Computational Fluid Dynamics in OpenFOAM

Mesh Generation and Quality

Rebecca Gullberg

December 1, 2017

TKP 4555 Advanced Process Simulation

Abstract

In this report, three different mesh generation methods for OpenFOAM are studied and tested. Two of these, blockMesh and snappyHexMesh, are supplied with OpenFOAM, while the third, Gmsh, is an external mesh generation software. The most important metrics used to assess mesh quality in OpenFOAM are also presented.

Contents

1 Introduction

Computational fluid dynamics (CFD) is a method used to numerically analyze fluid flows. Aerospace engineers were the primary developers of CFD tools and CFD analysis has been widely used in Aerodynamics research over the years. Since the 1980s process industries has shown increased interest in CFD for design purposes. In the early 1990, CFD tools were adapted for chemical engineering purposes and applications for chemical reactions in fluid flows were added. In chemical engineering today, CFD is used for e.g. design of chemical reactors and mixing processes [1].

CFD requires discretization of geometries into smaller cells, defined as grid or mesh. In these cells, the Navier-Stokes equations describing the fluid problem can be solved. Mesh generation is an important step in CFD studies, since it affects both the accuracy and efficiency of the solution. There are adaptive mesh algorithms available, but they often require human intervention to be able to mesh complex geometries. Meshing is therefore considered to be a significant bottleneck in CFD analysis and it is not uncommon that CFD users spend more than 50% of their time on mesh generation for a CFD project [2].

The aim of this project has been to study the metrics used for mesh quality assessment and to investigate different mesh generation methods. This report will discuss the choice of discretization method, cell shape, mesh type, mesh generation method and the definition of a "good" mesh. The focus has been on the open source software's OpenFOAM and Gmsh. OpenFOAM is a C++ toolbox of numerical solvers with CFD capabilities and includes pre/post-processing applications [3]. Mesh generation can be done directly in OpenFOAM with the utilities blockMesh and snappyHexMesh. Gmsh is an external mesh generation software and the mesh generated in Gmsh can be converted into OpenFOAM format [4].

2 Definitions

In OpenFOAM, the geometric domain is divided into smaller control volumes, called cells. Each cell has a cell center and faces surrounding the cell. Internal faces connect cells in a geometry, while boundary faces, also called patches, are part of the boundary of the computational domain. The face area vector, $\boldsymbol{S_f}$ is the normal vector at the face centroid and should be pointed outwards of the cell. The center-to-center vector between adjacent cells is denoted as $\boldsymbol{d}$.

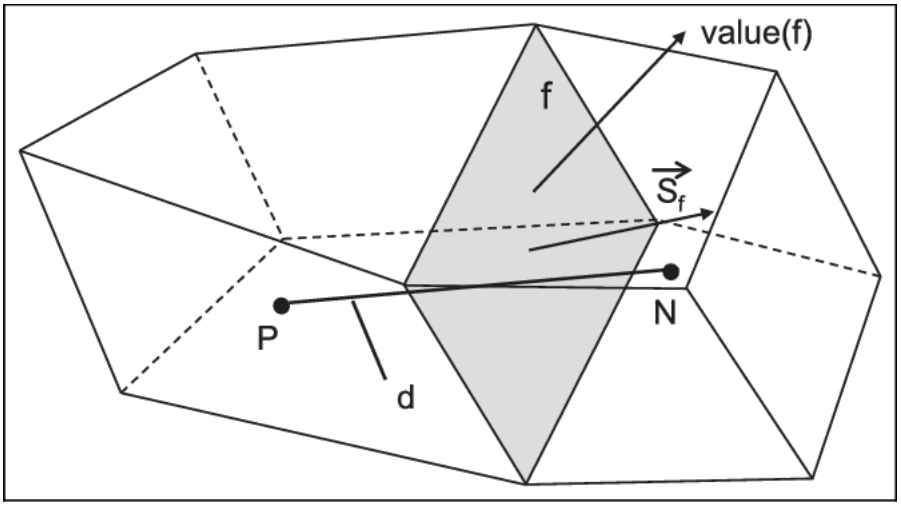


Figure 1: Neighboor cells, P and N, with internal face f and internal face area vector $\boldsymbol{S_f}$ [5].

A valid mesh must satisfy the properties listed below [6].

