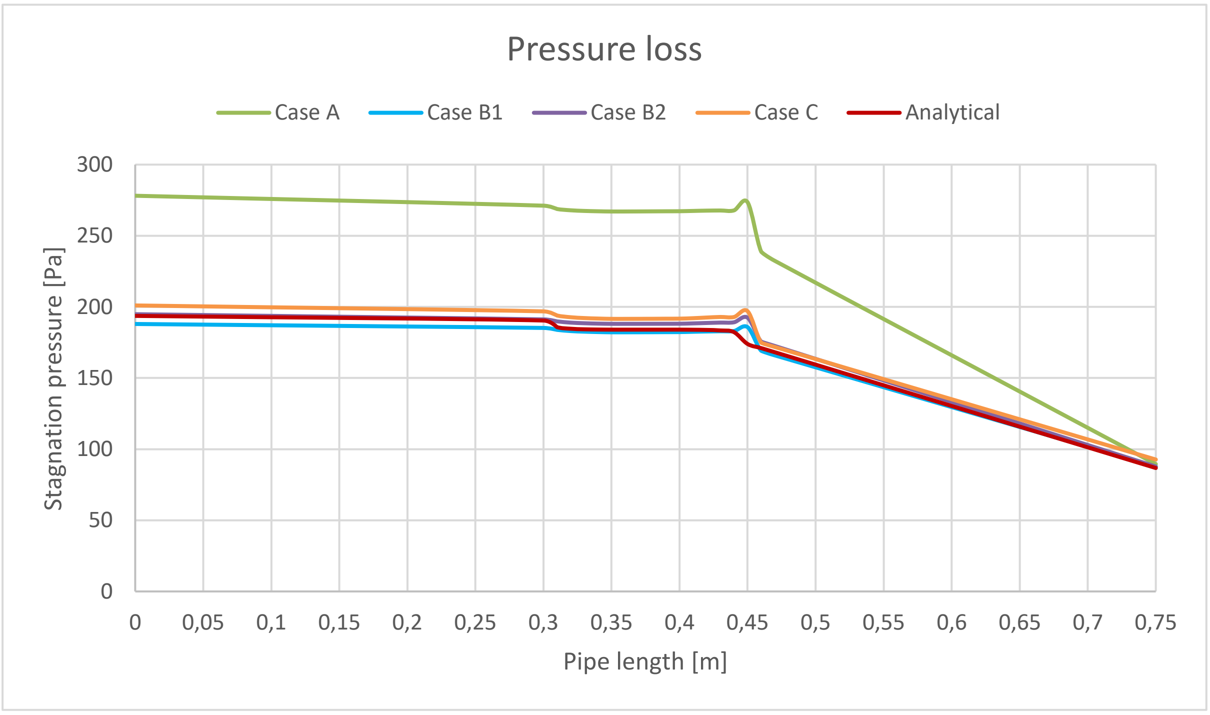


Figure 4.2: Stagnation pressure loss.

The results displayed in Figure 4.2 show that the analytical solution agrees quite well with Cases B1, B2 and C. Case A, on the other hand, show results diverging from the analytical solution; the pressure loss is larger, especially along the outlet section.

Since Case A does not satisfy the required y^+ range for standard wall functions, inaccurate results can be expected. The deviations in Case A support the theory concerning turbulence boundary layers and wall functions.

Even though Case B1 and Case B2 have a considerable amount of $y^+ < 30$, their results are consistent with the analytical solution. This can be explained by the fact that most of the $y^+ > 11.63$. The adjusted wall functions in Case B2 create a small change in pressure from Case B1, and the overall pressure loss in Case B2 agrees best with the analytical solution of all the cases.

It should also be noted that Cases B1 and B2 which employ the HRN $k - \varepsilon$ models with wall functions have a grid with considerably less number of cells and shorter simulation time than Case C with the full resolution approach. Case C runs for more than 2 hours, while Case B1 and Case B2 converge in 3 minutes. Yet, the results are very similar. If anything, Cases B1 and B2 show slightly better agreement with the analytical result in terms of average pressure loss.

The pressure results support the claim that wall functions are faster and more economical and can give good approximations of the flow. Although, in this particular case, a simulation time of 2 hours for the LRN model was not problematic. But in other flow applications, the simulation time may play a bigger role, especially in simulation cases that take days or weeks.

4.4 Velocity

Figure 4.3 shows the simulated cross-sectional average velocity compared with the calculated velocity. The calculated velocity is calculated with the continuity equation for each point.

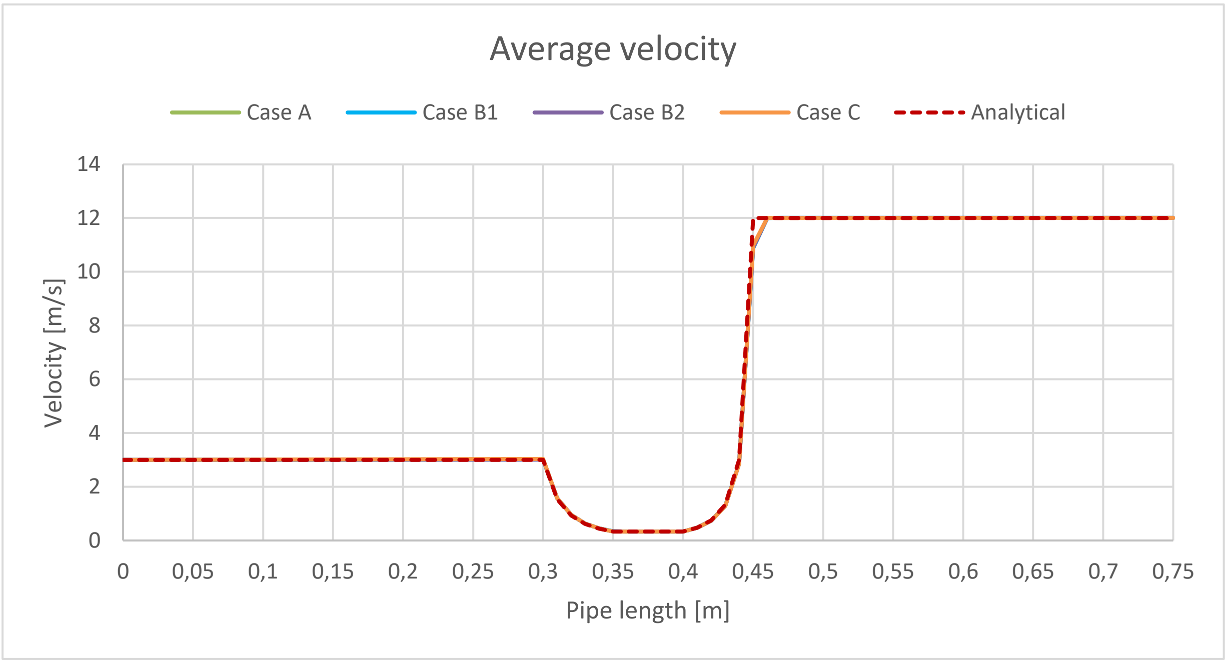


Figure 4.3: Average velocity along pipe length.

The velocities overlap, and it is not possible to observe deviations on the plot, except for in the region 45-47 mm from the inlet. The analytical solution gives a velocity of 12 m/s when the pipe diameter is 10 mm at 45 mm from the inlet, thus upholding the continuity equation. The simulated velocities, on the other hand, can be claimed to violate the continuity equation in this area because the velocity is lower than predicted. Energy loss cannot be the reason, because the velocities increase to 12 m/s at 47 mm from the inlet, even though the diameter has a constant cross-section from 45 mm.

Another interesting result is that Case A, which did not give consistent stagnation pressure results, matched the other three simulations in terms of average velocity. A reason for this may be that the computed velocities are more accurate than the computed pressures in Case A.

Figure 4.4 shows the centerline velocity compared with the calculated centerline velocity. The analytical maximum velocities are calculated with the equations in section 2.4.1. Laminar flow is assumed for Reynolds numbers below 4000. For the calculation of the turbulent centerline velocity, fully developed flow is assumed to be reached at $10D$ for the inlet section and $15D$ for the outlet section, and the centerline velocity 45 mm from the inlet is assumed to be 13.5 m/s. These approximations make deviations from the calculated centerline velocity expected. In addition, the standard laminar flow predictions do not take into account a flow regime with a jet along the axis of the pipe.

Fully developed turbulent flow is achieved when the velocity along the centerline does not change. Along the inlet pipe section, which is $15D$ long, simulation data shows that fully developed flow is not achieved for all the simulations. However, along the outlet pipe section, which is $30D$ long, simulation data shows that fully developed flow is achieved at $29D$ (74 mm from the inlet) for all cases, except Case C. However, the behavior of Case C indicates that fully developed flow would be achieved downstream if the outlet section had been slightly longer.

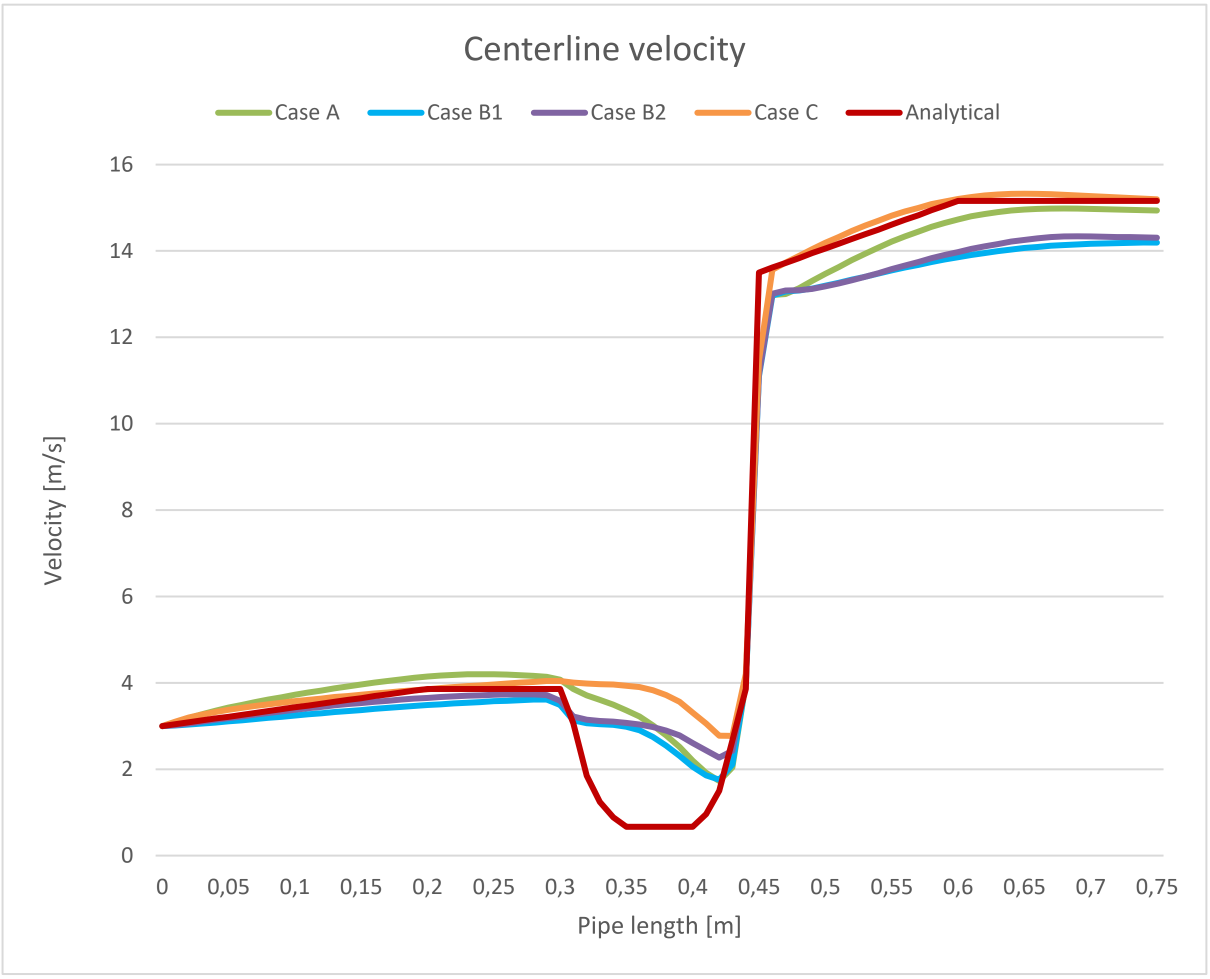


Figure 4.4: Centerline velocity along pipe length.

Figure 4.4 shows that Case C generally agrees best with the analytical solution in the pipe sections with turbulent flow (inlet and outlet). Cases B1 and B2 display larger deviations from the analytical solution, especially along the outlet section. They generally show similar behaviour, although Case B2 agrees slightly better with the analytical solution. Case A is quite consistent with the calculated velocity at the outlet. It should be noted that the analytical solution in the laminar flow regime is considered to give poor results because of the actual formation of a jet along the middle section of the pipe.

Figure 4.5 shows the turbulent velocity profile and analytical solution 74 mm from the inlet. The analytical solution fails to model the velocity at the pipe center and predicts a sharp point at the center of the pipe, although the profile should be rounded [18, p. 276].

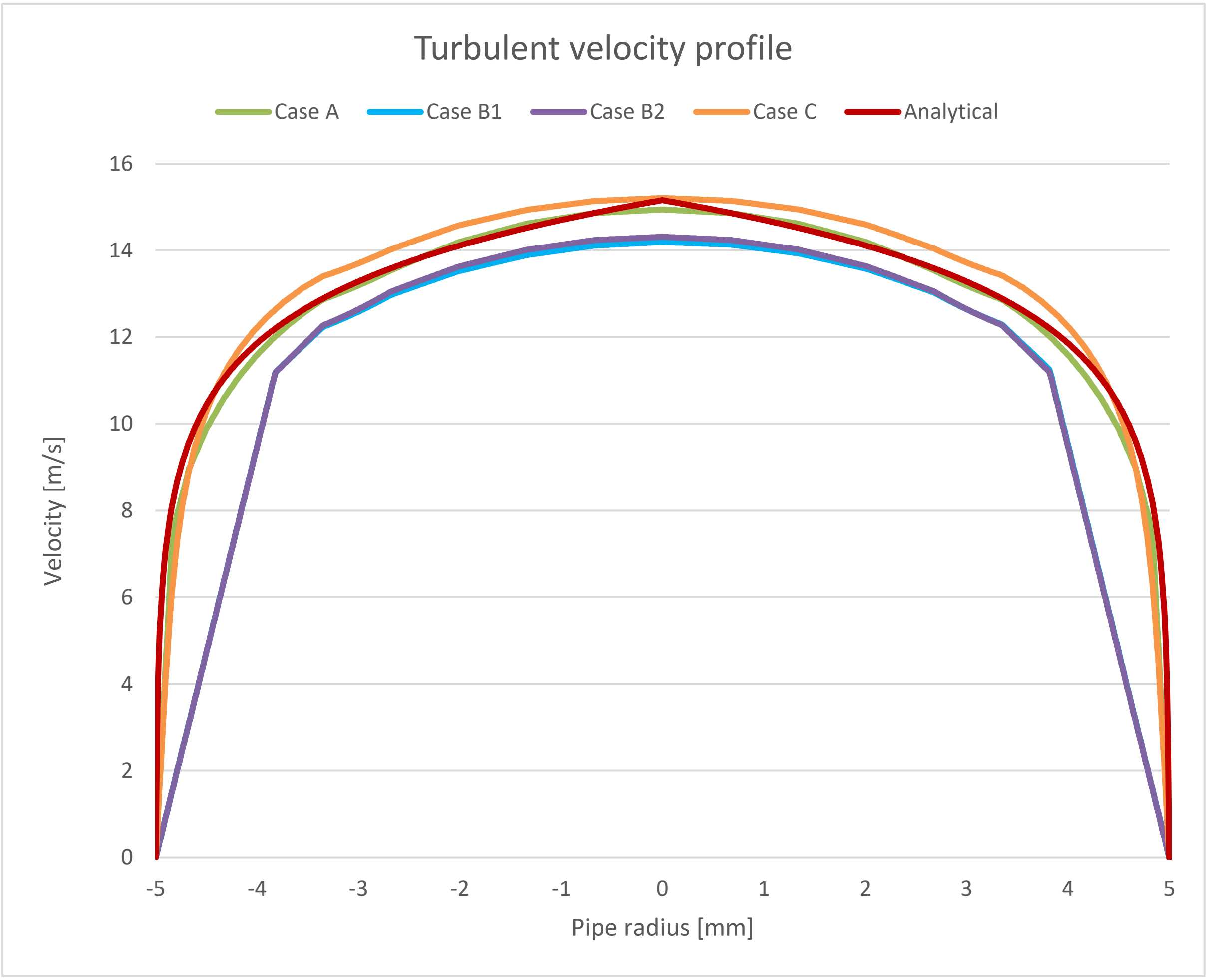


Figure 4.5: Velocity profile 74 mm from the inlet.

Case A and Case C show results consistent with the analytical solution. Case A predicts the velocity in the area close to the axis best, while Case C predicts the maximum velocity best. Case B1 and Case B2 diverge from the analytical solution in the area near the wall, which can be explained by the wall functions. Case B2 shows slightly better agreement with the analytical solution than Case B1.

Comparison of velocities with the analytical solutions show that Case C generally predicts the velocity field best. Case C is most successful in predicting both the maximum velocity at the centerline and the turbulent velocity profile. This can be explained by the full boundary layer resolution approach. LRN models **resolve** the flow in the boundary layers, while HRN models with wall functions **model** the flow in the near-wall area.

4.5 Evaluation of models

This section evaluates the applied mesh topology, flow models, wall functions and analytical models.

4.5.1 Evaluation of mesh topology

For this particular flow geometry and problem, the OH-grid is not considered to give good **mesh optimality** for standard wall functions. When the circular cross-section of the pipe is expanding or contracting, the aspect ratio of the OH-grid in the radial direction stays constant. This makes it difficult to control the height of the first node in all the sections of the pipe. Moreover, it was not possible to increase the $y^{+} > 30$ in most areas without getting too low **mesh quality**.

However, for the full resolution approach, the OH-grid gives good **mesh optimality**, and it is even possible to decrease the values of y^{+} more by refining the mesh in the near-wall area. The drawback of the fine grid close to the wall is that the cells there attain a large aspect ratio, which reduces the **mesh quality**.

4.5.2 Evaluation of flow models

According to Kalitzin et al. [21] RANS models are unable to predict transitional flow satisfactorily. Turbulence models should generally not be used to model transitional and laminar flows because it can cause erroneous results.

In this case, turbulence models were applied to a complex geometry with turbulent, transition and laminar flow regimes. However, the models overall seem to have predicted the general flow behavior quite consistent with the analytical solution.

In the discussion forums, some claim that the $k - \omega\ SST$ model performs better in laminar flow than the $k - \varepsilon$ model. However, there is no clear indication that this is the case here, and any discrepancy from the $k - \varepsilon$ model is more likely caused by the difference in wall treatment.

4.5.3 Evaluation of wall functions

Assessment of the pressure and velocity results show that the adjusted wall functions applied in Case B2 seem to have predicted the flow slightly better than the default wall functions utilized in Case B1. Thus, the intention of better accuracy with the adjusted wall functions was achieved.

4.5.4 Evaluation of analytical models

The analytical solutions presented are based on approximations and simplifications and generally do not take into account the specific flow phenomena occurring in this particular model. Therefore, it can be difficult to know for sure what cases obtained the best result in a given comparison, especially if the cases do not show large deviations from each other.

5 Conclusion

Theory concerning mesh quality parameters and turbulence modeling have been presented and an evaluation of different open-source meshing tools has been made. Salome was evaluated to be the most user-friendly tool and was used to create and mesh a 3D model of a diverging converging pipe with belly section. 3 meshes with a hexahedral OH-grid and different wall treatments were constructed and imported to OpenFOAM.

4 simulations of air flowing through the pipe were run with the steady-state turbulent incompressible flow solver simpleFoam. Case A and Case B1 applied Mesh A and Mesh B, respectively, with the *standard* $k - \varepsilon$ model and default wall functions. Case B2 used Mesh B with the *standard* $k - \varepsilon$ model and adjusted wall functions. Case C utilized Mesh C with the $k - \omega$ SST model and full boundary layer resolution.

The computed average y^+ values were 3.7, 29, 27 and 0.55 for Cases A, B1, B2 and C, respectively. Case A did not fulfill the wall function requirements. The OH-grid topology was evaluated to give good mesh optimality for the full resolution approach and unsatisfactory mesh optimality for the wall function approach. An evaluation of basic fluid mechanics checkpoints, symmetry and residuals did not indicate numerical errors in the solutions.

The stagnation pressure loss agreed quite well with the analytical solution for Cases B1, B2 and C. Case B2 showed the most accurate result, while Case A showed results diverging from the analytical solution and obtained too large pressure loss.

All the cases showed consistent average velocity results compared with the analytical solution, except for in the inlet region of the outlet section, where all the simulations showed a lower velocity than predicted. Comparisons of maximum velocities along the centerline with the analytical solution showed that Case C achieved the best agreement with the analytical solution. Cases B1 and B2 displayed some deviations from the analytical solution. Assessment of the turbulence velocity profile demonstrated that Case A and Case C obtained results quite consistent with the analytical solution. Case B1 and Case B2 diverged from the analytical solution in the area near the wall, which can be explained by the wall functions.

The simulation cases generally obtained results that agreed well with the analytical solutions, even though the complex flow field of turbulent, transitional and laminar flow was modeled with turbulence models. The adjusted wall functions gave slightly better results than the default wall functions for the $k - \varepsilon$ model. The $k - \varepsilon$ model on Mesh B obtained pressure results that agreed well with the analytical solution. The $k - \omega$ SST model generally predicted the velocity field best, and also showed consistent results in terms of pressure loss.

A student tutorial describing the workflow of case B1 was made.

5.1 Further work

Possible further work on the simulation cases could be to make additional assessments of the results. Additional parameters could be checked, and force plots could be implemented. For the $k - \omega\ SST$ model, a grid independence study could be performed to check if the mesh resolution influences the results. Furthermore, the length of the inlet and outlet sections could be increased to $60D$ to achieve fully turbulent flow for all cases and to get a better overview of the flow development.

The student tutorial has many areas of improvement. The main weakness of the tutorial is that it only provides a step-by-step explanation of the workflow. It may be a good idea to add some theory about weaknesses of the model, mesh quality, boundary layers, wall functions and calculation formulas for k and ε. Furthermore, descriptions of some steps could be more detailed, tables and figures could be referred to, and the post-processing in ParaView could be described.

When learning to use a new meshing software, it can be difficult to follow and understand the steps of a written tutorial. Further work could involve making a supplementary video of the tutorial. It would also be a good idea to provide the hdf-file of the complete geometry and mesh in Salome to the students for download.

References

[1] V. Hernandez-Perez, M. Abdulkadir, and B. J. Azzopardi, "Grid Generation Issues in the CFD Modelling of Two-Phase Flow in a Pipe," *The Journal of Computational Multiphase Flows*, vol. 3, no. 1, pp. 13-26, 2011. [Online]. Available: doi: 10.1260/1757-482X.3.1.13

[2] Z. Gao, R. Bresson, Y. Qu, M. Milliez, C. de Munck, and B. Carissimo, "High resolution unsteady RANS simulation of wind, thermal effects and pollution dispersion for studying urban renewal scenarios in a neighborhood of Toulouse," *Urban climate,* vol. 23, pp. 114-130, 2018. [Online]. Available: doi: 10.1016/j.uclim.2016.11.002

[3] S. M. Nour, "CFD simulations of flow in bent pipe at high Reynolds numbers conditions," Master's Thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo, 2014. [Online]. Available: http://urn.nb.no/URN:NBN:no-46977

[4] U. C. Liestyarini, "CFD Analysis of Internal Pipe Flows," Master's Thesis, Faculty of Science and Technology, University of Stavanger, Stavanger, 2016. [Online]. Available: http://hdl.handle.net/11250/2411432

[5] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*, 2. ed. Harlow: Pearson Education Limited, 2007, pp. 57-59, 76-78, 85, 88, 90-92, 97, 106, 151, 264, 275-279, 283, 286, 292, 296, 301-302, 305, 308-312, 342.

[6] J. Tu, G. H. Yeoh, and C. Liu, *Computational Fluid Dynamics: A Practical Approach,* 3 ed. Butterworth-Heinemann, 2018, pp. 125, 133, 136, 139-143, 149-150.

[7] H. Jasak, "Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows," Ph.D. Thesis, Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, London, 1996. [Online]. Available: https://spiral.imperial.ac.uk/bitstream/10044/1/8335/1/Hrvoje_Jasak-1996-PhD-Thesis.pdf

[8] Shyam2791, *Basic three-dimensional cell shapes,* Licenced under CC BY-SA 3.0 / Modified, Nov. 15, 2012. Accessed on: Apr. 17, 2021. [Online figure]. Available: https://en.wikipedia.org/wiki/Types_of_mesh

[9] J. Guerrero, *Mesh quality metrics. Smoothness,* Licenced under CC BY-SA 4.0, Wolf Dynamics, 2020. Accessed on: Apr. 9, 2021. [Online figure]. Available: http://www.wolfdynamics.com/training/introOF8/all.pdf

[10] J. Guerrero, *Mesh quality metrics. Mesh aspect ratio AR,* Licenced under CC BY-SA 4.0, Wolf Dynamics, 2020. Accessed on: Apr. 9, 2021. [Online figure]. Available: http://www.wolfdynamics.com/training/introOF8/all.pdf

[11] Shyam2791, *Types of mesh,* Licenced under CC BY-SA 3.0, Nov. 15, 2012. Accessed on: Apr. 17, 2021. [Online figure]. Available: https://en.wikipedia.org/wiki/Types_of_mesh

[12] J. Guerrero, *OpenFOAM® Introductory Training Online session – 2020 Edition*, Wolf Dynamics, 2020. Accessed on: Apr. 9, 2021. [Online]. Available: http://www.wolfdynamics.com/training/introOF8/all.pdf

[13] A. Syrakos, S. Varchanis, Y. Dimakopoulos, A. Goulas, and J. Tsamopoulos, "A critical analysis of some popular methods for the discretisation of the gradient operator in finite volume methods," *Physics of Fluids,* vol. 29, no. 12, p. 127103, 2017. [Online]. Available: doi: 10.1063/1.4997682

[14] R. N. Gopalakrishnan and P. J. Disimile, "CFD Analysis of Twin Turbulent Impinging Round Jets at Different Impingement Angles," *Fluids,* vol. 3, no. 4, 79, 2018. [Online]. Available: doi: 10.3390/fluids3040079

[15] V. Hernandez Perez, "Gas-liquid two-phase flow in inclined pipes," Ph.D. Thesis, Department of Chemical and Environmental Engineering, University of Nottingham, 2008. [Online]. Available: http://eprints.nottingham.ac.uk/11764/1/Hernandez-Perez.pdf

[16] P. Zhang, R. M. Roberts, and A. Bénard, "Computational guidelines and an empirical model for particle deposition in curved pipes using an Eulerian-Lagrangian approach," *Journal of Aerosol Science,* vol. 53, pp. 1-20, 2012. [Online]. Available: doi: 10.1016/j.jaerosci.2012.05.007

[17] Aokomoriuta, *Law of the wall,* Licenced under CC BY-SA 3.0, July 1, 2011. Accessed on: March. 17, 2021. [Online figure]. Available: https://en.wikipedia.org/wiki/Law_of_the_wall

[18] J. B. Franzini and E. J. Finnemore, *Fluid Mechanics with Engineering Applications,* 10 ed. New York: McGraw-Hill, 2002, pp. 109, 139, 145, 261, 265, 284, 362.

[19] F. M. White, *Fluid Mechanics*, 7th ed. New York: McGraw-Hill, 2011, p. 473.

[20] D. B. Spalding, "A Single Formula for the “Law of the Wall”," *Journal of Applied Mechanics,* vol. 28, no. 3, pp. 455-458, 1961. [Online]. Available: doi: 10.1115/1.3641728

[21] G. Kalitzin, G. Medic, G. Iaccarino, and P. Durbin, "Near-wall behavior of RANS turbulence models and implications for wall functions," *Journal of Computational Physics,* vol. 204, no. 1, pp. 265-291, 2005. [Online]. Available: doi: 10.1016/j.jcp.2004.10.018

[22] J. Bredberg, "On the Wall Boundary Condition for Turbulence Models," Chalmers University of Technology, Department of Thermo and Fluid Dynamics, Technical Report 00/4, 2000. Accessed on: Apr. 4, 2021. [Online]. Available: http://www.tfd.chalmers.se/~lada/postscript_files/jonas_report_WF.pdf

[23] F. Liu, "Thorough Description Of How Wall Functions Are Implemented In OpenFOAM," in "In Proceedings of CFD with OpenSource Software," edited by: H. Nilsson and M. Irannezhad, Chalmers University of Technology, Göteborg, Project Report, 2016. Accessed on: Apr. 4, 2021. [Online]. Available: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016/FangqingLiu/openfoamFinal.pdf

[24] S. N. Liu, "Implementation of a Complete Wall Function for the Standard k−epsilon Turbulence Model in OpenFOAM 4.0," in "In Proceedings of CFD with OpenSource

Software," edited by: H. Nilsson and M. Arabnejad, University of Stavanger, Stavanger, Project Work, 2016. Accessed on: Apr. 4, 2021. [Online]. Available: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016/ShengnanLiu/FinalReport-Shengnan.pdf

[25] G. Chen, Q. Xiong, P. J. Morris, E. G. Paterson, A. Sergeev, and Y.-C. Wang, "OpenFOAM for Computational Fluid Dynamics," *Notices of the American Mathematical Society,* vol. 61, no. 4, pp. 354-363, 2014. [Online]. Available: doi: 10.1090/noti1095

[26] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine, *Incropera's Principles of Heat and Mass Transfer*, 8. Global. ed. Singapore: Wiley, 2017, pp. 467-477.

[27] B. E. Launder and D. B. Spalding, "The numerical computation of turbulent flows," *Computer Methods in Applied Mechanics and Engineering,* vol. 3, no. 2, pp. 269-289, 1974. [Online]. Available: doi: 10.1016/0045-7825(74)90029-2

[28] S. H. E. Tahry, "k-epsilon Equation for Compressible Reciprocating Engine Flows," *Journal of Energy,* vol. 7, no. 4, pp. 345-353, 1983. [Online]. Available: doi: 10.2514/3.48086

[29] OpenFOAM, *k-epsilon,* OpenFOAM: User Guide v2006, OpenCFD Ltd. Accessed on: Apr. 4, 2021. [Online]. Available: https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras-k-epsilon.html

[30] ANSYS, Inc., "ANSYS Fluent User's Guide, Release 19.0," ANSYS, Inc., 2018.

[31] F. Menter and T. Esch, "Elements of Industrial Heat Transfer Prediction," presented at the *16th Brazilian Congr. of Mechanical Engineering (COBEM),* Nov. 2001.

[32] F. Menter, M. Kuntz, and R. Langtry, "Ten Years of Industrial Experience with the SST Turbulence Model," in *Proc. of the 4th Int. Symp. on Turbulence, Heat and Mass Transfer,* Antalya, Turkey, Begell House, 2003, pp. 625–632.

[33] OpenFOAM, *k-omega Shear Stress Transport (SST),* OpenFOAM: User Guide v2012, OpenCFD Ltd. Accessed on: Apr. 24, 2021. [Online]. Available: https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras-k-omega-sst.html

[34] OpenFOAM®, *Steady turbulent flow over a backward-facing step,* Tutorial Guide, OpenCFD Ltd. Accessed on: May 6, 2021. [Online]. Available: https://www.openfoam.com/documentation/tutorial-guide/3-compressible-flow/3.1-steady-turbulent-flow-over-a-backward-facing-step

[35] P. K. Swamee and A. Jain, "Explicit Equations for Pipe-Flow Problems," *Journal of Hydraulic Division,* vol. 102, no. 5, pp. 657-664, 1976. [Online]. Available: doi: 10.1061/JYCEAJ.0004542

[36] H. Blasius, "Das Aehnlichkeitsgesetz bei Reibungsvorgängen in Flüssigkeiten," *VDI Forsch. Gebiete Ingenieurw.,* vol. 131, pp. 1-39, 1913. [Online]. Available: doi: 10.1007/978-3-662-02239-9_1

[37] T. Al-Shemmeri, *Engineering Fluid Mechanics,* 1. ed. Bookboon, 2012, p. 69. [Online]. Available: https://bookboon.com/nb/engineering-fluid-mechanics-ebook

[38] SALOME, *Introduction to HEXABLOCK*, CEA/DEN, EDF R&D. Accessed on: Apr. 6, 2021. [Online]. Available: https://docs.salome-platform.org/latest/gui/HEXABLOCK/general.html

[39] R. W. Pitz and J. W. Daily, "Combustion in a turbulent mixing layer formed at a rearward-facing step," *AIAA Journal*, vol. 21, no. 11, pp. 1565-1570, 1983. [Online]. Available: doi: 10.2514/3.8290

[40] SimScale, *Turbulent Pipe Flow*, 2019. Accessed on: Nov. 23, 2019. [Online]. Available: https://www.simscale.com/docs/validation/TurbulentPipeFlow/TurbulentPipeFlow.html

[41] C. J. Greenshields, "OpenFOAM User Guide version 8," OpenFOAM Foundation Ltd., CFD Direct Ltd., July 22, 2020.

[42] D. A. Jones, M. Chapuis, M. Liefvendahl, D. Norrison, and R. Widjaja, "RANS Simulations using OpenFOAM Software," Australian Government: Department of Defence: Defence Science and Technology Group, Technical Report DST-Group-TR-3204, Jan. Jan., 2016. Accessed on: Mar. 22, 2021. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1002391.pdf

[43] E. Furbo, "Evaluation of RANS turbulence models for flow problems with signigicant impact of boundary layers," Master's Thesis, Department of Information Technology, Uppsala University, Uppsala, 2010. [Online]. Available: http://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A379743

[44] A. Lipej, S. Muhič, and D. Mitruševski, "Wall Roughness Influence on the Efficiency Characteristics of Centrifugal Pump," *Strojniški vestnik - Journal of Mechanical Engineering*, vol. 63, no. 9, pp. 529-536, 2017. [Online]. Available: doi: 10.5545/sv-jme.2017.4526

[45] J. S. Jayakumar and R. Manuraj, "Estimation of local wall shear stress in curved pipes using cgns mesh in OpenFOAM," presented at the *5th Int. and 41st Nat. Conf. on Fluid Mechanics and Fluid Power (FMFP) 2014*, Indian Institute of Technology Kanpur, 2014. Accessed on: Apr. 29, 2021. [Online]. Available: https://ijcrt.org/papers/IJCRT_193642.pdf

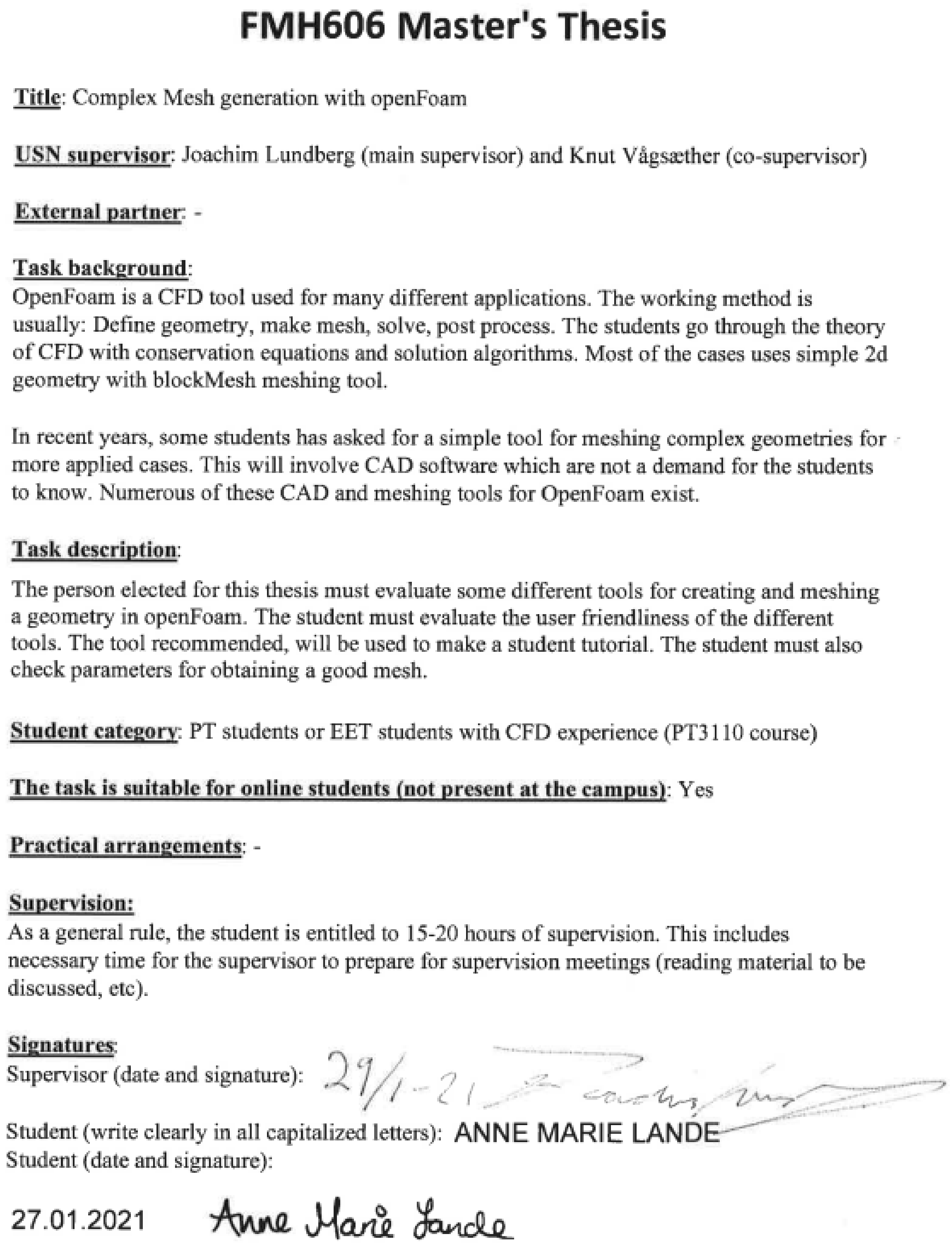
[46] OpenFOAM, *Turbulent plane channel flow with smooth walls*, OpenFOAM: User Guide v2012, OpenCFD Ltd. Accessed on: Apr. 25, 2021. [Online]. Available: https://www.openfoam.com/documentation/guides/latest/doc/verification-validation-turbulent-plane-channel-flow.html

[47] J. C. Puig, *Laminar flow through a circular pipe*, OPENFOAM GUIDE FOR BEGINNERS, UPC Sch. of Prof. & Exec. Dev., ESEIAAT. Accessed on: Apr. 25, 2021. [Online]. Available: http://files.the-foam-house5.webnode.es/200000363-59a695aa6c/Chapter4_Pipe.pdf

[48] SimScale, *How to Model Wall Roughness in CFD?*, 2020. Accessed on: Apr. 29, 2021. [Online]. Available: https://www.simscale.com/knowledge-base/how-to-model-wall-roughness-in-cfd/

Appendices

Appendix A - Project topic description



FMH606 Master's Thesis

Title: Complex Mesh generation with openFoam

USN supervisor: Joachim Lundberg (main supervisor) and Knut Vågsæther (co-supervisor)

External partner: -

Task background:
OpenFoam is a CFD tool used for many different applications. The working method is usually: Define geometry, make mesh, solve, post process. The students go through the theory of CFD with conservation equations and solution algorithms. Most of the cases uses simple 2d geometry with blockMesh meshing tool.

In recent years, some students has asked for a simple tool for meshing complex geometries for more applied cases. This will involve CAD software which are not a demand for the students to know. Numerous of these CAD and meshing tools for OpenFoam exist.

Task description:

The person elected for this thesis must evaluate some different tools for creating and meshing a geometry in openFoam. The student must evaluate the user friendliness of the different tools. The tool recommended, will be used to make a student tutorial. The student must also check parameters for obtaining a good mesh.

Student category: PT students or EET students with CFD experience (PT3110 course)

The task is suitable for online students (not present at the campus): Yes

Practical arrangements: -

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:
Supervisor (date and signature): 29/1-21 [illegible]

Student (write clearly in all capitalized letters): ANNE MARIE LANDE
Student (date and signature):

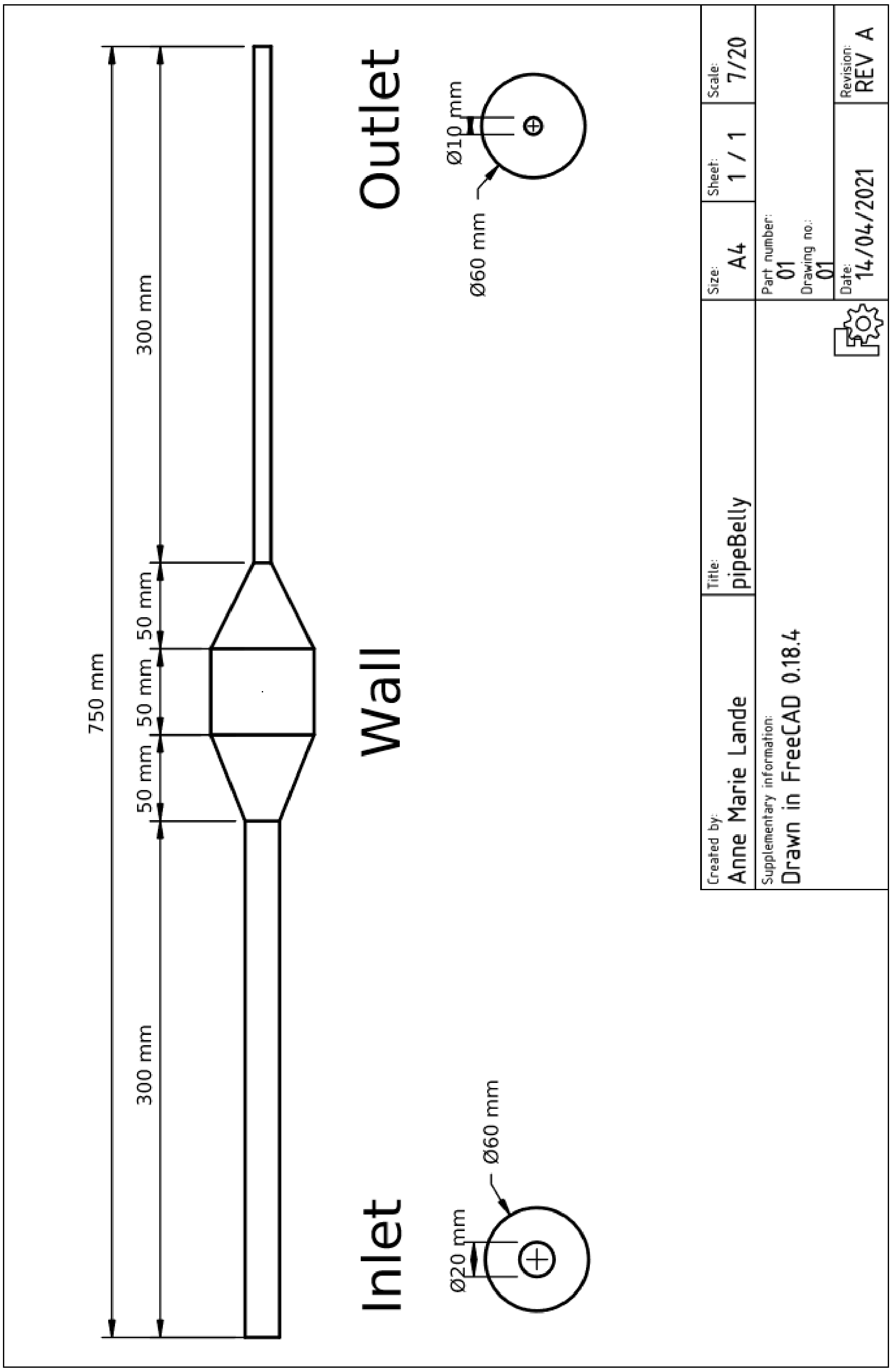
27.01.2021 Anne Marie Lande

Appendix B - Software

The software used for the cases are listed below:

- Oracle VM VirtualBox 6.1-18
- Ubuntu 20.04.2 LTS (focal)
- Salome 9.6.0
- openfoam8
- paraviewopenfoam56
- 40 GB RAM and 4 cores assigned to the virtual machine