- the whole computational domain needs to be covered in cells
- every cell must be convex and its cell centre inside the cell
- every cell must be closed

3 Types of Mesh

Triangle and quadrilateral shaped cells are the most used cell types in 2D meshes. In 3D meshes, hexahedron or tetrahedron cells are the most common cell shapes.

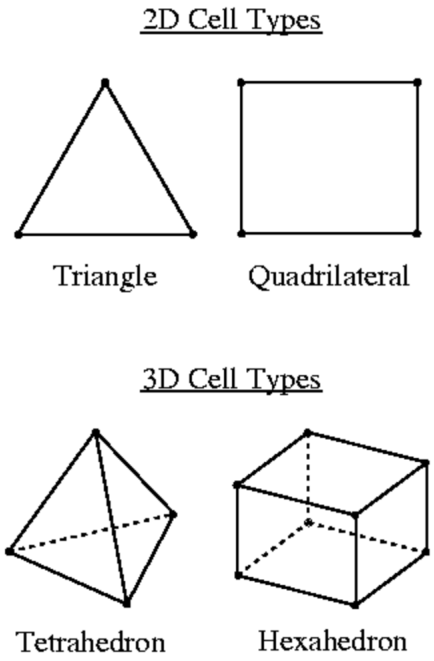
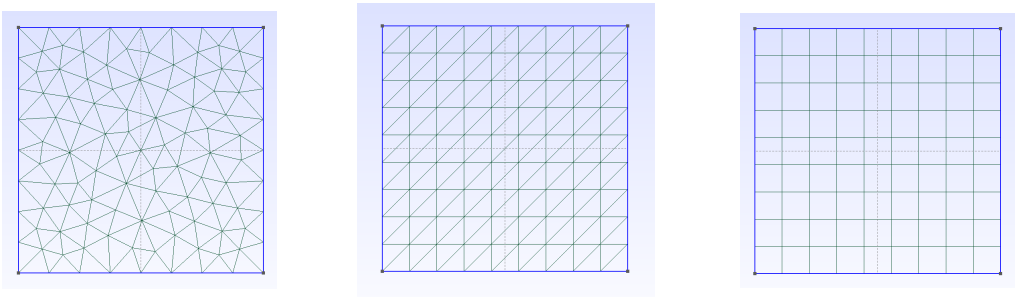


Figure 2: Commonly used cell shapes [7].

It is easy to automatically generate unstructured meshes consisting of triangles or tetrahedrons for complex geometries. However, structured meshes with quadrilateral or hexahedron elements have several advantages over unstructured meshes in OpenFOAM.



(a) (b) (c)

Figure 3: Figure of (a) unstructured triangular mesh, (b) structured triangular mesh and (c) structured quadrilateral mesh.

Structured meshes are easier to implement in a natural sequence than unstructured meshes. Generally, the solvers in OpenFOAM can more efficiently solve structured meshes. The disadvantage with structured meshes is that they have low adaptability to complex geometries [8].

4 Discretization Methods

To gain a better understanding of OpenFOAM and the assessment of mesh quality, it is useful to look into how the partial differential equations are solved in the discretized cells. Often, the PDEs governing fluid dynamics problems are non-linear and coupled. The analytical solutions are in a lot of cases not possible to obtain. Therefore, the equations need to be discretized, transformed into a set of algebraic equations and solved numerically. The most important discretization methods are Finite difference method (FDM), Finite element method (FEM) and Finite volume method (FVM).

FDM is one of the oldest methods for discretizing PDEs. FDM approximates the partial derivatives with Taylor expansions. Nodes are spaced in a square grid and the equations are solved at the nodes. FDM works straighforward for simple geometries with structured grids, but the Taylor expansions need to be transformed for more irregular shapes

FEM is used in some CFD software e.g. COMSOL Multiphysics [9], where the domain is divided into elements. In 2D, the elements are often triangular with 3 nodes associated with each element. The unknown variables are computed at the nodes and used to approximate the values inside the elements. FEM works for both unstructured meshes and complex geometries. The disadvantage with this method is that it is a highly mathematical approach and the algebraic equations obtained do not have any physical meaning [10].

Finite volume method is clearly the most popular method used in CFD software e.g. OpenFOAM, ANSYS Fluent. The reason for this is that all the terms that are approximated have a physical meaning. Each node is surrounded by a small volume. The differential equations are integrated over each volume and transformed from volume

integrals into surface integrals. The unknown variables are stored at each node (cell center) and linear interpolation is used to approximate the fluxes at the face centroid [11].

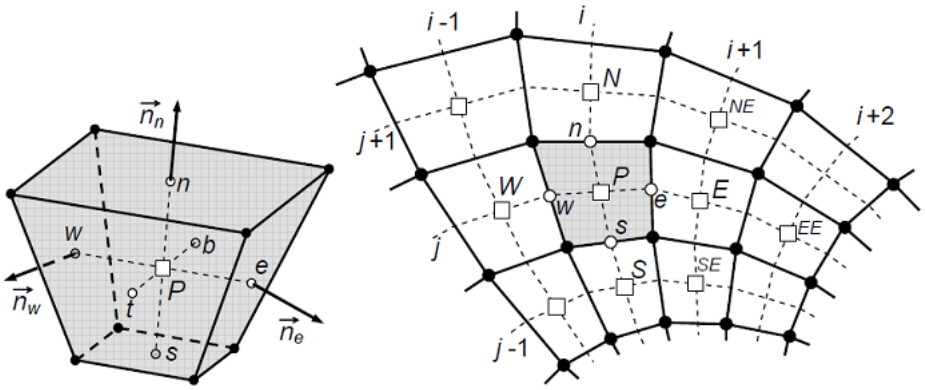


Figure 4: Figure of a single cell (left) and a mesh consisting of discretized volumes (right) used in FVM [6].

The generic conservation equation for a quantity, ϕ, can be written as follows

$$\underbrace{\frac{\partial(\phi)}{\partial t}}_{\text{unsteady term}} + \underbrace{\bigtriangledown \cdot (\boldsymbol{v}\phi)}_{\text{convection term}} = \underbrace{\bigtriangledown \cdot (\Gamma^{\phi} \bigtriangledown \phi)}_{\text{diffusion term}} + \underbrace{Q^{\phi}}_{sourceterm} \tag{1}$$

The steady state form is given by

$$\bigtriangledown \cdot (\boldsymbol{v}\phi) = \bigtriangledown \cdot (\Gamma^{\phi} \bigtriangledown \phi) + Q^{\phi} \tag{2}$$

By integrating the equation over a control volume, V_c, the expression becomes

$$\int_{V_c} \bigtriangledown \cdot (\boldsymbol{v}\phi) dV = \int_{V_c} \bigtriangledown \cdot (\Gamma^{\phi} \bigtriangledown \phi) dV + \int_{V_c} Q^{\phi} dV \tag{3}$$

The convective and diffusive terms can be replaced by surface integrals through the use of the divergence theorem

$$\int_{\partial V_c} (\boldsymbol{v}\phi) \cdot dS = \int_{\partial V_c} (\Gamma^{\phi} \bigtriangledown \phi) dS + \int_{V_c} Q^{\phi} \cdot dV \tag{4}$$

The surface integrals can be approximated as the sum of fluxes of all the face elements, while the value of the source term at the center is assumed to represent the average value of the control volume. The expression above is approximated as

$$\sum_f \boldsymbol{S_f} \cdot (\boldsymbol{v}\phi_f) = \sum_f \boldsymbol{S_f} \cdot (\Gamma^{\phi} \bigtriangledown \phi_f) + Q^{\phi} V_c \tag{5}$$

where f is a face on the surface and S_f is the face area vector. As earlier mentioned, the values of the unknown quantities are stored at the cell centers. To be able to solve Equation 5, the quantity ϕ needs to be approximated at the cells faces. The value at the face centroid, ϕ_f between cells P and N, can be obtained by interpolation

$$\phi_f = f_x \phi_P + (1 - f_x)\phi_N \tag{6}$$

where $f_x = \overline{fN}/\overline{PN}$. To approximate the gradient at a cell face, the difference between the values of the neighboring cell centers is divided by the center-to-center vector, $\boldsymbol{d}$.

$$(\bigtriangledown\phi)_f = \frac{\phi_N - \phi_P}{|\boldsymbol{d}|} \tag{7}$$

FVM is a robust method for discretization of conservation laws, ensuring that the flux leaving one cell is equal to the flux entering the neighbor cells. The disadvantage with FVM is that high precision can be difficult to obtain for complex geometries. For structured meshes, the fluxes can easily be interpolated from the cell center to the face. However, complex geometries are likely to generate irregularities in meshes, causing precision loss. This disadvantage affects how we define mesh quality in OpenFOAM [11].