Appendix C - Technical drawing geometry



Appendix D - Python script for Mesh A, B and C

```
#!/usr/bin/env python

### This file is generated automatically by SALOME v9.6.0 with dump python functionality
###

import sys
import salome

salome.salome_init()
import salome_notebook
notebook = salome_notebook.NoteBook()
sys.path.insert(0, r'/home/virtualbox/Documents/salome_files_FINAL')

###
### GEOM component
###

import GEOM
from salome.geom import geomBuilder
import math
import SALOMEDS


geompy = geomBuilder.New()

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)
geomObj_1 = geompy.MakeMarker(0, 0, 0, 1, 0, 0, 0, 1, 0)
sk = geompy.Sketcher2D()
sk.addPoint(0.000000, 0.000000)
sk.addSegmentAbsolute(300.000000, 0.000000)
sk.addSegmentAbsolute(350.000000, 0.000000)
sk.addSegmentAbsolute(400.000000, 0.000000)
sk.addSegmentAbsolute(450.000000, 0.000000)
sk.addSegmentAbsolute(750.000000, 0.000000)
Sketch_1 = sk.wire(geomObj_1)
[Edge_1,Edge_2,Edge_3,Edge_4,Edge_5] = geompy.ExtractShapes(Sketch_1, geompy.ShapeType["EDG
E"], True)
[Vertex_1,Vertex_2,Vertex_3,Vertex_4,Vertex_5,Vertex_6] = geompy.ExtractShapes(Sketch_1, ge
ompy.ShapeType["VERTEX"], True)
Wire_1 = geompy.MakeWire([Edge_1, Edge_2, Edge_3, Edge_4, Edge_5], 1e-07)
Divided_Disk_1 = geompy.MakeDividedDiskPntVecR(Vertex_1, Edge_1, 10, GEOM.SQUARE)
[Face_1, Face_2, Face_3] = geompy.Propagate(Divided_Disk_1)
[Face_1,Face_2,Face_3,Face_4,Face_5] = geompy.ExtractShapes(Divided_Disk_1, geompy.ShapeTyp
e["FACE"], True)
Divided_Disk_2 = geompy.MakeDividedDiskPntVecR(Vertex_2, Edge_1, 10, GEOM.SQUARE)
[Face_6,Face_7,Face_8,Face_9,Face_10] = geompy.ExtractShapes(Divided_Disk_2, geompy.ShapeTy
pe["FACE"], True)
Divided_Disk_3 = geompy.MakeDividedDiskPntVecR(Vertex_3, Edge_1, 30, GEOM.SQUARE)
[Face_11,Face_12,Face_13,Face_14,Face_15] = geompy.ExtractShapes(Divided_Disk_3, geompy.Sha
peType["FACE"], True)
Divided_Disk_4 = geompy.MakeDividedDiskPntVecR(Vertex_4, Edge_1, 30, GEOM.SQUARE)
[Face_16,Face_17,Face_18,Face_19,Face_20] = geompy.ExtractShapes(Divided_Disk_4, geompy.Sha
peType["FACE"], True)
Divided_Disk_5 = geompy.MakeDividedDiskPntVecR(Vertex_5, Edge_1, 5, GEOM.SQUARE)
```

```
[Face_21,Face_22,Face_23,Face_24,Face_25] = geompy.ExtractShapes(Divided_Disk_5, geompy.Sha
peType["FACE"], True)
Divided_Disk_6 = geompy.MakeDividedDiskPntVecR(Vertex_6, Edge_1, 5, GEOM.SQUARE)
[Face_26,Face_27,Face_28,Face_29,Face_30] = geompy.ExtractShapes(Divided_Disk_6, geompy.Sha
peType["FACE"], True)
Pipe_1 = geompy.MakePipeWithDifferentSectionsBySteps([Face_1, Face_6, Face_11, Face_16, Fac
e_21, Face_26], [Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6], Wire_1)
Pipe_2 = geompy.MakePipeWithDifferentSectionsBySteps([Face_2, Face_7, Face_12, Face_17, Fac
e_22, Face_27], [Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6], Wire_1)
Pipe_3 = geompy.MakePipeWithDifferentSectionsBySteps([Face_3, Face_8, Face_13, Face_18, Fac
e_23, Face_28], [Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6], Wire_1)
Pipe_4 = geompy.MakePipeWithDifferentSectionsBySteps([Face_4, Face_9, Face_14, Face_19, Fac
e_24, Face_29], [Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6], Wire_1)
Pipe_5 = geompy.MakePipeWithDifferentSectionsBySteps([Face_5, Face_10, Face_15, Face_20, Fa
ce_25, Face_30], [Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6], Wire_1)
Compound_1 = geompy.MakeCompound([Pipe_1, Pipe_2, Pipe_3, Pipe_4, Pipe_5])
Glue_1 = geompy.MakeGlueFaces(Compound_1, 1e-07)
[corners, flowedges_inlet_outlet, flowedges_belly, Compound_5, Compound_6, Compound_7, Comp
ound_8, Compound_9] = geompy.Propagate(Glue_1)
flowedges_inlet_outlet = geompy.UnionListOfGroups([Compound_5, Compound_9])
flowedges_belly = geompy.UnionListOfGroups([Compound_6, Compound_7, Compound_8])
inlet = geompy.CreateGroup(Glue_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(inlet, [296, 136, 205, 32, 254])
outlet = geompy.CreateGroup(Glue_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(outlet, [310, 190, 114, 288, 239])
wall = geompy.CreateGroup(Glue_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(wall, [158, 171, 128, 145, 184, 101, 41, 14, 61, 81, 280, 243, 256, 264, 27
2, 304, 298, 292, 301, 307])
[corners, Compound_5, Compound_6, Compound_7, Compound_8, Compound_9, flowedges_inlet_outle
t, flowedges_belly, inlet, outlet, wall] = geompy.GetExistingSubObjects(Glue_1, False)
[corners, Compound_5, Compound_6, Compound_7, Compound_8, Compound_9, flowedges_inlet_outle
t, flowedges_belly, inlet, outlet, wall] = geompy.GetExistingSubObjects(Glue_1, False)
[corners, Compound_5, Compound_6, Compound_7, Compound_8, Compound_9, flowedges_inlet_outle
t, flowedges_belly, inlet, outlet, wall] = geompy.GetExistingSubObjects(Glue_1, False)
geompy.addToStudy( O, 'O' )
geompy.addToStudy( OX, 'OX' )
geompy.addToStudy( OY, 'OY' )
geompy.addToStudy( OZ, 'OZ' )
geompy.addToStudy( Sketch_1, 'Sketch_1' )
geompy.addToStudyInFather( Sketch_1, Edge_1, 'Edge_1' )
geompy.addToStudyInFather( Sketch_1, Edge_2, 'Edge_2' )
geompy.addToStudyInFather( Sketch_1, Edge_3, 'Edge_3' )
geompy.addToStudyInFather( Sketch_1, Edge_4, 'Edge_4' )
geompy.addToStudyInFather( Sketch_1, Edge_5, 'Edge_5' )
geompy.addToStudyInFather( Sketch_1, Vertex_1, 'Vertex_1' )
geompy.addToStudyInFather( Sketch_1, Vertex_2, 'Vertex_2' )
geompy.addToStudyInFather( Sketch_1, Vertex_3, 'Vertex_3' )
geompy.addToStudyInFather( Sketch_1, Vertex_4, 'Vertex_4' )
geompy.addToStudyInFather( Sketch_1, Vertex_5, 'Vertex_5' )
geompy.addToStudyInFather( Sketch_1, Vertex_6, 'Vertex_6' )
geompy.addToStudy( Wire_1, 'Wire_1' )
geompy.addToStudy( Divided_Disk_1, 'Divided Disk_1' )
geompy.addToStudy( Divided_Disk_2, 'Divided Disk_2' )
geompy.addToStudy( Divided_Disk_3, 'Divided Disk_3' )
geompy.addToStudy( Divided_Disk_4, 'Divided Disk_4' )
geompy.addToStudy( Divided_Disk_5, 'Divided Disk_5' )
geompy.addToStudy( Divided_Disk_6, 'Divided Disk_6' )
geompy.addToStudyInFather( Divided_Disk_1, Face_1, 'Face_1' )
geompy.addToStudyInFather( Divided_Disk_1, Face_2, 'Face_2' )
geompy.addToStudyInFather( Divided_Disk_1, Face_3, 'Face_3' )
geompy.addToStudyInFather( Divided_Disk_1, Face_4, 'Face_4' )
geompy.addToStudyInFather( Divided_Disk_1, Face_5, 'Face_5' )
geompy.addToStudyInFather( Divided_Disk_2, Face_6, 'Face_6' )
geompy.addToStudyInFather( Divided_Disk_2, Face_7, 'Face_7' )
geompy.addToStudyInFather( Divided_Disk_2, Face_8, 'Face_8' )
geompy.addToStudyInFather( Divided_Disk_2, Face_9, 'Face_9' )
geompy.addToStudyInFather( Divided_Disk_2, Face_10, 'Face_10' )
geompy.addToStudyInFather( Divided_Disk_3, Face_11, 'Face_11' )
geompy.addToStudyInFather( Divided_Disk_3, Face_12, 'Face_12' )
geompy.addToStudyInFather( Divided_Disk_3, Face_13, 'Face_13' )
geompy.addToStudyInFather( Divided_Disk_3, Face_14, 'Face_14' )
geompy.addToStudyInFather( Divided_Disk_3, Face_15, 'Face_15' )
geompy.addToStudyInFather( Divided_Disk_4, Face_16, 'Face_16' )
geompy.addToStudyInFather( Divided_Disk_4, Face_17, 'Face_17' )
geompy.addToStudyInFather( Divided_Disk_4, Face_18, 'Face_18' )
geompy.addToStudyInFather( Divided_Disk_4, Face_19, 'Face_19' )
geompy.addToStudyInFather( Divided_Disk_4, Face_20, 'Face_20' )
geompy.addToStudyInFather( Divided_Disk_5, Face_21, 'Face_21' )
geompy.addToStudyInFather( Divided_Disk_5, Face_22, 'Face_22' )
geompy.addToStudyInFather( Divided_Disk_5, Face_23, 'Face_23' )
geompy.addToStudyInFather( Divided_Disk_5, Face_24, 'Face_24' )
geompy.addToStudyInFather( Divided_Disk_5, Face_25, 'Face_25' )
geompy.addToStudyInFather( Divided_Disk_6, Face_26, 'Face_26' )
geompy.addToStudyInFather( Divided_Disk_6, Face_27, 'Face_27' )
geompy.addToStudyInFather( Divided_Disk_6, Face_28, 'Face_28' )
geompy.addToStudyInFather( Divided_Disk_6, Face_29, 'Face_29' )
geompy.addToStudyInFather( Divided_Disk_6, Face_30, 'Face_30' )
geompy.addToStudy( Pipe_1, 'Pipe_1' )
geompy.addToStudy( Pipe_2, 'Pipe_2' )
geompy.addToStudy( Pipe_3, 'Pipe_3' )
geompy.addToStudy( Pipe_4, 'Pipe_4' )
geompy.addToStudy( Pipe_5, 'Pipe_5' )
geompy.addToStudy( Compound_1, 'Compound_1' )
geompy.addToStudy( Glue_1, 'Glue_1' )
geompy.addToStudyInFather( Glue_1, corners, 'corners' )
geompy.addToStudyInFather( Glue_1, flowedges_inlet_outlet, 'flowedges_inlet_outlet' )
geompy.addToStudyInFather( Glue_1, flowedges_belly, 'flowedges_belly' )
geompy.addToStudyInFather( Glue_1, Compound_5, 'Compound_5' )
geompy.addToStudyInFather( Glue_1, Compound_6, 'Compound_6' )
geompy.addToStudyInFather( Glue_1, Compound_7, 'Compound_7' )
geompy.addToStudyInFather( Glue_1, Compound_8, 'Compound_8' )
geompy.addToStudyInFather( Glue_1, Compound_9, 'Compound_9' )
geompy.addToStudyInFather( Glue_1, inlet, 'inlet' )
geompy.addToStudyInFather( Glue_1, outlet, 'outlet' )
geompy.addToStudyInFather( Glue_1, wall, 'wall' )

###
### SMESH component
###

import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New()

MeshA = smesh.Mesh(Glue_1)
Regular_1D = MeshA.Segment()
Number_of_Segments_10 = Regular_1D.NumberOfSegments(10)
Quadrangle_2D = MeshA.Quadrangle(algo=smeshBuilder.QUADRANGLE)
Hexa_3D = MeshA.Hexahedron(algo=smeshBuilder.Hexa)
inlet_1 = MeshA.GroupOnGeom(inlet,'inlet',SMESH.FACE)
outlet_1 = MeshA.GroupOnGeom(outlet,'outlet',SMESH.FACE)
wall_1 = MeshA.GroupOnGeom(wall,'wall',SMESH.FACE)
isDone = MeshA.Compute()
```

```
[ smeshObj_1, smeshObj_2, smeshObj_3, smeshObj_4, smeshObj_5, smeshObj_6, smeshObj_7, smesh
Obj_8, inlet_1, outlet_1, wall_1 ] = MeshA.GetGroups()
Regular_1D_1 = MeshA.Segment(geom=flowedges_inlet_outlet)
Number_of_Segments_300 = Regular_1D_1.NumberOfSegments(100)
[ smeshObj_1, smeshObj_2, smeshObj_3, smeshObj_4, smeshObj_5, smeshObj_6, smeshObj_7, smesh
Obj_8, inlet_1, outlet_1, wall_1 ] = MeshA.GetGroups()
Regular_1D_2 = MeshA.Segment(geom=flowedges_belly)
Number_of_Segments_50 = Regular_1D_2.NumberOfSegments(50)
[ smeshObj_1, smeshObj_2, smeshObj_3, smeshObj_4, smeshObj_5, smeshObj_6, smeshObj_7, smesh
Obj_8, inlet_1, outlet_1, wall_1 ] = MeshA.GetGroups()
Number_of_Segments_300.SetNumberOfSegments( 300 )
isDone = MeshA.Compute()
[ smeshObj_1, smeshObj_2, smeshObj_3, smeshObj_4, smeshObj_5, smeshObj_6, smeshObj_7, smesh
Obj_8, inlet_1, outlet_1, wall_1 ] = MeshA.GetGroups()
MeshB = smesh.Mesh(Glue_1)
status = MeshB.AddHypothesis(Number_of_Segments_10)
Regular_1D_3 = MeshB.Segment()
Quadrangle_2D_1 = MeshB.Quadrangle(algo=smeshBuilder.QUADRANGLE)
Hexa_3D_1 = MeshB.Hexahedron(algo=smeshBuilder.Hexa)
inlet_2 = MeshB.GroupOnGeom(inlet,'inlet',SMESH.FACE)
outlet_2 = MeshB.GroupOnGeom(outlet,'outlet',SMESH.FACE)
wall_2 = MeshB.GroupOnGeom(wall,'wall',SMESH.FACE)
Regular_1D_4 = MeshB.Segment(geom=flowedges_inlet_outlet)
status = MeshB.AddHypothesis(Number_of_Segments_300,flowedges_inlet_outlet)
Regular_1D_5 = MeshB.Segment(geom=flowedges_belly)
status = MeshB.AddHypothesis(Number_of_Segments_50,flowedges_belly)
[ smeshObj_9, smeshObj_10, smeshObj_11, smeshObj_12, smeshObj_13, smeshObj_14, smeshObj_15,
 smeshObj_16, inlet_2, outlet_2, wall_2 ] = MeshB.GetGroups()
Regular_1D_6 = MeshB.Segment(geom=corners)
Number_of_Segments_2 = Regular_1D_6.NumberOfSegments(2,0.4,[])
[ smeshObj_9, smeshObj_10, smeshObj_11, smeshObj_12, smeshObj_13, smeshObj_14, smeshObj_15,
 smeshObj_16, inlet_2, outlet_2, wall_2 ] = MeshB.GetGroups()
Number_of_Segments_2.SetNumberOfSegments( 2 )
Number_of_Segments_2.SetScaleFactor( 0.4 )
Number_of_Segments_2.SetReversedEdges( [ 9, 13, 40, 60, 80, 100, 123, 127, 144, 157, 170, 1
83 ] )
isDone = MeshB.Compute()
[ inlet_2, outlet_2, wall_2 ] = MeshB.GetGroups()
MeshC = smesh.Mesh(Glue_1)
status = MeshC.AddHypothesis(Number_of_Segments_10)
Regular_1D_7 = MeshC.Segment()
Quadrangle_2D_2 = MeshC.Quadrangle(algo=smeshBuilder.QUADRANGLE)
Hexa_3D_2 = MeshC.Hexahedron(algo=smeshBuilder.Hexa)
inlet_3 = MeshC.GroupOnGeom(inlet,'inlet',SMESH.FACE)
outlet_3 = MeshC.GroupOnGeom(outlet,'outlet',SMESH.FACE)
wall_3 = MeshC.GroupOnGeom(wall,'wall',SMESH.FACE)
Regular_1D_8 = MeshC.Segment(geom=flowedges_inlet_outlet)
status = MeshC.AddHypothesis(Number_of_Segments_300,flowedges_inlet_outlet)
Regular_1D_9 = MeshC.Segment(geom=flowedges_belly)
status = MeshC.AddHypothesis(Number_of_Segments_50,flowedges_belly)
[ smeshObj_17, smeshObj_18, smeshObj_19, smeshObj_20, smeshObj_21, smeshObj_22, smeshObj_23
, smeshObj_24, inlet_3, outlet_3, wall_3 ] = MeshC.GetGroups()
Regular_1D_10 = MeshC.Segment(geom=corners)
Regular_1D_11 = MeshC.Segment(geom=corners)
Number_of_Segments_20 = Regular_1D_10.NumberOfSegments(20,10,[ 9, 13, 40, 60, 80, 100, 123,
 127, 144, 157, 170, 183 ])
isDone = MeshC.Compute()
[ smeshObj_17, smeshObj_18, smeshObj_19, smeshObj_20, smeshObj_21, smeshObj_22, smeshObj_23
, smeshObj_24, inlet_3, outlet_3, wall_3 ] = MeshC.GetGroups()
flowedges_inlet_outlet_1 = Regular_1D_1.GetSubMesh()
flowedges_belly_1 = Regular_1D_2.GetSubMesh()
flowedges_inlet_outlet_2 = Regular_1D_4.GetSubMesh()
```

```
flowedges_belly_2 = Regular_1D_5.GetSubMesh()
corners_1 = Regular_1D_6.GetSubMesh()
flowedges_inlet_outlet_3 = Regular_1D_8.GetSubMesh()
flowedges_belly_3 = Regular_1D_9.GetSubMesh()
corners_2 = Regular_1D_10.GetSubMesh()

## Set names of Mesh objects
smesh.SetName(Regular_1D.GetAlgorithm(), 'Regular_1D')
smesh.SetName(Hexa_3D.GetAlgorithm(), 'Hexa_3D')
smesh.SetName(Quadrangle_2D.GetAlgorithm(), 'Quadrangle_2D')
smesh.SetName(Number_of_Segments_300, 'Number of Segments=300')
smesh.SetName(Number_of_Segments_50, 'Number of Segments=50')
smesh.SetName(inlet_1, 'inlet')
smesh.SetName(outlet_1, 'outlet')
smesh.SetName(Number_of_Segments_10, 'Number of Segments=10')
smesh.SetName(wall_1, 'wall')
smesh.SetName(Number_of_Segments_2, 'Number of Segments=2')
smesh.SetName(flowedges_inlet_outlet_2, 'flowedges_inlet_outlet')
smesh.SetName(flowedges_belly_2, 'flowedges_belly')
smesh.SetName(corners_1, 'corners')
smesh.SetName(MeshA.GetMesh(), 'MeshA')
smesh.SetName(MeshC.GetMesh(), 'MeshC')
smesh.SetName(MeshB.GetMesh(), 'MeshB')
smesh.SetName(flowedges_belly_1, 'flowedges_belly')
smesh.SetName(flowedges_inlet_outlet_1, 'flowedges_inlet_outlet')
smesh.SetName(corners_2, 'corners')
smesh.SetName(Number_of_Segments_20, 'Number of Segments=20')
smesh.SetName(flowedges_inlet_outlet_3, 'flowedges_inlet_outlet')
smesh.SetName(flowedges_belly_3, 'flowedges_belly')
smesh.SetName(outlet_3, 'outlet')
smesh.SetName(wall_3, 'wall')
smesh.SetName(inlet_3, 'inlet')
smesh.SetName(wall_2, 'wall')
smesh.SetName(outlet_2, 'outlet')
smesh.SetName(inlet_2, 'inlet')


if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser()
```

Appendix E - Importing mesh to OpenFOAM

A mesh created in Salome can be imported to OpenFOAM with the following procedure:

In the mesh module in Salome, right-click on the mesh and export it as an unv-file.

Copy the unv-file to an OpenFOAM case directory. Delete the existing polyMesh or blockMesh file. Open the terminal and import the mesh with the command:

```
ideasUnvToFoam mesh_name.unv
```

OpenFOAM uses the SI system, but the mesh made in Salome does not have a length unit. The mesh is transformed from *m* to *mm* with the *transformPoint*s command:

```
transformPoints -scale '(1e-3 1e-3 1e-3)'
```

In addition, the correct boundaries and patches must be set in the polyMesh directory.

Appendix F - Cross-sectional mesh properties

Aspect ratio

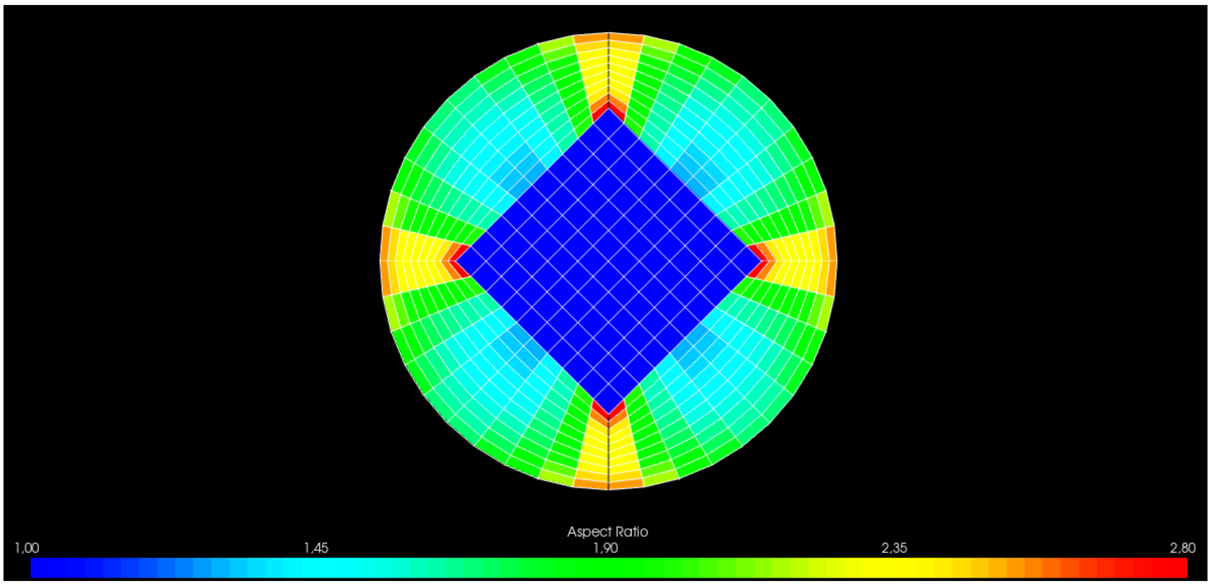


Figure F.1: Aspect ratio Mesh A.

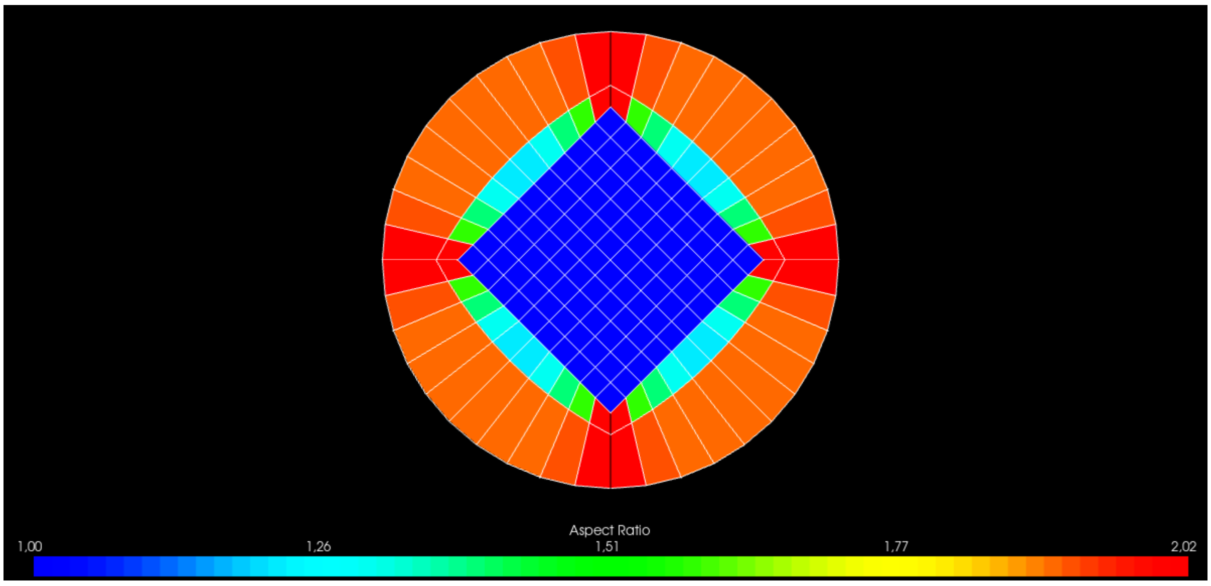


Figure F.2: Aspect ratio Mesh B.

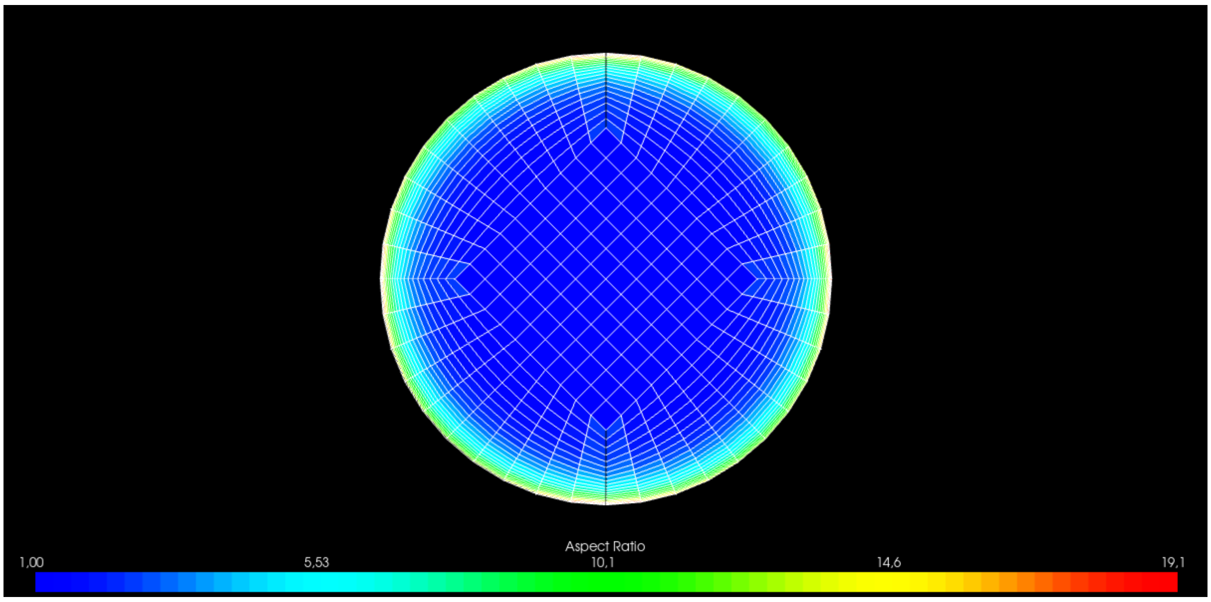


Figure F.3: Aspect ratio Mesh C.

Skewness

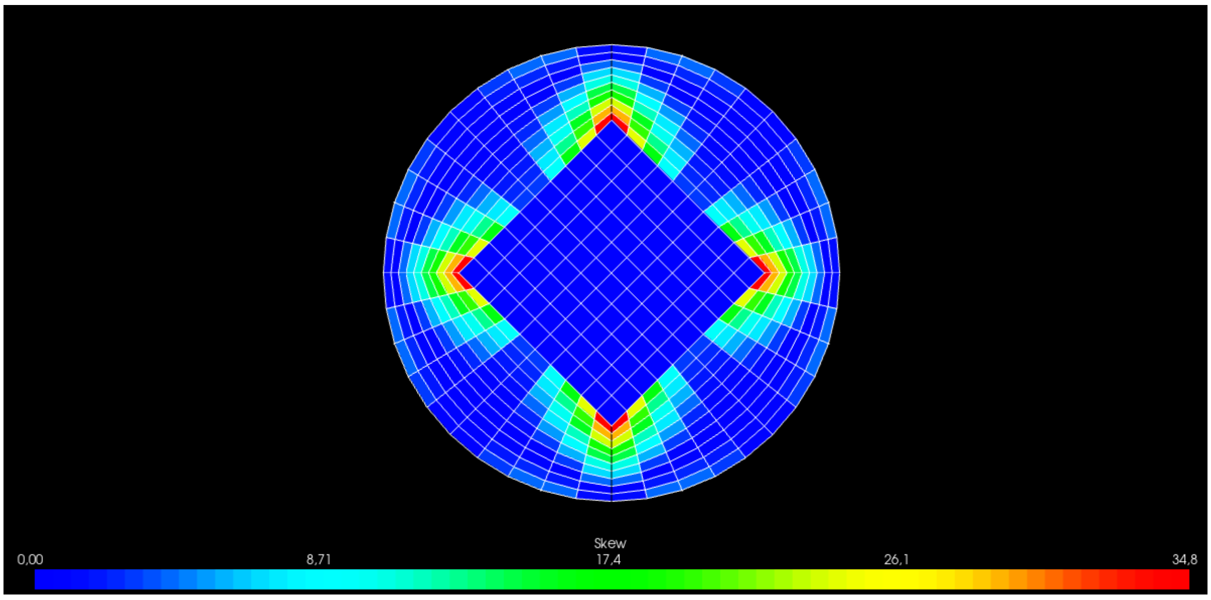


Figure F.4: Skewness Mesh A.

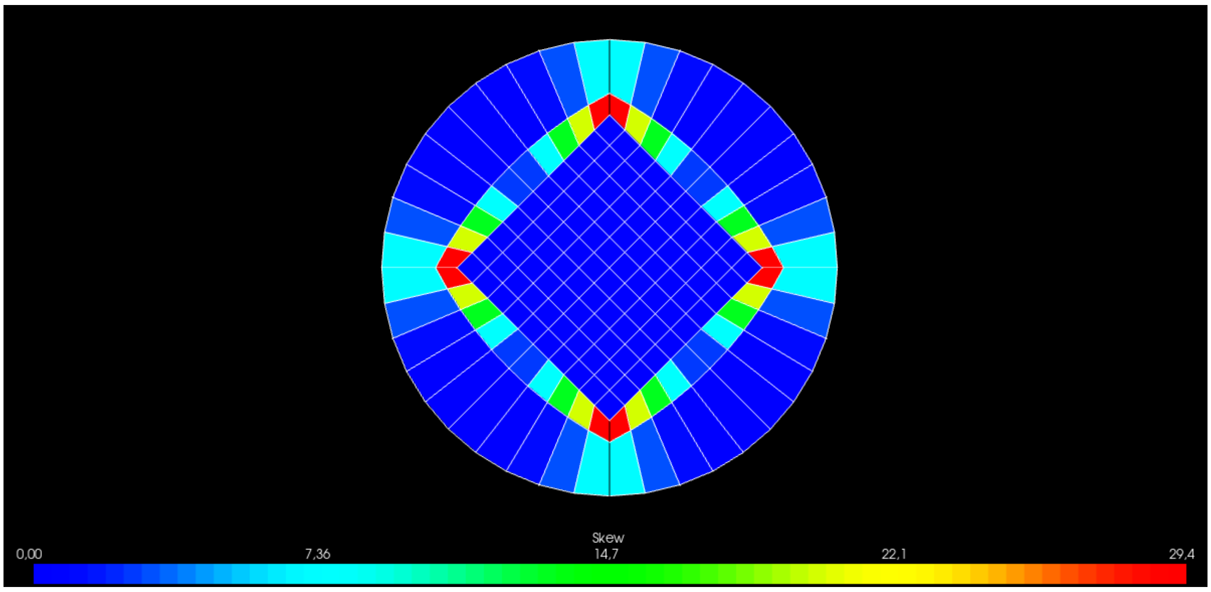


Figure F.5: Skewness Mesh B.

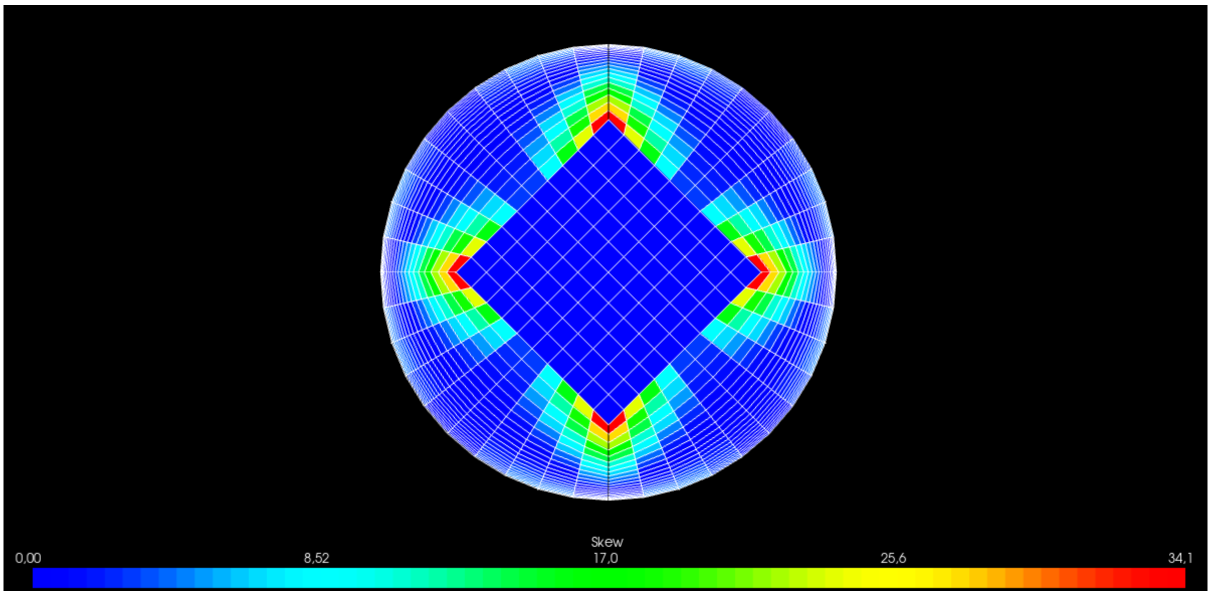


Figure F.6: Skewness Mesh C.

Appendix G - Wall functions in OpenFOAM

Available wall functions in OpenFOAMv8 are listed and described by typing the following command in the terminal:

```
foamInfo wallFunction
```

k wall functions

Table G.1: k wall functions.

Wall function	**Suitable for turbulence model**	
	HRN	**LRN**
kqRWallFunction	X	
kLowReWallFunction	X	X

ε wall functions

Table G.2: ε wall functions.

Wall function	**Suitable for turbulence model**	
	HRN	**LRN**
epsilonWallFunction	X	

ω wall functions

Table G.3: ω wall functions.

Wall function	**Suitable for turbulence model**	
	HRN	**LRN**
omegaWallFunction	X	X

μ_t wall functions

Table G.4: μ_t wall functions [1].

Wall function	**Suitable for turbulence model**		
	HRN	**LRN**	**All regions**
nutkWallFunction	X		
nutLowReWallFunction	X	X	
nutUSpaldingWallFunction			X
nutUTabulatedWallFunction			X
nutURoughWallFunction	X		
nutUWallFunction	X		
nutkRoughWallFunction	X		
nutkAtmRoughWallFunction	X		

References

[1] S. N. Liu, "Implementation of a Complete Wall Function for the Standard k−epsilon Turbulence Model in OpenFOAM 4.0," in "In Proceedings of CFD with OpenSource Software," edited by: H. Nilsson and M. Arabnejad, University of Stavanger, Stavanger, Project Work, 2016. Accessed on: Apr. 4, 2021. [Online]. Available: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016/ShengnanLiu/FinalReport-Shengnan.pdf

Appendix H - Constants and boundary conditions

Table H.1: Air properties [1, p. 733].

$\boldsymbol{T}$ [℃]	$\boldsymbol{\nu}\ [\boldsymbol{m^2/s}]$	$\boldsymbol{\rho}\ [\boldsymbol{kg/m^3}]$	$\boldsymbol{\mu}\ [\boldsymbol{Ns/m^2}]$
20	$1.5 \cdot 10^{-5}$	1.205	$1.81 \cdot 10^{-5}$

Table H.2: Constants and calculated boundary conditions at inlet.

$\boldsymbol{D}$ **[m]**	0.02
$\boldsymbol{l}$ **[m]**	0.0014
$\boldsymbol{U}$ **[m/s]**	3
$\boldsymbol{Re}\ [-]$	4000
$\boldsymbol{T_i}\ [-]$	0.057
$\boldsymbol{k}\ [\boldsymbol{m^2/s^2}]$	0.043457
$\boldsymbol{\varepsilon}\ [\boldsymbol{m^2/s^3}]$	1.06327
$\boldsymbol{\omega}\ [\boldsymbol{s^{-1}}]$	271.857

Table H.3: Initial and boundary conditions U and p for all cases.

Type	$\boldsymbol{U}\ [\boldsymbol{m/s}]$	$\boldsymbol{p}\ [\boldsymbol{m^2/s^2}]$
internalField	uniform (0 0 0)	uniform 0
inlet	fixedValue uniform (3 0 0)	zeroGradient
outlet	zeroGradient	fixedValue uniform 0
wall	noSlip	zeroGradient

Table H.4: Initial and boundary conditions Case A and Case B1.

$\boldsymbol{Type}$	$\boldsymbol{k}\ [\boldsymbol{m^2/s^2}]$	$\boldsymbol{epsilon}\ [\boldsymbol{m^2/s^3}]$	$\boldsymbol{nut}\ [\boldsymbol{m^2/s}]$
internalField	uniform 0.043457	uniform 1.06327	uniform 0
inlet	fixedValue uniform 0.043457	fixedValue uniform 1.06327	calculated uniform 0
outlet	zeroGradient	zeroGradient	calculated uniform 0
wall	kqRWallFunction uniform 0.043457	epsilonWallFunction uniform 1.06327	nutkWallFunction uniform 0

Table H.5: Initial and boundary conditions Case B2.

$\boldsymbol{Type}$	$\boldsymbol{k}\ [\boldsymbol{m^2/s^2}]$	$\boldsymbol{epsilon}\ [\boldsymbol{m^2/s^3}]$	$\boldsymbol{nut}\ [\boldsymbol{m^2/s}]$
internalField	uniform 0.043457	uniform 1.06327	uniform 0
inlet	fixedValue uniform 0.043457	fixedValue uniform 1.06327	calculated uniform 0
outlet	zeroGradient	zeroGradient	calculated uniform 0
wall	kLowReWallFunction uniform 0.043457	epsilonWallFunction uniform 1.06327	nutUSpaldingWallFunction uniform 0

Table H.6: Initial and boundary conditions Case C.

$\boldsymbol{Type}$	$\boldsymbol{k}\ [\boldsymbol{m^2/s^2}]$	$\boldsymbol{omega}\ [\boldsymbol{s^{-1}}]$	$\boldsymbol{nut}\ [\boldsymbol{m^2/s}]$
internalField	uniform 0.043457	uniform 271.857	uniform 0
inlet	fixedValue uniform 0.043457	fixedValue uniform 271.857	calculated uniform 0
outlet	zeroGradient	zeroGradient	calculated uniform 0
wall	fixedValue uniform 0	omegaWallFunction uniform 271.857	nutLowReWallFunction uniform 0

References

[1] J. B. Franzini and E. J. Finnemore, *Fluid Mechanics with Engineering Applications*, 10 ed. New York: McGraw-Hill, 2002, p. 733.

Appendix I - Symmetry planes

Velocity

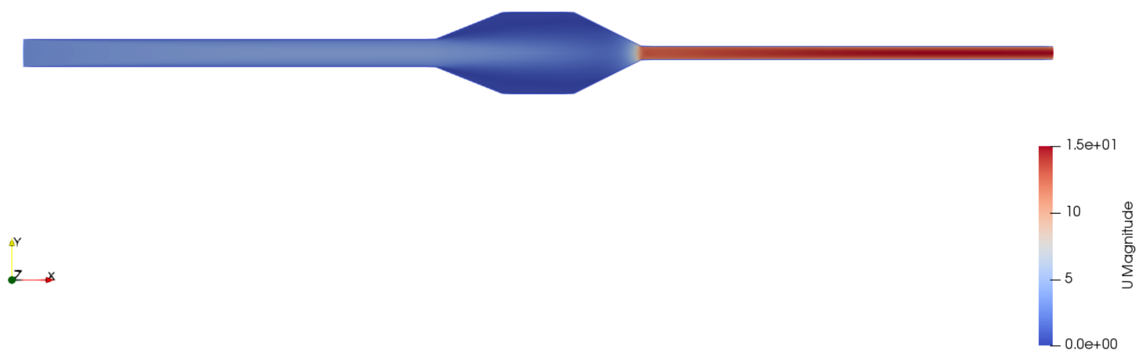


Figure I.1: Velocity Case A.

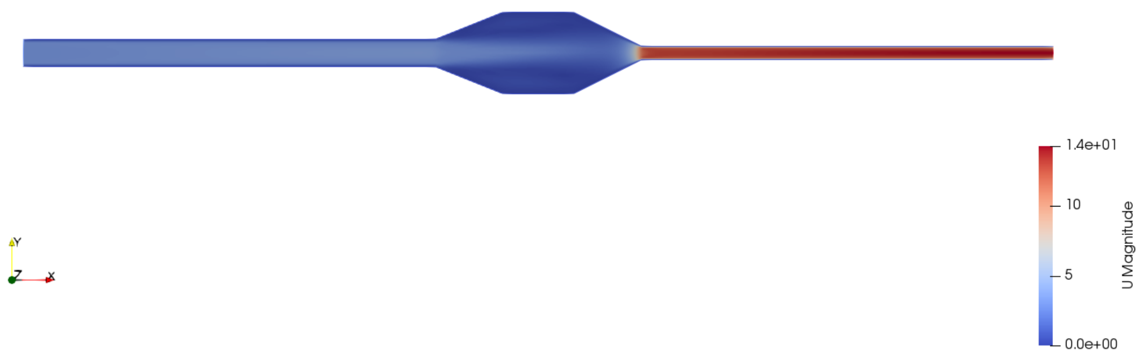


Figure I.2: Velocity Case B1.

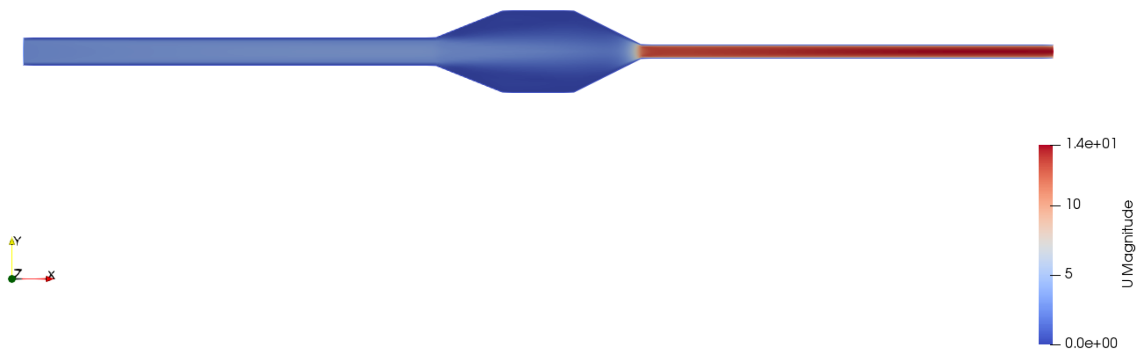


Figure I.3: Velocity Case B2.

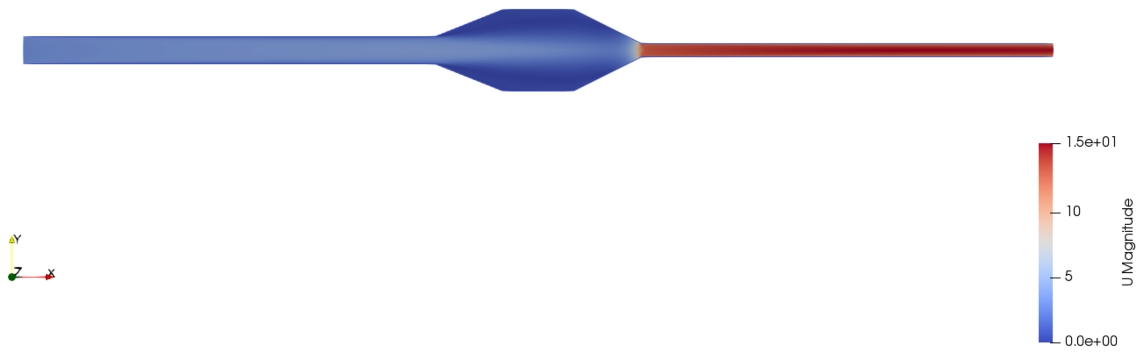


Figure I.4: Velocity Case C.

Stagnation pressure

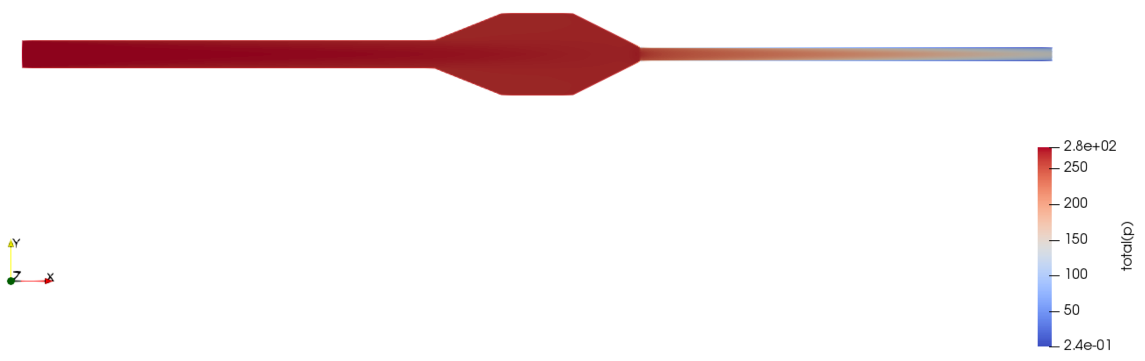


Figure I.5: Stagnation pressure Case A.

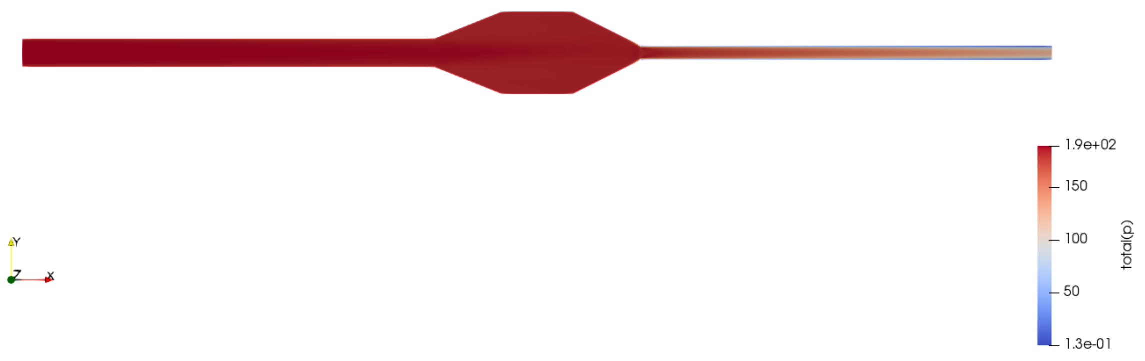


Figure I.6: Stagnation pressure Case B1.

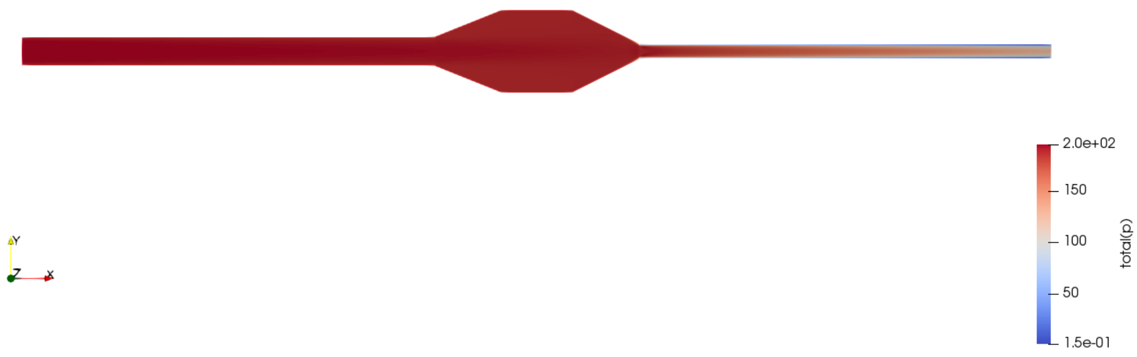


Figure I.7: Stagnation pressure Case B2.

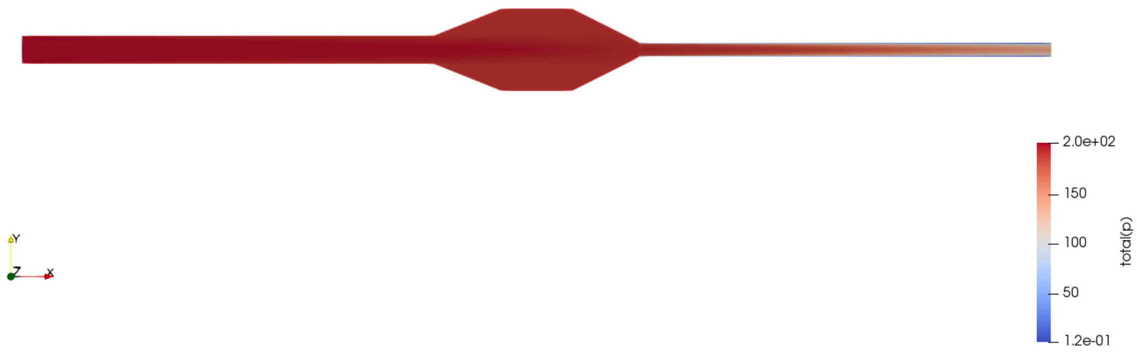


Figure I.8: Stagnation pressure Case C.

Static pressure

The static pressure can be calculated in OpenFOAM with the same post-processing procedure as in Appendix L for total pressure. Just change all the occurrences of *totalPressureIncompressible* to *staticPressure*, and *total(p)* to *static(p)*, and the correct density will be used in the calculations.

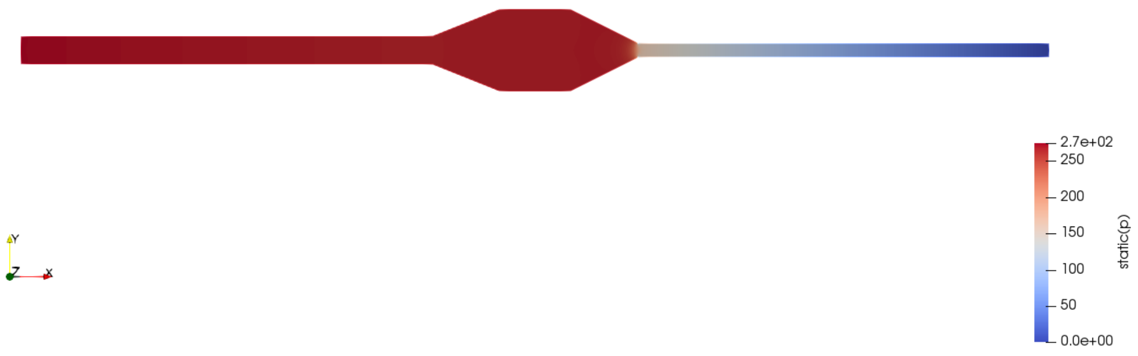


Figure I.9: Static pressure Case A.

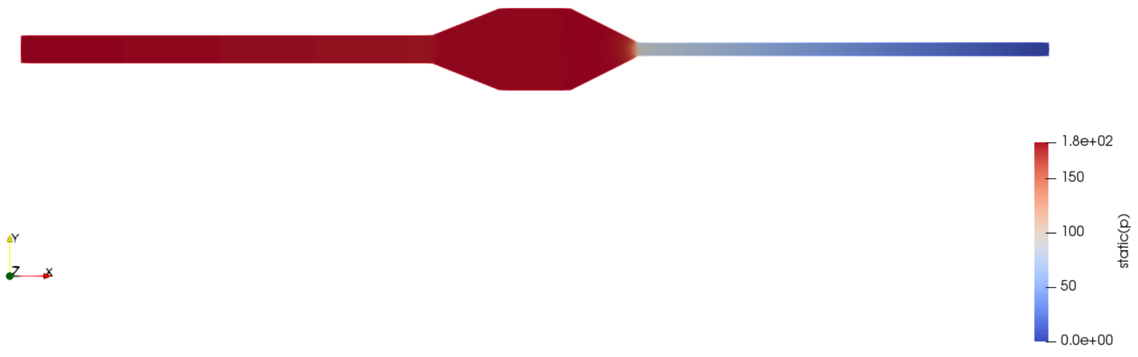


Figure I.10: Static pressure Case B1.

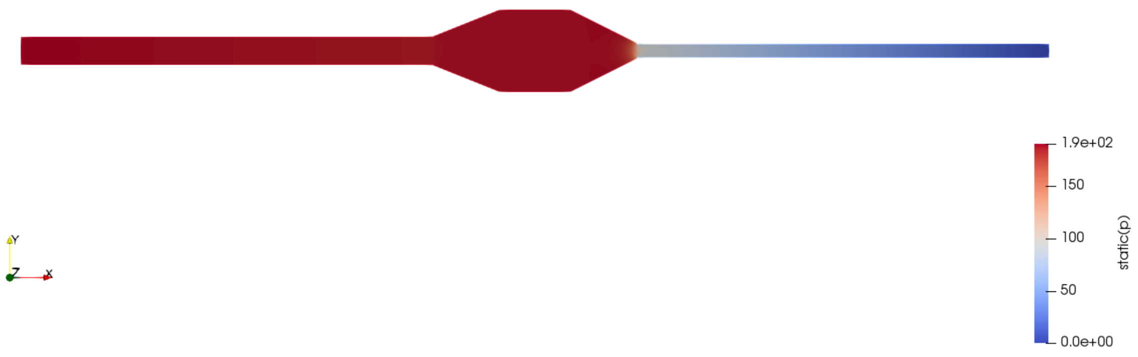


Figure I.11: Static pressure Case B2.

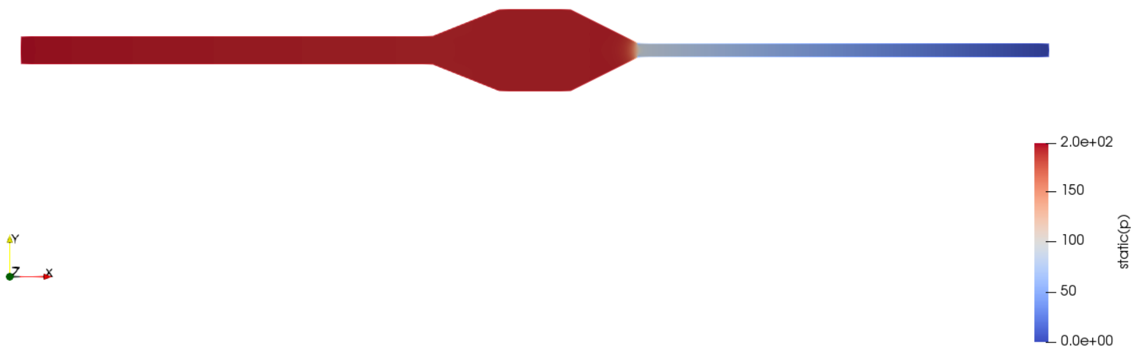


Figure I.12: Static pressure Case C.

Appendix J - Residual plots

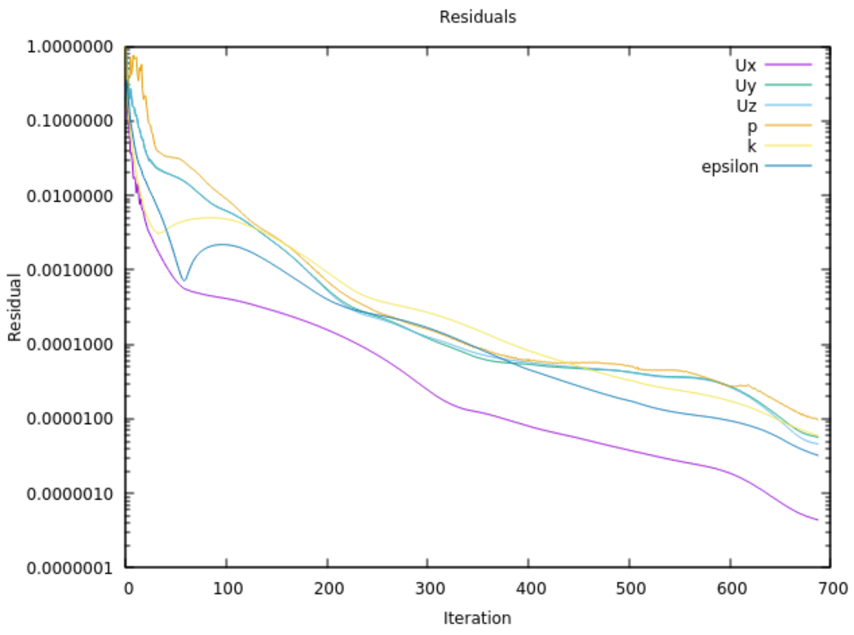


Figure J.1: Residuals Case A.

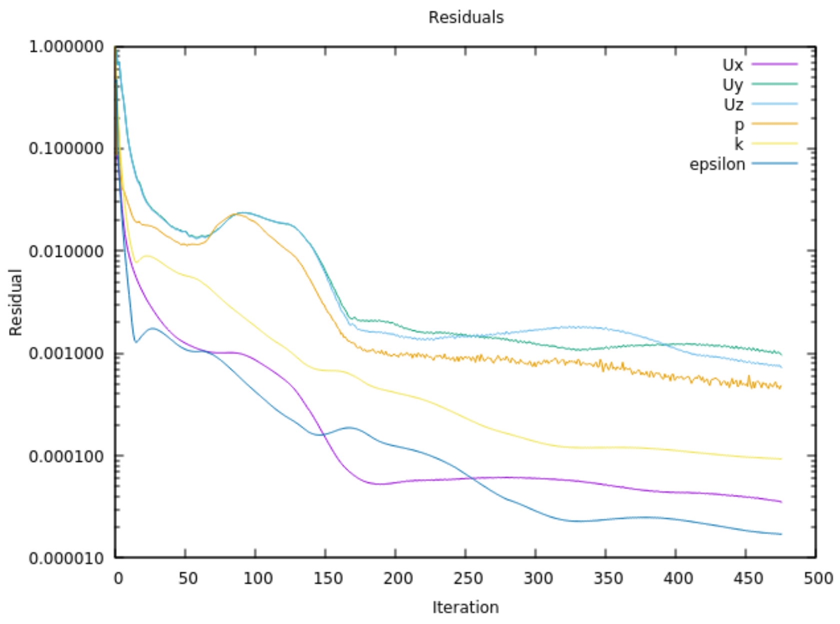


Figure J.2: Residuals Case B1.

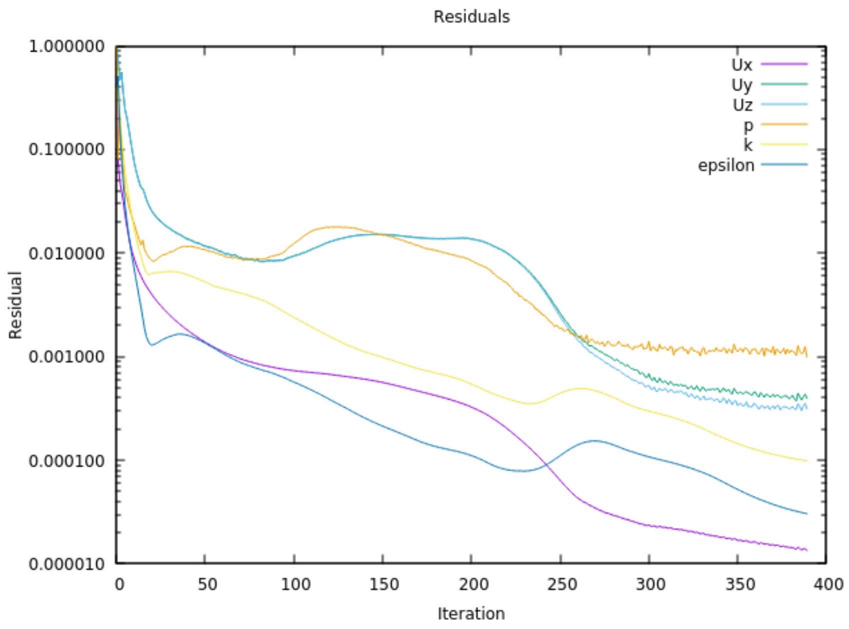


Figure J.3: Residuals Case B2.

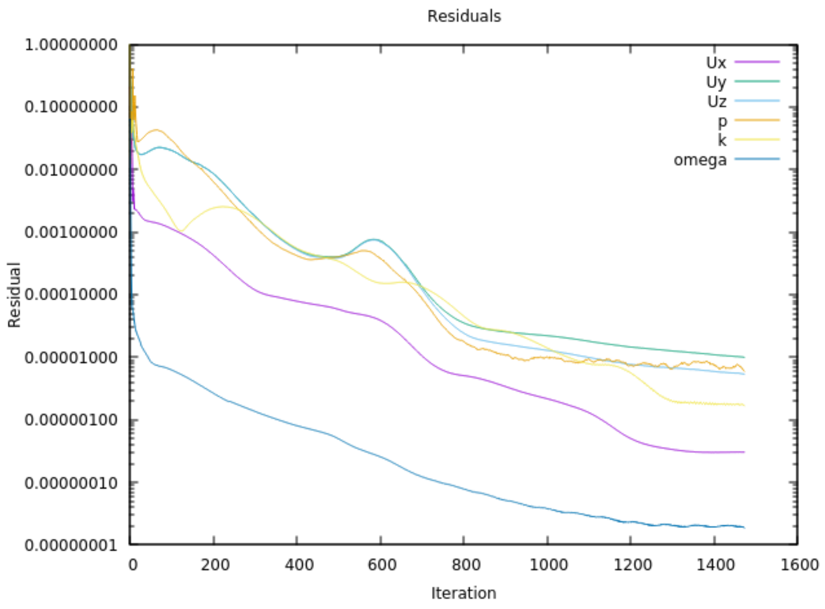


Figure J.4: Residuals Case C.

Appendix K - Visualization of y^+ distribution

Visualization of the y^+ field in ParaView of the four simulations.

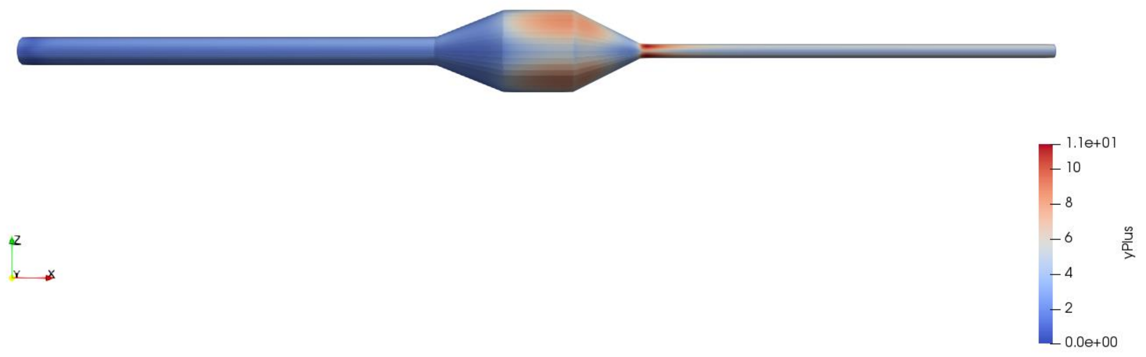


Figure K.1: Case A.

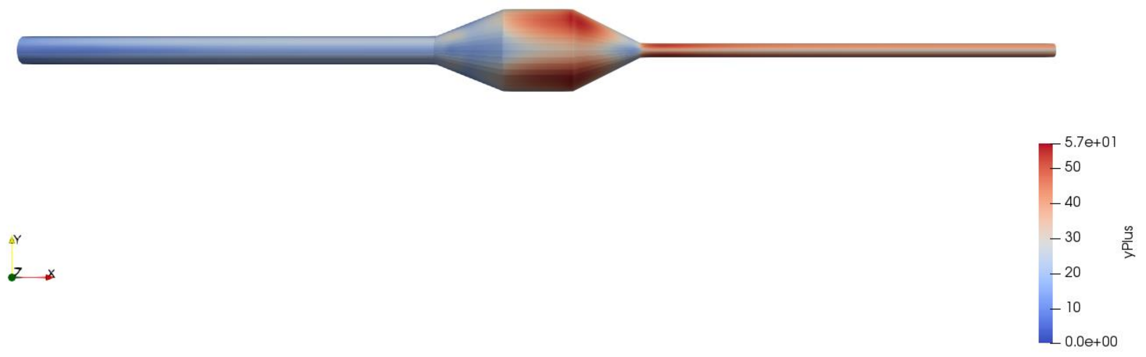


Figure K.2: Case B1.

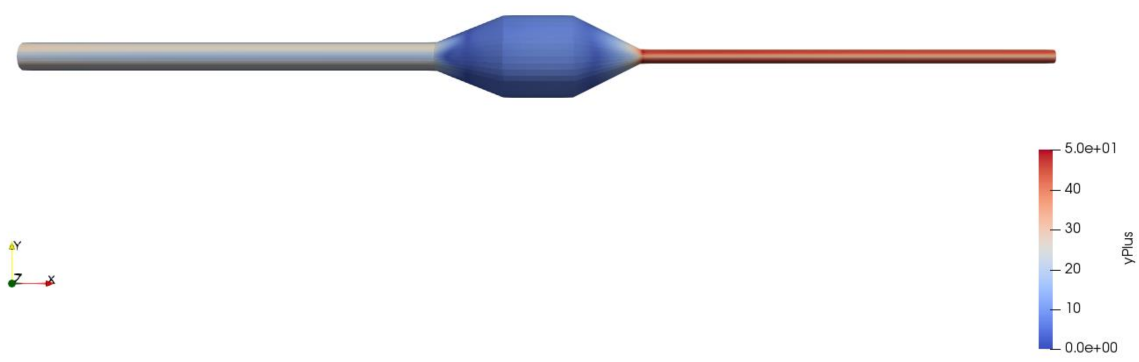


Figure K.3: Case B2.

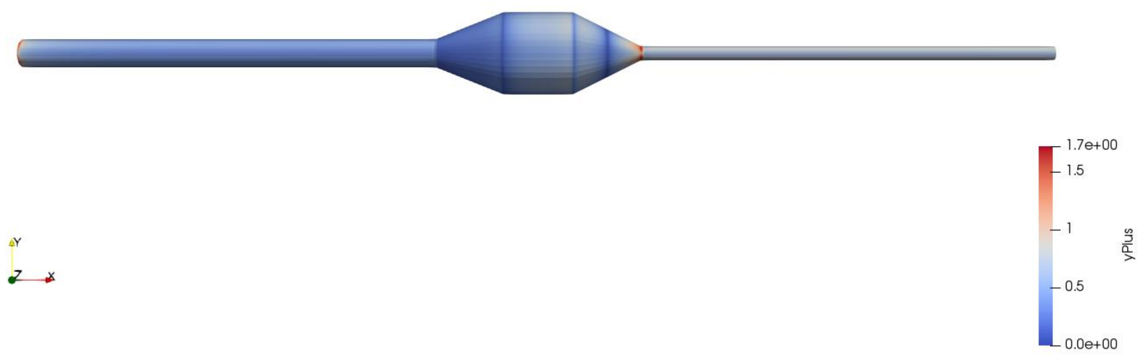


Figure K.4: Case C.

Appendix L - Pressure loss calculation

Calculating total pressure in OpenFOAM

The default density for incompressible solvers in OpenFOAM is 1.2. In order for OpenFOAM to calculate the total pressure with a chosen density, the density must be provided in an additional file.

The density for calculating the stagnation pressure in OpenFOAM is added by creating an additional file named *totalPressureIncompressible1* in the system directory containing the correct density:

```
#includeEtc "caseDicts/postProcessing/pressure/totalPressureIncompressible.cfg"

      pRef    0.0;
      rhoInf  1.205; //1.2;
```

The function must be added in the controlDict dictionary with the following lines:

```
      functions
      {
      #includeFunc totalPressureIncompressible1
      }
```

The total pressure for each time step is then calculated by typing in the terminal:

```
simpleFoam -postProcess -fields "(p U)" -func totalPressureIncompressible1
```

The average values of the inlet and outlet patches can be calculated by the command:

```
postProcess -func 'patchAverage(name=inlet,p,U,total(p))' -latestTime
```

and

```
postProcess -func 'patchAverage(name=outlet,p,U,total(p))' -latestTime
```

The total pressure for all fields can be viewed in ParaView. To find the value of the average pressure over an area:

- make a *slice* on the X Normal on the desired X coordinate.
- *Integrate variables*. Attruibute *Cell Data*. To get the total pressure: manually divide the result of the integrated variable by the area, or use the calculator filter.
- *(*Alternatively: *Export data to spreadsheet*. Several CSV-files can be imported to Excel in one batch and divided by Area in Excel for simplicity and data storage)

Stagnation pressure loss calculations

Pipe section	Inlet	Diverging	Belly	Converging	Outlet
$L\ [mm]$	300	50	50	50	300
$D\ [mm]$	20	*	60	*	10
$A\ [m^2]$	3.14E-04	*	2.83E-03	*	7.85E-05
$Q\ [m^3/s]$	9.42E-04	9.42E-04	9.42E-04	9.42E-04	9.42E-04
$U_{calculated}$	3	*	0.33	*	12
Re	4000	*	1333	*	8000
f_{turb}	0.040	*	-	*	0.033
f_{lam}	-	*	0.048	*	-
K	-	0.438[1]	-	1[2]	-
$p_{loss}\ [Pa]$[3]	3.24	1.06	0.003	96.76	87.19

[1] Conical enlargement 22°

[2] Sudden contraction 0.03 → tank exit 0.5 + 0.5 extra

[3] The sum of the pressure losses is 188.25 Pa

* Calculated at each point

Appendix M - Wall treatment procedures

Mesh B

The procedure for creating Mesh B is explained in the student tutorial in Appendix N.

Mesh A

To create Mesh A the procedure is the same as in the student tutorial, but the step *Sub-mesh for edges* must be omitted.

Mesh C

To create Mesh C, the procedure is the same as in the student tutorial, but the step *Sub-mesh for edges* must be replaced with the following sub-mesh:

Right-click on *MeshC* and click *Create Sub-mesh.*

Name	edges		
Mesh	MeshC		
Geometry	edges		
Algorithm	Wire Discretisation		
Hypothesis	Number of Segments	**Name**	Number of Segments=20
		Number of Segments	20
		Type of distribution	Scale distribution
		Scale factor	10
		Helper	Check the box: *Propagation chains.* Mark *Chain 1 (24 edges)* and click *Add.*

Appendix N -Student tutorial

Contents

Introduction

This tutorial describes how to use the meshing software Salome. We will create and mesh a 3D model of a diverging converging pipe with belly section with a hexahedral OH-grid. We will import the mesh to OpenFOAM and run the **simpleFoam** solver, applying the turbulence model **standard $\mathbf{k-\varepsilon}$** with wall functions.

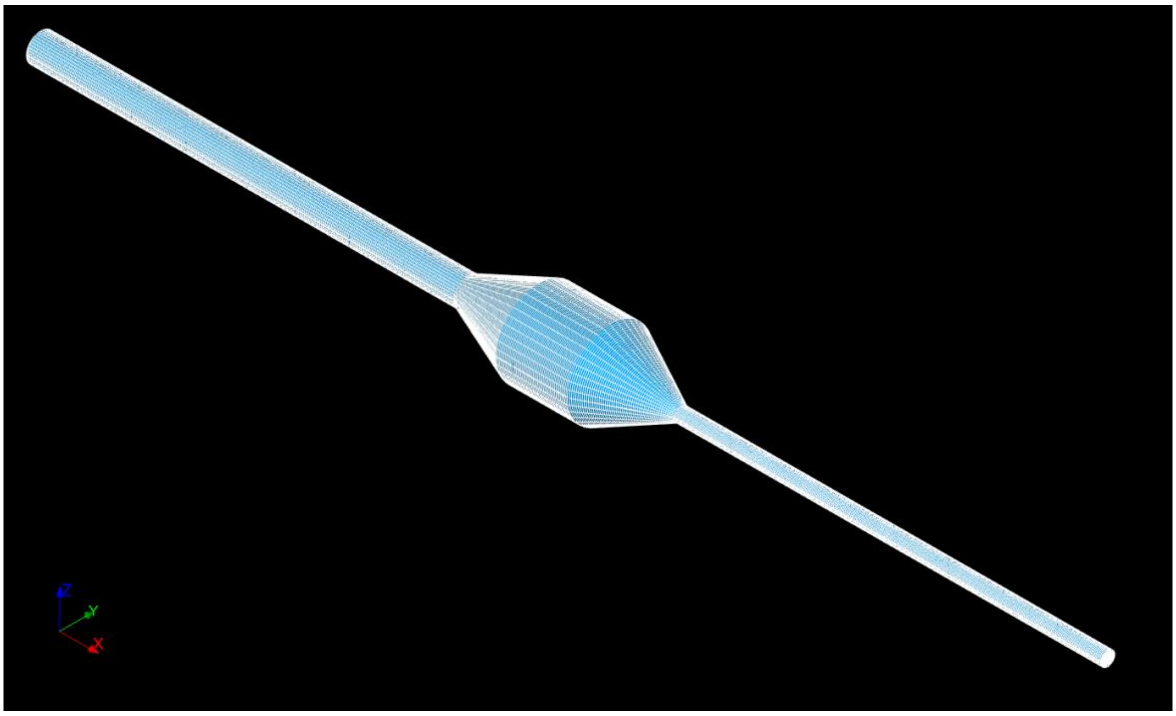


Figure N.1: Diverging converging pipe with belly section.

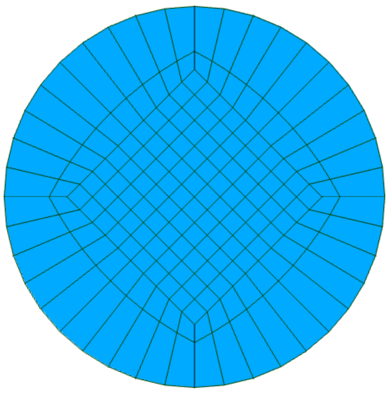


Figure N.2: Cross-section of pipe.

Workflow:

- Install the meshing software Salome
- Draw a diverging converging pipe with belly section in 3D
- Create boundary groups
- Generate mesh
- Improve mesh for better mesh quality and optimality
- Export mesh
- Import mesh to OpenFOAM
- Set appropriate boundary conditions and patches
- Run a simulation
- Post-process

The aim of this tutorial is to show the students a complete workflow from creating and meshing a complex geometry in Salome to importing it to OpenFOAM, running a simulation and post-processing. The intention is that the students can use what they have learned in this tutorial to create their own geometries and meshes for more applied cases.

Installation of Salome in Ubuntu

First, some additional packages in Ubuntu need to be installed. Open the terminal and run the command:

```
sudo apt install net-tools
```

and:

```
sudo apt install libopengl0 -y
```

To download Salome, go to the following website:

https://www.salome-platform.org/downloads/current-version

Download:

Universal Linux binary

Move the downloaded file to the directory you want Salome to be located in, e.g., Home. Open the terminal window from the directory. In the terminal, list the contents in the directory by typing:

```
ls
```

Copy the name of the Salome tar-file (e.g., ***SALOME-x.x.x.tar.gz***). Then paste the filename after the untar-command:

```
tar -zxvf SALOME-x.x.x.tar.gz
```

The output of the untar-process is seen in the terminal. Go to the untared directory by typing:

```
cd SALOME-x.x.x
```

To run Salome, type:

```
./salome.
```

Installation problems

If there are problems opening Salome, try updating and upgrading the ubuntu software:

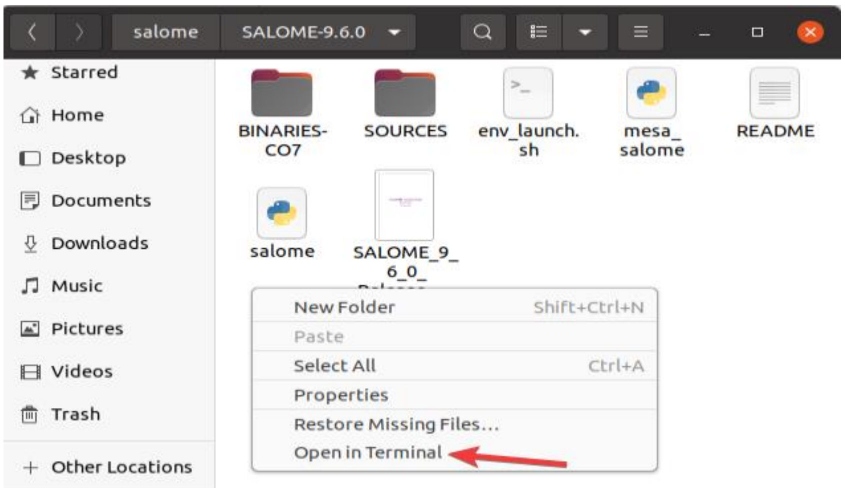
```
sudo apt-get update
```

and:

```
sudo apt-get upgrade
```

How to use Salome

Open the **Salome** directory, right-click and click *Open in Terminal*.



Launch Salome by typing in the terminal window:

```
./salome
```

Tips & tricks in Salome

Save as

Save the hdf-file with the *Save as* command many times during the process. Salome might crash, or you might have made the wrong input. If you have saved previous versions of the file, you can go back, and do not need to start over again.

Marking multiple objects in the object browser:

- Hold down Shift-key to mark objects adjacent to each other.
- Hold down Ctrl-key to mark objects not adjacent to each other.

Eye symbol

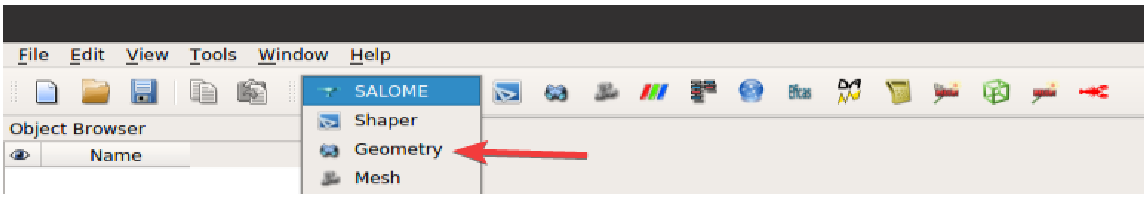
Click on the eye symbol in the *Object Browser* to show or hide an object.

Show only

If you do not know where an object listed in the object browser is located on the geometry, right-click on the object in the object tree and choose *Show Only*.

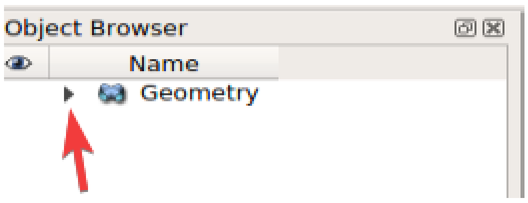
Geometry module in Salome

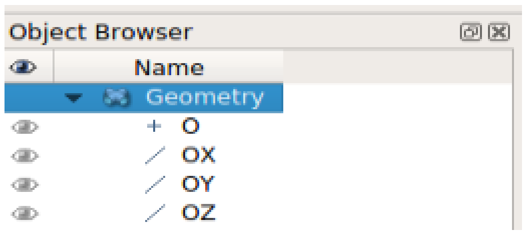
Go to the scrolldown-menu and click *Geometry* to open the geometry module.



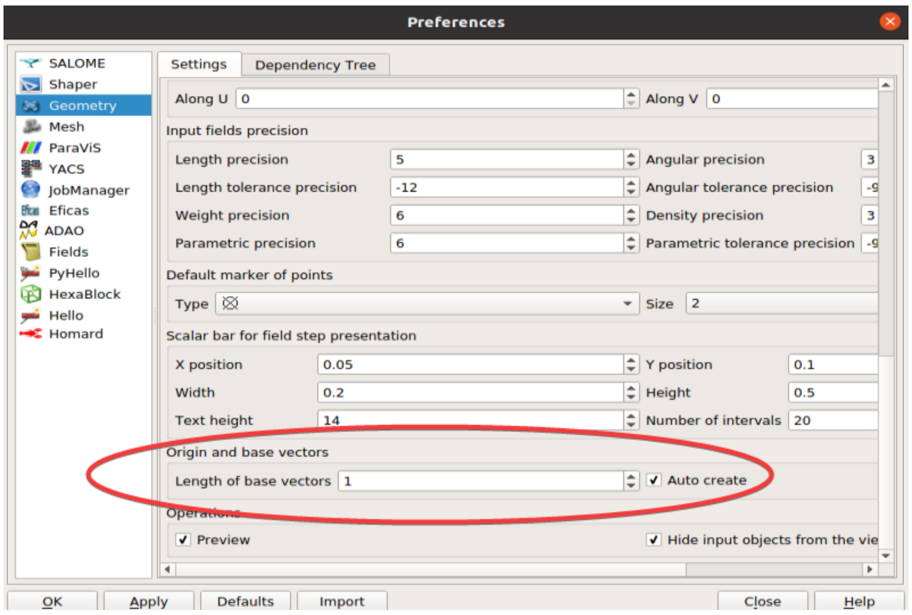
Check that vectors are present in the object browser.

Click the arrow in the object browser in front of *geometry*.





If the vectors *OX*, *OY* and *OZ* do not appear, open settings with command Ctrl + P, and check the box *Auto create* under the heading *Origin and base vectors*.



Along U 0 | Along V 0

Input fields precision

Length precision	5	Angular precision	3
Length tolerance precision	-12	Angular tolerance precision	-9
Weight precision	6	Density precision	3
Parametric precision	6	Parametric tolerance precision	-9

Default marker of points

Type ⊠ | Size 2

Scalar bar for field step presentation

X position	0.05	Y position	0.1
Width	0.2	Height	0.5
Text height	14	Number of intervals	20

Origin and base vectors

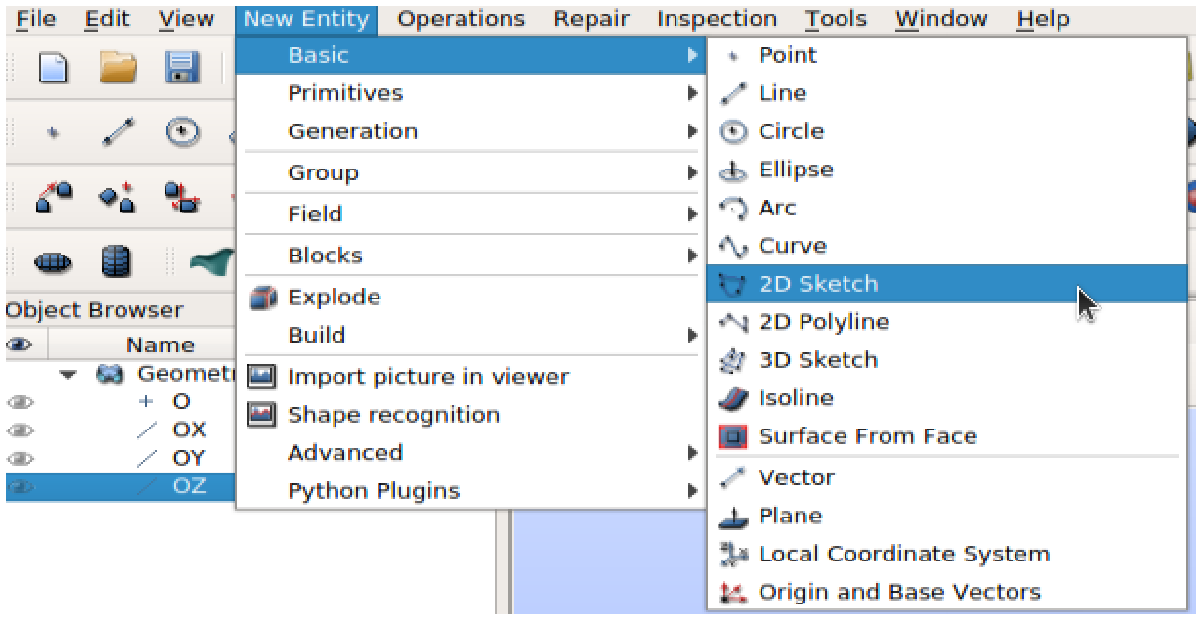
Length of base vectors 1 | ✓ Auto create

Operations

✓ Preview | ✓ Hide input objects from the vie

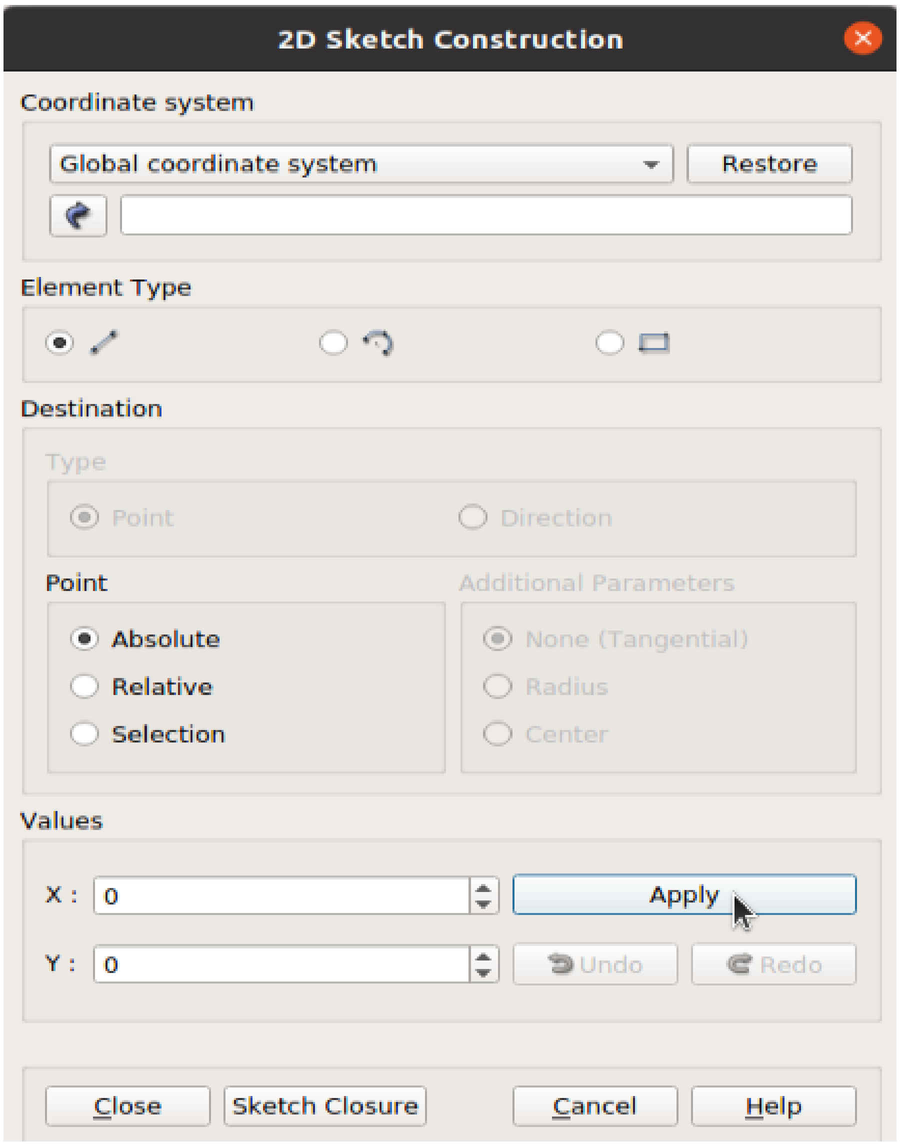
Create 2D sketch

Go to *New Entity -> Basic -> 2D Sketch*



Make a 2D sketch with the following coordinates. Click *Apply* after adding each point. Close.

X	**Y**
0	0
300	0
350	0
400	0
450	0
750	0


2D Sketch Construction

Coordinate system

Global coordinate system

Restore

Element Type

Destination

Type

Point

Direction

Point

Absolute

Relative

Selection

Additional Parameters

None (Tangential)

Radius

Center

Values

X : 0

Y : 0

Apply

Undo

Redo

Close

Sketch Closure

Cancel

Help

Explode sketch into edges and vertices

Go to *New Entity -> Explode*

Main Object: Sketch_1

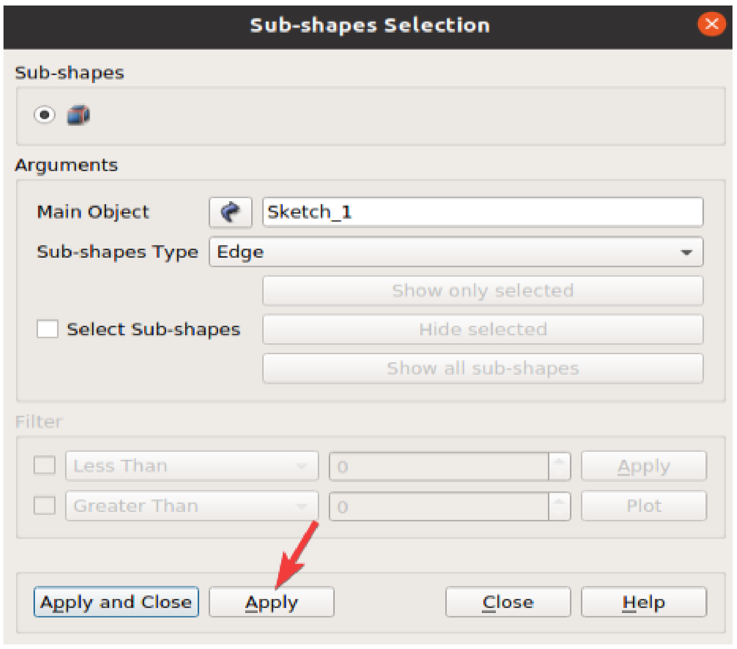
Sub-shapes Type: Edge

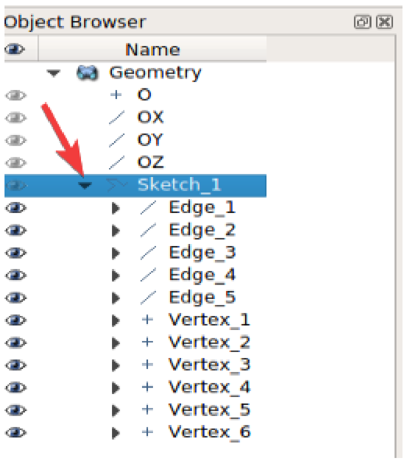
Click *Apply*

Change *Sub-shapes Type* to *Vertex*

Click *Apply and Close*

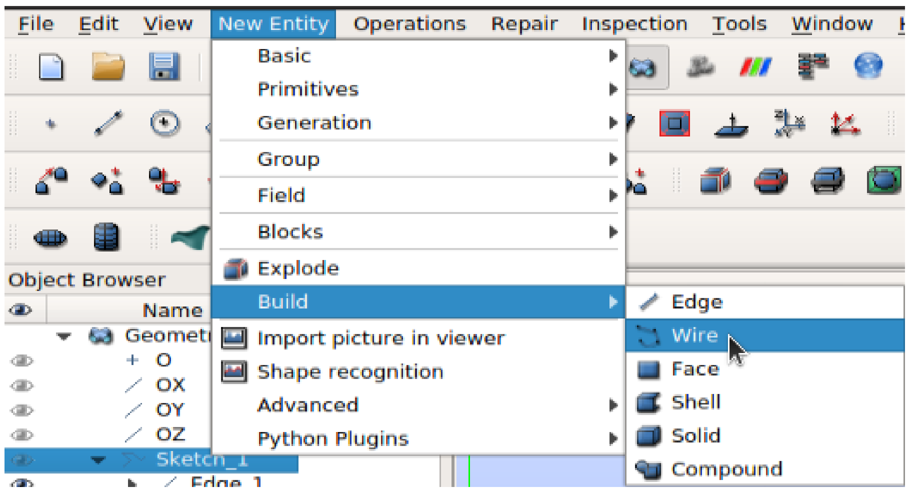
Click on the arrow next to *Sketch_1* in the *Object Browser* to see the created (exploded) edges and vertices. There should be 5 edges and 6 vertices.





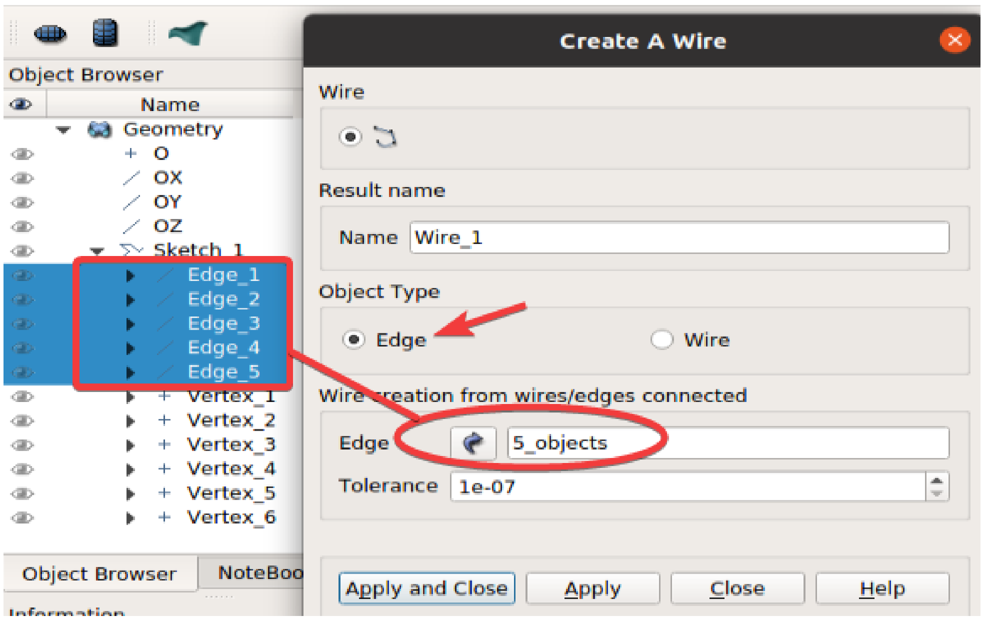
Build wire

Go to *New Entity -> Build -> Wire*



Select *Object Type* as *Edge.*

Click the blue arrow next to *Edge* and select all the five edges in *Sketch_1* by holding down the Shift-key. Click *Apply and Close.*



A wire is created based on all the edges.

Create divided disks

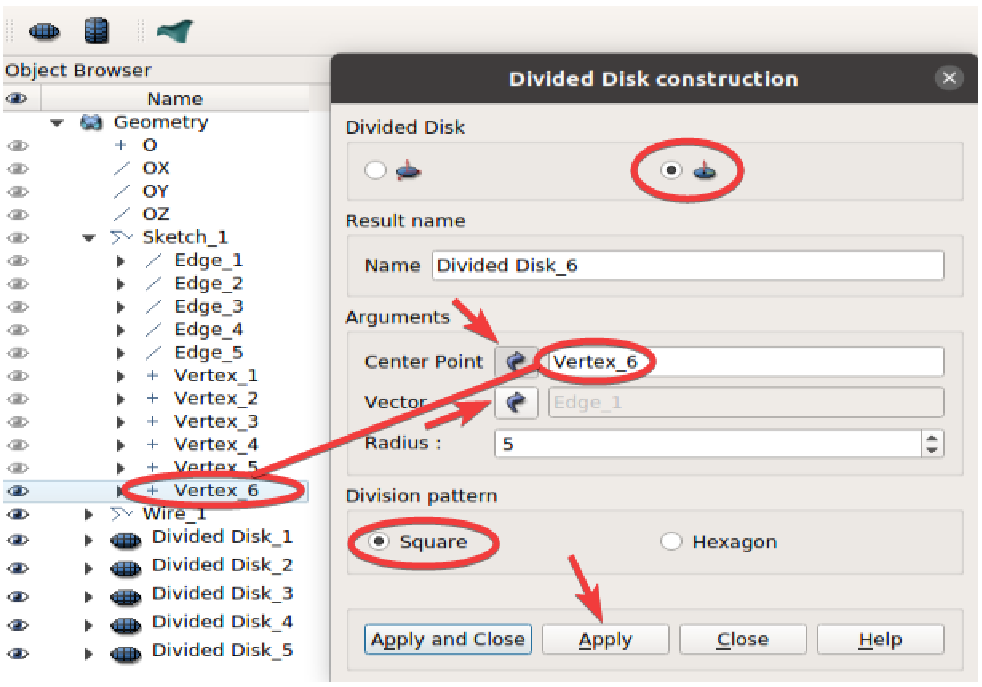
Go to *New Entity -> Blocks -> Divided Disk*

Go to the second constructor.

Division pattern: *Square*

Fill in the following *Arguments* by pushing the blue arrow to mark the argument in the *Object Browser*. Click *Apply* for each divided disk.

Name	**Center Point**	**Vector**	**Radius**
Divided Disk_1	Vertex_1	Edge_1	10
Divided Disk_2	Vertex_2	Edge_1	10
Divided Disk_3	Vertex_3	Edge_1	30
Divided Disk_4	Vertex_4	Edge_1	30
Divided Disk_5	Vertex_5	Edge_1	5
Divided Disk_6	Vertex_6	Edge_1	5



Name: Divided Disk_6

Center Point: Vertex_6

Vector: Edge_1

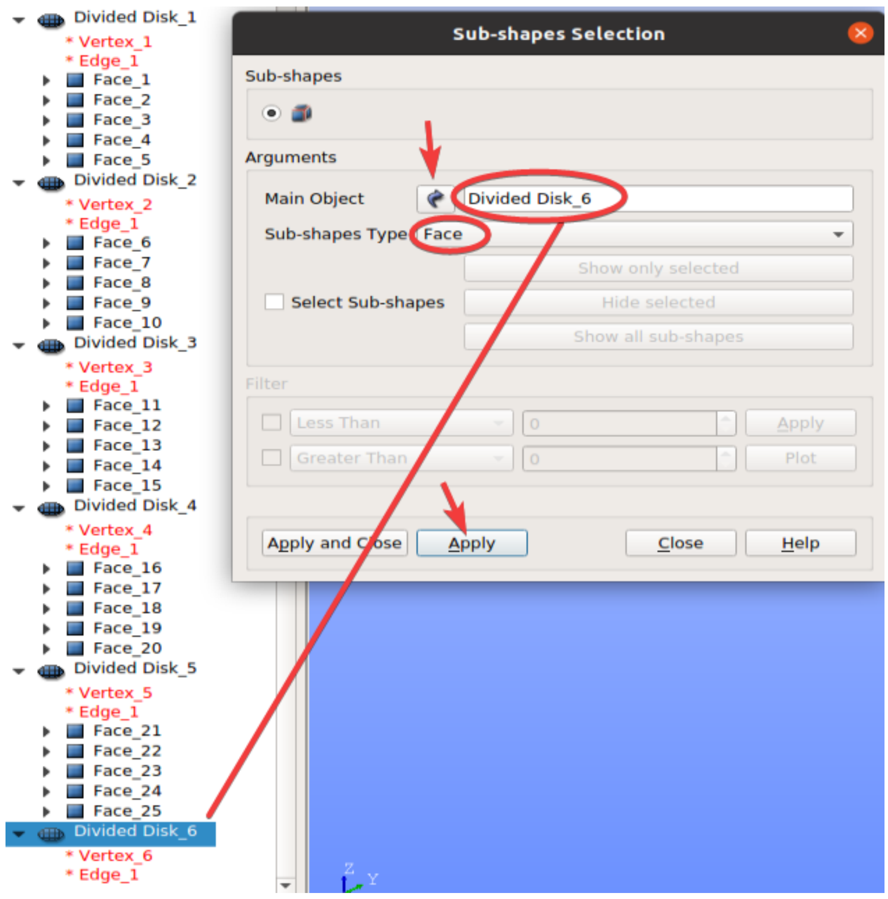
Radius : 5

[✓ Square] Hexagon

Explode disks

Go to *New Entity -> Explode*. Each disk should be divided into faces, as displayed in the table below. Click *Apply* for each divided disk. Each divided disk will be exploded into 5 faces.

Name	**Sub-shapes Type**
Divided Disk_1	Face
Divided Disk_2	Face
Divided Disk_3	Face
Divided Disk_4	Face
Divided Disk_5	Face
Divided Disk_6	Face



Divided Disk_1
Vertex_1
Edge_1
Face_1
Face_2
Face_3
Face_4
Face_5
Divided Disk_2
Vertex_2
Edge_1
Face_6
Face_7
Face_8
Face_9
Face_10
Divided Disk_3
Vertex_3
Edge_1
Face_11
Face_12
Face_13
Face_14
Face_15
Divided Disk_4
Vertex_4
Edge_1
Face_16
Face_17
Face_18
Face_19
Face_20
Divided Disk_5
Vertex_5
Edge_1
Face_21
Face_22
Face_23
Face_24
Face_25
Divided Disk_6
Vertex_6
Edge_1

Sub-shapes Selection

Sub-shapes

Arguments

Main Object: Divided Disk_6

Sub-shapes Type: Face

Select Sub-shapes

Show only selected

Hide selected

Show all sub-shapes

Filter

Less Than 0 Apply

Greater Than 0 Plot

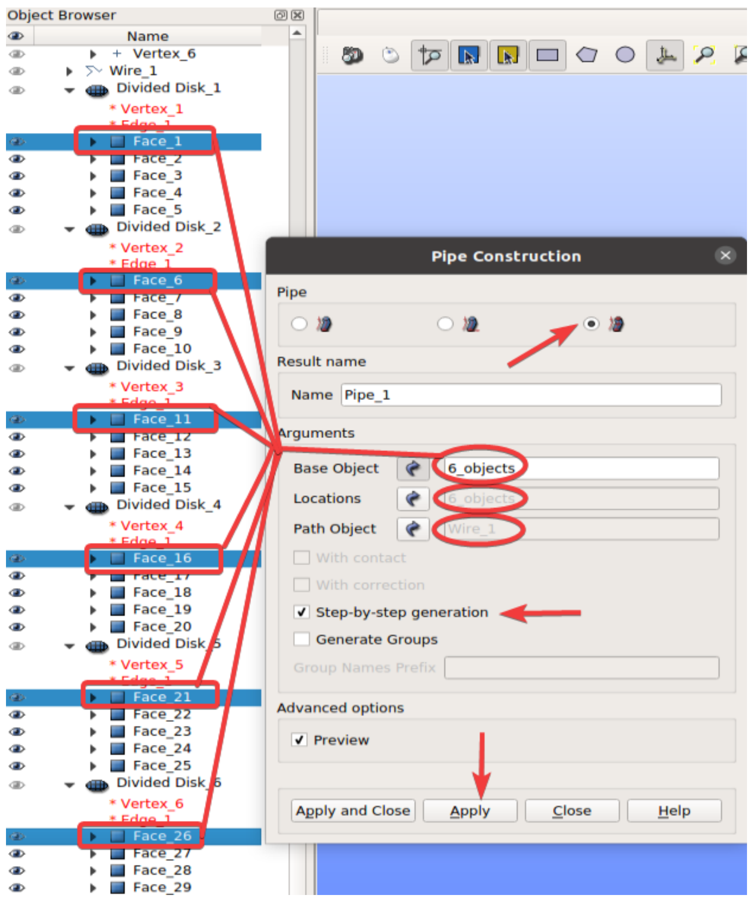
Apply and Close | Apply | Close | Help

Generate pipe with different sections

Go to *New Entity* -> *Generation* -> *Extrusion Along Path*

Go to the third pipe constructor. Choose *Arguments* by clicking the blue arrow, holding down the Ctrl-key and marking the *Base Objects* in the *Object Browser*. Hold down the Shift-key to mark the *Locations*. Do this for each argument. Choose *Step-by-step generation*. Click *Apply* after adding each pipe. 5 pipe sections will be created.

Name	**Base Object**	**Locations**	**Path Object**
Pipe_1	Face_1, Face_6, Face_11, Face_16, Face_21, Face_26	Vertex_1, Vertex_2, Vertex_3, Vertex_4, Vertex_5, Vertex_6 (6_objects)	Wire_1
Pipe_2	Face_2, Face_7, Face_12, Face_17, Face_22, Face_27		
Pipe_3	Face_3, Face_8, Face_13, Face_18, Face_23, Face_28		
Pipe_4	Face_4, Face_9, Face_14, Face_19, Face_24, Face_29		
Pipe_5	Face_5, Face_10, Face_15, Face_20, Face_25, Face_30		



Pipe

Result name

Name: Pipe_1

Arguments

Base Object: 6_objects

Locations: 6_objects

Path Object: Wire_1

☐ With contact

☐ With correction

☑ Step-by-step generation

☐ Generate Groups

Group Names Prefix

Advanced options

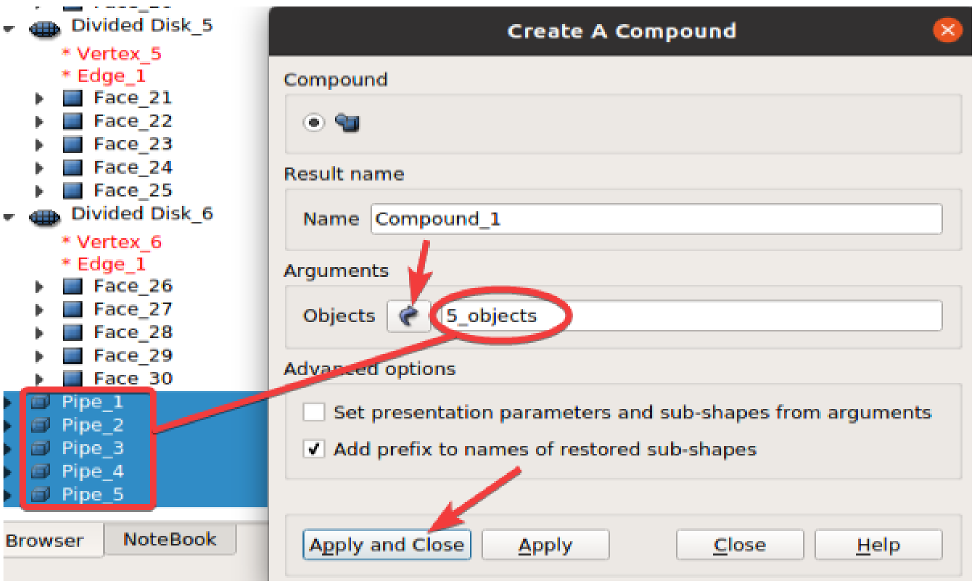
☑ Preview

Build compound

Go to *New Entity* -> *Build* -> *Compound.* Fill in the information in the table below.

Choose all the pipe sections and click *Apply and Close.* A compound with all the pipe sections has been created.

Name	**Objects**
Compund_1	Pipe_1, Pipe_2, Pipe_3, Pipe_4, Pipe_5 (5_objects)



Name: Compound_1

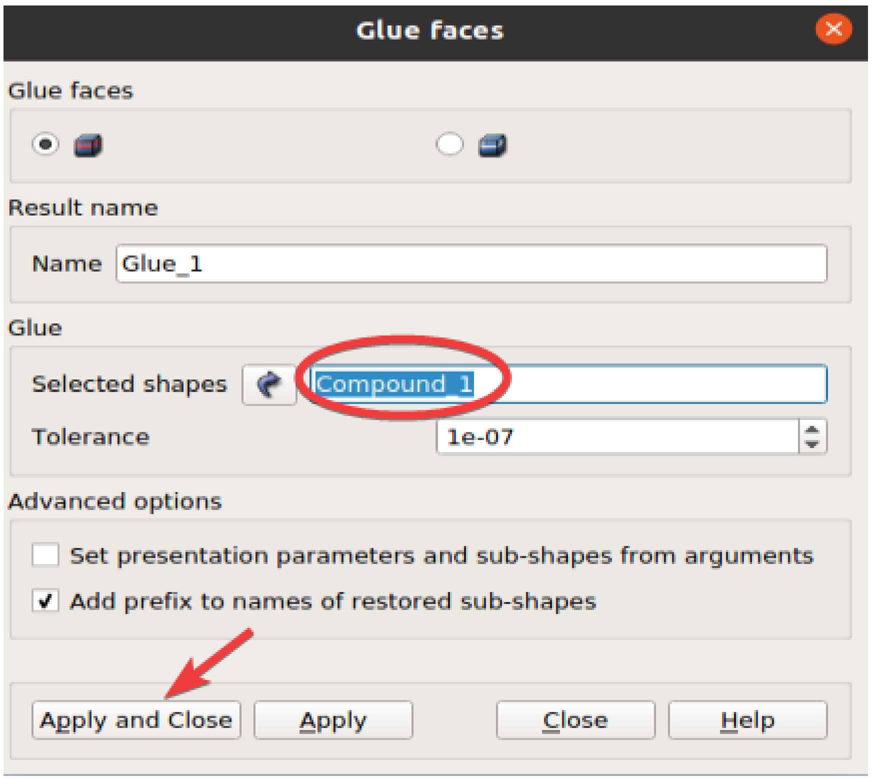
Objects: 5_objects

☐ Set presentation parameters and sub-shapes from arguments

☑ Add prefix to names of restored sub-shapes

Glue faces

Go to *Repair -> Glue Faces*. Select shape *Compound_1* and click *Apply and Close*. *Glue_1* has been created.



Name: Glue_1

Selected shapes: Compound_1

Tolerance: 1e-07

☐ Set presentation parameters and sub-shapes from arguments

☑ Add prefix to names of restored sub-shapes

Create edges groups

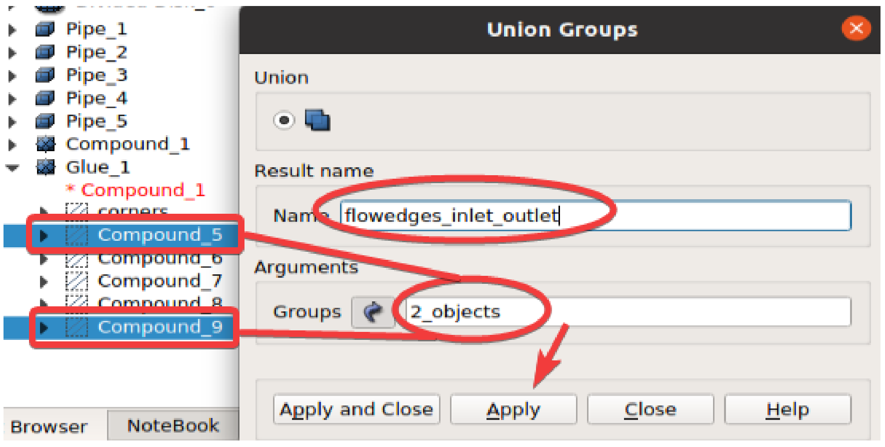
Go to *Operations* -> *Blocks* -> *Propagate*. Choose object *Glue_1*. Click *Apply and Close*. Groups *Compound_2* - *Compound_9* have been created.

Rename the created groups in the object browser by clicking F2 after selecting a group. To differentiate the groups, right-click on a group in the object browser and click *Show Only*.

Name	**Rename to**
Compound_2	corners

Delete *Compound_3* and *Compound_4.*

Go to *New Entity* -> *Group* -> *Union Groups*



Name: flowedges_inlet_outlet
Groups: 2_objects

The group union and names are shown in the table below:

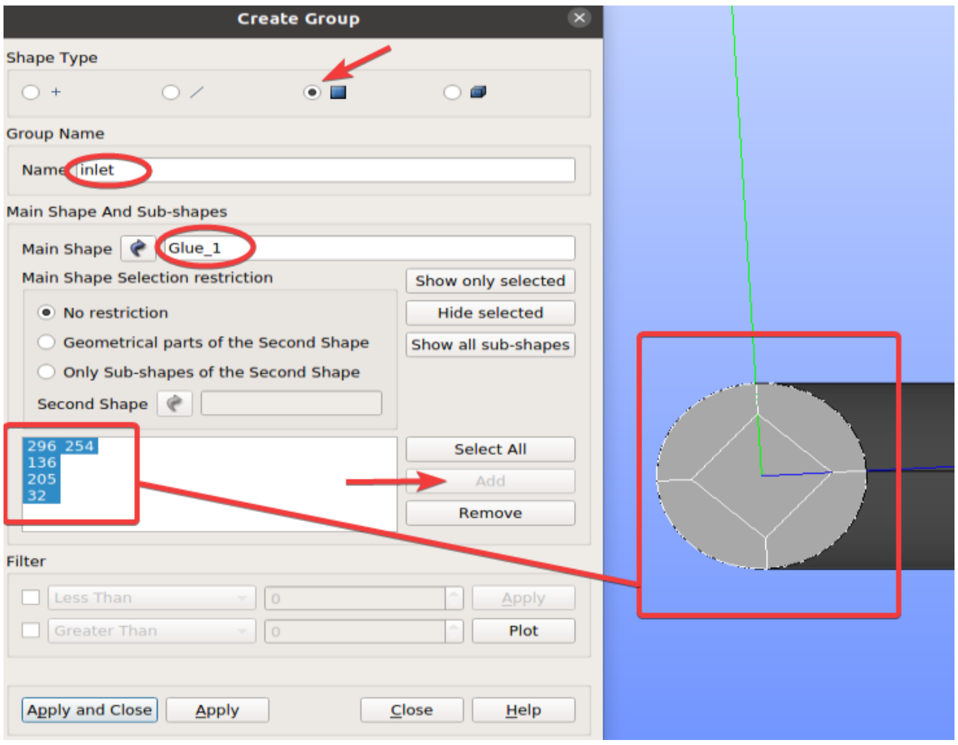
Groups	**Name**
Compound_5, Compound_9	flowedges_inlet_outlet
Compound_6, Compund_7, Compound_8	flowedges_belly

Create faces groups

Go to *New Entity* -> *Group* -> *Create Group*. Click on the third constructor (face*). Main Shape*: Glue_1. Mark the inlet, outlets, or walls, respectively, by holding down the Shift key and clicking on all the parts of the respective face group. The marked faces get white lines at the intersection of faces. Click *Add*. Click *Apply*.

Name	**Main Shape Selection Numbers**
inlet	296, 136, 205, 32, 254
outlet	310, 190, 114, 288, 239
wall	158, 171, 128, 145, 184, 101, 41, 14, 61, 81, 280, 243, 256, 264, 272, 304, 298, 292, 301, 307

Check that the groups are created correctly by only displaying one of them and moving the viewpoint (especially for the wall).



Create Group

Shape Type

Group Name

Name: inlet

Main Shape And Sub-shapes

Main Shape: Glue_1

Main Shape Selection restriction

- No restriction (selected)
- Geometrical parts of the Second Shape
- Only Sub-shapes of the Second Shape

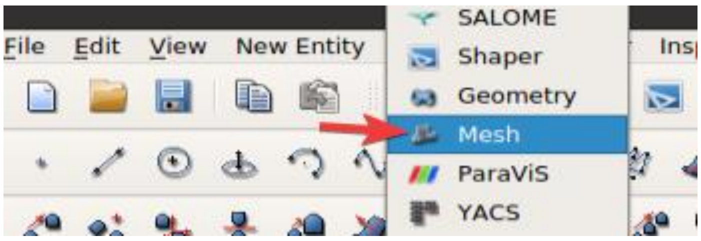
Second Shape

296 254
136
205
32

Filter

Mesh module in Salome

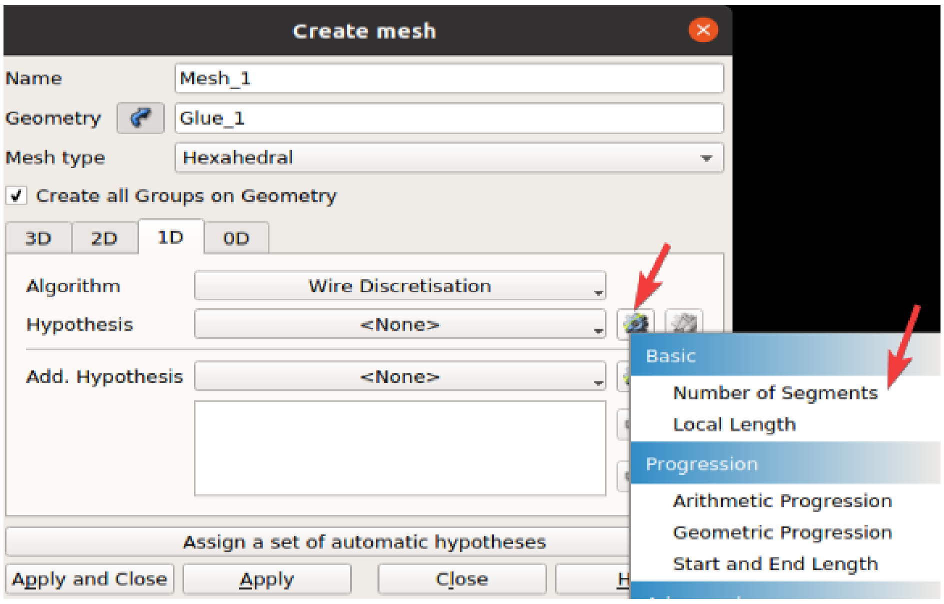
Open the Mesh module from the dropdown menu:



Mesh_1

Go to *Mesh ->Create Mesh.* Fill in the following settings:

Name	Mesh_1			
Geometry	Glue_1			
Mesh Type	Hexahedral			
Create all Groups on Geometry	Check the box			
Algorithm	**3D**	Hexahedron (i,j,k)		
	2D	Quadrangle Mapping		
	1D	Wire Discretisation		
Hypothesis	**1D**	**Number of Segments**	**Name**	Number of Segments=10
			Number of Segments	10


Create mesh

Name: Mesh_1

Geometry: Glue_1

Mesh type: Hexahedral

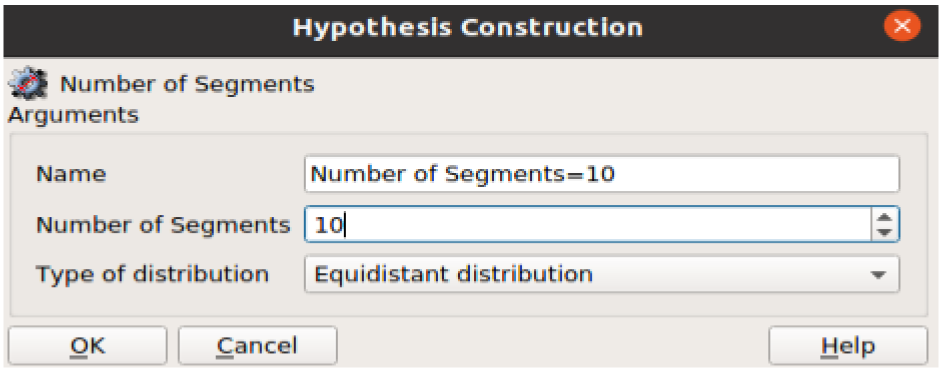
✓ Create all Groups on Geometry

3D | 2D | 1D | 0D

Algorithm: Wire Discretisation

Hypothesis: <None>

Add. Hypothesis: <None>


Hypothesis Construction

Number of Segments

Arguments

Name: Number of Segments=10

Number of Segments: 10

Type of distribution: Equidistant distribution

Click *Apply and Close*.

Right-click on *Mesh_1* in the Object Browser and click *Compute*.

Sub-mesh flowedges_inlet_outlet

Right-click on *Mesh_1* and click *Create Sub-mesh.*

Name	flowedges_inlet_outlet		
Mesh	Mesh_1		
Geometry	flowedges_inlet_outlet		
Algorithm	Wire Discretisation		
Hypothesis	Number of Segments	**Name**	Number of Segments=300
		Number of Segments	300

Click *Apply and Close.* Right-click on *Mesh_1* and click *Compute.*

Sub-mesh flowedges_belly

Right-click on *Mesh_1* and click *Create Sub-mesh.*

Name	flowedges_belly		
Mesh	Mesh_1		
Geometry	flowedges_belly		
Algorithm	Wire Discretisation		
Hypothesis	Number of Segments	**Name**	Number of Segments=50
		Number of Segments	50

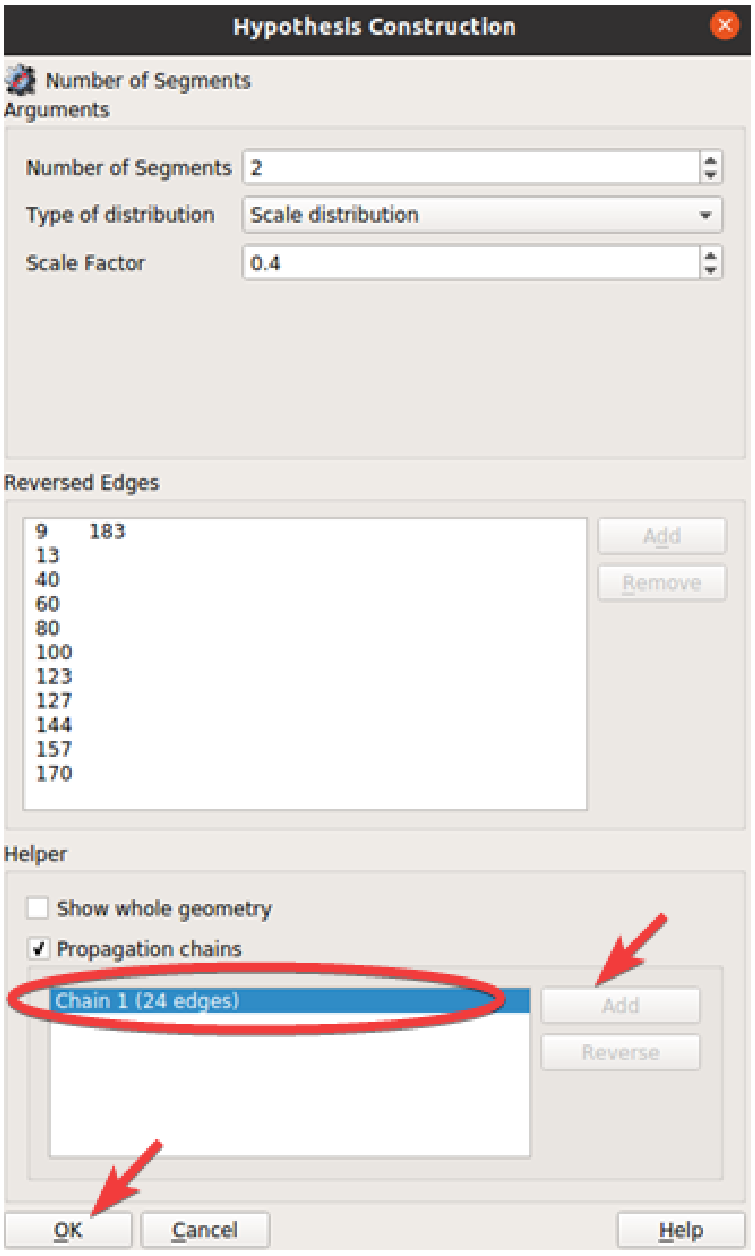
Click *Apply and Close.* Right-click on *Mesh_1* and click *Compute.*

Sub-mesh for edges

Right-click on *Mesh_1* and click *Create Sub-mesh.*

Name	edges		
Mesh	Mesh_1		
Geometry	edges		
Algorithm	Wire Discretisation		
Hypothesis	Number of Segments	**Name**	Number of Segments=2
		Number of Segments	2
		Type of distribution	Scale distribution
		Scale factor	0.4
		Helper	Check the box: *Propagation chains.* Mark *Chain 1 (24 edges)* and click *Add.*

Click *Apply and Close.* Right-click on *Mesh_1* and click *Compute.*



Number of Segments: 2
Type of distribution: Scale distribution
Scale Factor: 0.4

Reversed Edges

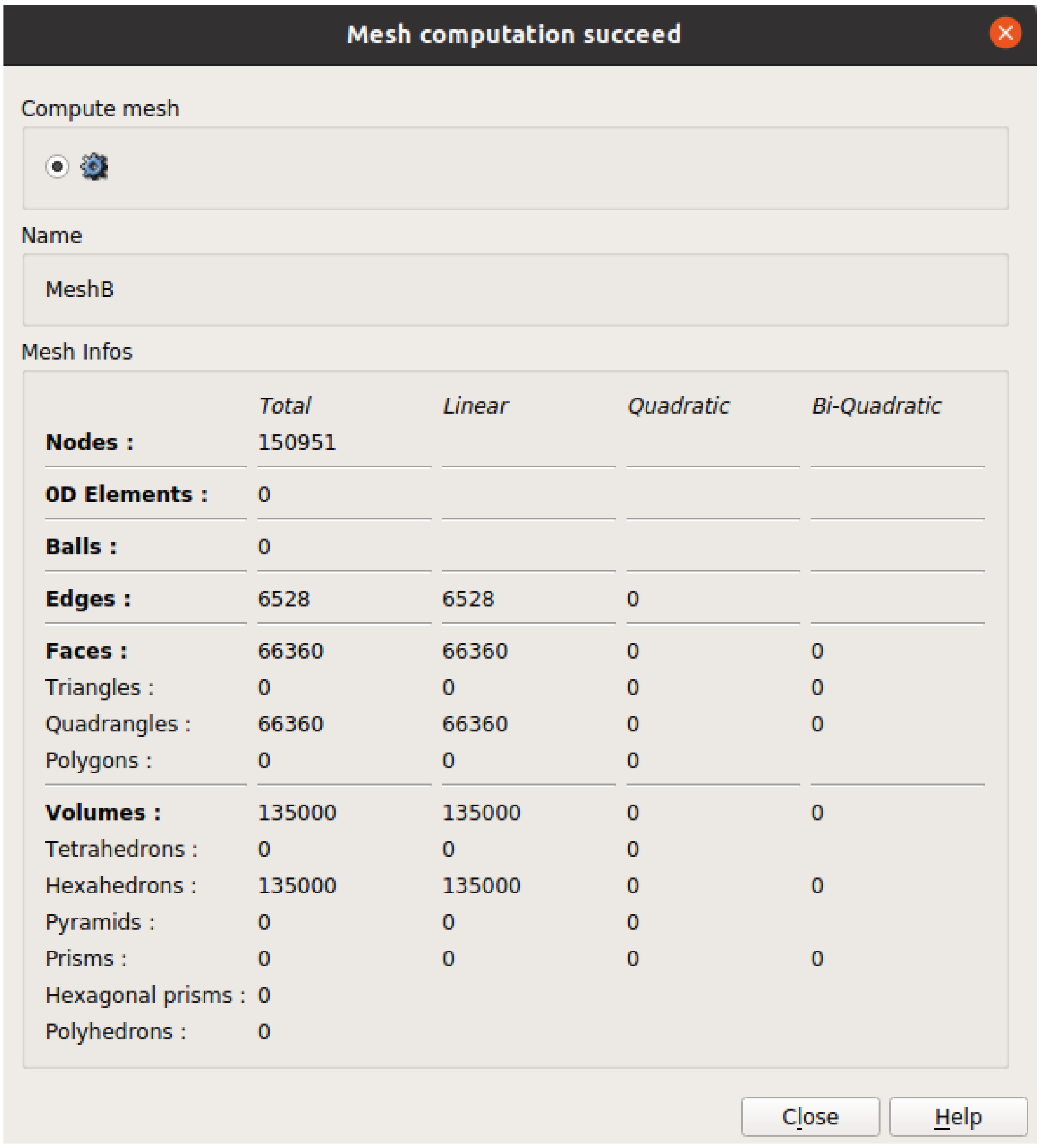
```
9    183
13
40
60
80
100
123
127
144
157
170
```

☐ Show whole geometry
☑ Propagation chains

Chain 1 (24 edges)

Mesh information

The final mesh should have the following *Mesh Info*:



Compute mesh

Name

MeshB

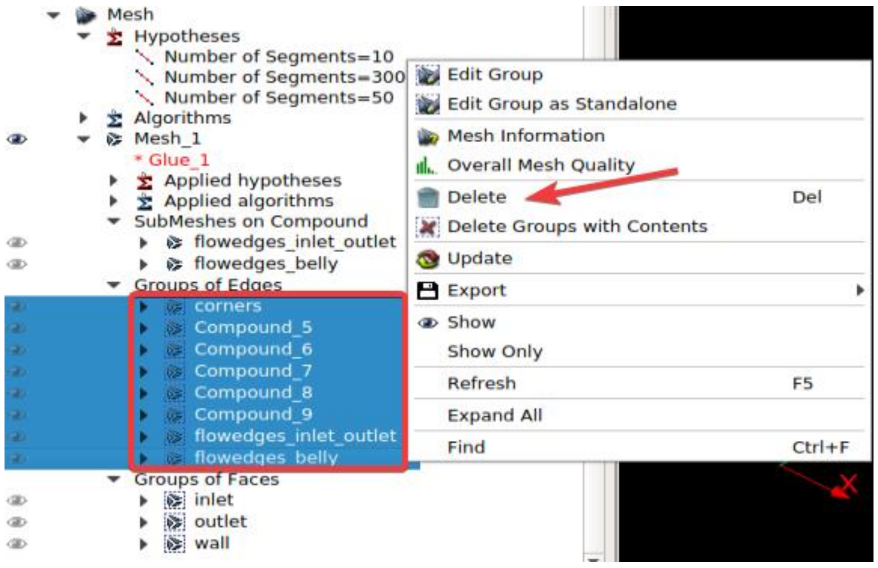
Mesh Infos

	Total	Linear	Quadratic	Bi-Quadratic
Nodes :	150951			
0D Elements :	0			
Balls :	0			
Edges :	6528	6528	0	
Faces :	66360	66360	0	0
Triangles :	0	0	0	0
Quadrangles :	66360	66360	0	0
Polygons :	0	0	0	
Volumes :	135000	135000	0	0
Tetrahedrons :	0	0	0	
Hexahedrons :	135000	135000	0	0
Pyramids :	0	0	0	
Prisms :	0	0	0	0
Hexagonal prisms :	0			
Polyhedrons :	0			

Delete Edges Groups

During the creation of *Mesh_1*, groups on geometry were created (*Create all Groups on Geometry box* was checked). However, all *Groups of Edges* must be deleted because they will create error messages in OpenFOAM.

Mark all *Groups on Edges* in the Object Browser, right-click and click *Delete.*



In contrast, The *Groups of Faces* (inlet, outlet, wall) are important, as they will be the boundaries of the mesh in the OpenFOAM simulation.

Export the mesh

The hdf-file is useless in OpenFOAM. The mesh must be exported to an unv-file. Righ-click on *Mesh_1*, click *Export* -> *UNV file.* Choose directory, and click *Save.*

Simulation in OpenFOAM

Copy the directory **pitzDaily** from opt/openfoam8/tutorials/incompressible/simpleFoam/.

Paste the directory in the wanted directory and rename it to **pipeBelly**. Copy the exported mesh, *Mesh_1.unv,* into the pipeBelly directory.

Import mesh to OpenFOAM

Open the terminal in the pipeBelly case directory (0, constant, system).

To import the mesh to OpenFOAM-format, write the command:

```
ideasUnvToFoam Mesh_1.unv
```

OpenFOAM uses the SI system, but the geometry made in Salome does not have a length unit. Transform points to mm by writing:

```
transformPoints -scale '(1e-3 1e-3 1e-3)'
```

Next, check the mesh:

```
checkMesh
```

Constant directory

polyMesh

The polyMesh directory contains the imported mesh. Open the *boundary* file. Change the wall type from *patch* to *wall* and save:

```
    wall
    {
        type            wall;
        nFaces          30000;
        startFace       390180;
    }
```

transportProperties

Change nu (the kinematic viscosity) to:

```
      nu              [0 2 -1 0 0 0 0] 1.5e-05;
```

0 directory

Delete *f, nuTilda, omega and v2* (we will not need them in the simulation).

Must change the *boundaryField* in all the files. We need to have the same groups we created in Salome:

- inlet
- outlet
- wall

In all the files:

- Delete *lowerWall, frontAndBack* and its contents.
- Change name from *upperWall* to *wall.*

P

Don not make any changes.

nut

Do not make any changes.

U

Change velocity from 10 m/s to 3 m/s:

```
        value           uniform (3 0 0);
```

k

Change the value of k to **0.043457**:

```
internalField   uniform 0.043457;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.043457;
    }
    outlet
    {
        type            zeroGradient;
    }
    wall
    {
        type            kqRWallFunction;
        value           uniform 0.043457;
    }
}
```

epsilon

Change the value of epsilon to **1.06327**:

```
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1.06327;
    }
    outlet
    {
        type            zeroGradient;
    }
    wall
    {
        type            epsilonWallFunction;
        value           uniform 1.06327;
    }
}
```

Table N.1 gives an overview of the boundary and initial conditions.

Table N.1: Boundary and initial conditions.

Type	$\boldsymbol{U}$ $[m/s]$	$\boldsymbol{p}$ $[m^2/s^2]$	$\boldsymbol{k}$ $[m^2/s^2]$	$\boldsymbol{epsilon}$ $[m^2/s^3]$	$\boldsymbol{nut}$ $[m^2/s]$
internal Field	uniform (0 0 0)	uniform 0	uniform 0.043457	uniform 1.06327	uniform 0
inlet	fixedValue uniform (3 0 0)	zeroGradient	fixedValue uniform 0.043457	fixedValue uniform 1.06327	calculated uniform 0
outlet	zeroGradient	fixedValue uniform 0	zeroGradient	zeroGradient	calculated uniform 0
wall	noSlip	zeroGradient	kqRWallFunction uniform 0.043457	epsilonWallFunction uniform 1.06327	nutkWallFunction uniform 0

System directory

Delete *blockMeshDict* and *streamlines*.

controlDict

Change deltaT to 0.0001 and writeInterval to 50:

```
deltaT          0.0001;
writeInterval   50;
```

Delete the following lines:

```
cacheTemporaryObjects
(
    kEpsilon:G
);

functions
{
    #includeFunc streamlines
    #includeFunc writeObjects(kEpsilon:G)
}
```

fvSchemes

do not make any changes.

fvSolution

Change to the values in the lines marked yellow:

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-09;
        relTol          0.01;
        smoother        GaussSeidel;
    }

    "(U|k|epsilon|omega|f|v2)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-09;
        relTol          0.1;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    consistent      yes;

    residualControl
    {
        p               1e-3;
        U               1e-3;
        "(k|epsilon|omega|f|v2)" 1e-4;
    }
}
```

```
relaxationFactors
{
    equations
    {
        U               0.8; // 0.9 is more stable but 0.95 more convergent
        “.*”            0.8; // 0.9 is more stable but 0.95 more convergent
    }
}
```

Residuals script

Open the text editor in Ubuntu and paste the following lines:

```
set term png
set output “residuals.png”
set logscale y
set title “Residuals”
set ylabel ‘Residual’
set xlabel ‘Iteration’
plot “<cat log | grep ‘Solving for Ux’ | cut -d ' ’ -f9 | tr -d ‘,'"
title 'Ux'
with lines, \
"<cat log | grep 'Solving for Uy' | cut -d ' ' -f9 | tr -d ','"
title 'Uy'
with lines, \
"<cat log | grep 'Solving for Uz' | cut -d ' ' -f9 | tr -d ','"
title 'Uz'
with lines, \
"<cat log | grep 'Solving for p' | cut -d ' ' -f9 | tr -d ','"
title 'p'
with lines, \
"<cat log | grep 'Solving for k' | cut -d ' ' -f9 | tr -d ','"
title 'k'
with lines, \
"<cat log | grep 'Solving for epsilon' | cut -d ' ' -f9 | tr -d ','"
title 'epsilon'
with lines, \
```

Save the file in the pipeBelly case directory and name it *residuals.*

Case directory

Open the *Allrun* file and change the contents to:

```
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

simpleFoam > log &
tail -200f log

#------------------------------------------------------------------------------
```

Run the script

Open the terminal from the case directory and run the command:

```
./Allrun
```

The solution is converged when the following line appears:

```
SIMPLE solution converged in 0.0476 iterations
```

When the solution converges, stop the process with the command:

```
Ctr+C
```

To create the residual plot: write in the terminal:

```
gnuplot residuals -
```

Check y^+:

```
simpleFoam -postProcess -func yPlus
```

To post-process, open ParaView:

```
paraFoam
```

Trouble shooting

Salome

It is recommended to make use of the Salome forum [1] when creating a mesh in Salome. Using the search bar, one may find discussions and answers to problems, meshes and geometries similar to your own case. Frequently, people will post answers in the form of a python script. It is a good idea to download the script and evaluate it both in a text editor program as well as to load it in the Salome GUI. Examining both the Python script and the objects in the GUI, it should be possible to understand what is done. Then it may be easier to implement the same method in your own case. In addition, an extensive collection of pre-made example scripts is available in the Salome documentation. The Salome documentation is available for download as an extra package on the Salome download page. There are also many videos available on YouTube, in addition to the written documentation.

OpenFOAM

When creating a simulation in OpenFOAM it is recommended to take a good look at the OpenFOAM User Guide [2] located on the path *OpenFOAM /opt/openfoam8/doc/Guides* and look here first if you have questions.

It is also a good idea to make use of the CFD Online Forum [3] if the information you are looking for is not easily available in the literature. There are discussions about many topics, and it is possible to search for specific problems. However, one must remember to not trust blindly in claims made in online discussion forums, but try to find more reliable sources to back up the information. Still, it might be a good help for convergence problems, simulation setup and for information about checking and validating the simulation. One tip is to create an account in the forum early on in the CFD learning process, because attached files and images in the forum are not available for non-users.

References

[1] SALOME, *SALOME forum*, OPEN CASCADE, 2005-2021. Accessed on: Jan.-Apr., 2021. [Online]. Available: https://www.salome-platform.org/forum

[2] C. J. Greenshields, "OpenFOAM User Guide version 8," OpenFOAM Foundation Ltd., CFD Direct Ltd., July 22, 2020.

[3] *CFD Online Forum*, CFD online, 1994-2021. Accessed on: Jan.-Apr., 2021. [Online]. Available: https://www.cfd-online.com/Forums/