5 Mesh Quality

There exists no clear definition of what is a "good" mesh, but the quality of a grid is generally measured by the main metrics; non-orthogonality, skewness, aspect ratio and smoothness [12]. What thresholds of these metrics define a high-quality mesh varies from solver to solver.

Non-orthogonality is defined as the angle between the face area vector, $\boldsymbol{S_f}$ and the center-to-center vector, $\boldsymbol{d}$. This angle can cause numerical errors when diffusive terms, $\Gamma^{\phi} \bigtriangledown^2 \phi$, are included.

$$\int_V \bigtriangledown(\mu \bigtriangledown \phi) dV \approx \sum_f \mu_f \boldsymbol{S_f} \cdot (\bigtriangledown\phi)_f \tag{8}$$

The diffusion term is calculated by taking the dot product between $\boldsymbol{S_f}$ and the approximated gradient, $(\bigtriangledown\phi)_f$.

$$\boldsymbol{S_f} \cdot (\bigtriangledown\phi)_f = |\boldsymbol{S_f}| \frac{\phi_N - \phi_P}{|\boldsymbol{d}|} \tag{9}$$

A significant numerical error can be introduced if the cells are non-orthogonal, meaning that there is a misalignment between $\boldsymbol{d}$ and $\boldsymbol{S_f}$. The default maximum value for non-orthogonality in OpenFOAM is 70 degrees, but it is desirable to keep the angle as small as possible.

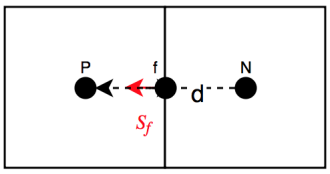


Figure 5: Orthogonal cells.

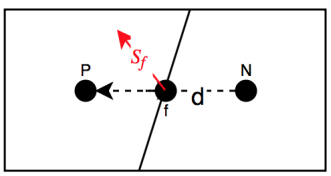


Figure 6: Non-orthogonal cells.

For a mesh with orthogonal cells aligned with the flow, the face quantities can be accurately interpolated from the cell centers. It is therefore best practice to use orthogonal, flow aligned cells. Flow misalignment, as seen in Figure 7, causes the flow to pass

an angle, which can affect the accuracy by introducing non-negligible numerical diffusion [13].

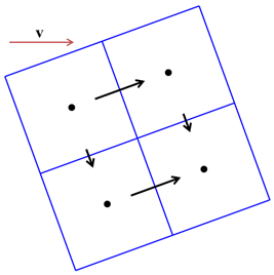


Figure 7: A representation of flow misalignment [13].

Another concern that can cause significant numerical diffusion is skewness [11]. Skewness arises when there is a discrepancy between the location of the face center, f, and where the center-to-center vector meets the face, f', as seen in Figure 8. Skewness, ϵ can be calculated from the expression

$$\epsilon = \frac{|f - f'|}{|\boldsymbol{d}|} \tag{10}$$

The ideal value for skewness i zero and the threshold for skewness in OpenFOAM is four.

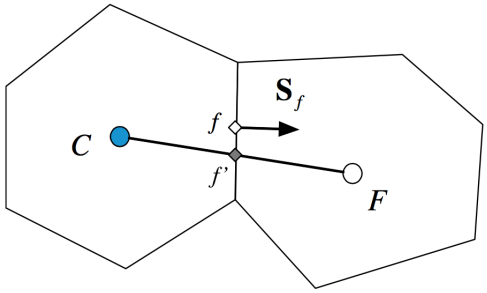


Figure 8: Skewness between neighbor cells [11].

Aspect ratio is the ratio between a cells maximum and minimum width. Large aspect ratios will decrease the efficiency and accuracy of the linear solvers in OpenFOAM. In addition, large aspect ratios introduce interpolation errors by smearing out the gradient, causing numerical diffusion. Large aspect ratios should especially be avoided in areas where the flow changes rapidly and the gradient is large. The aspect ratio should be as close to 1 as possible, but the standard threshold in OpenFOAM is 1000.

It is also important to avoid sudden jumps in cell sizes by keeping the mesh smooth. Sudden jumps in the size will cause interpolation error, adding diffusion to the solution. To ensure the mesh is smooth, the difference between minimum and maximum cell volume should be as small as possible [14].

The last step in a meshing procedure is a refinement study, where the mesh density is increased by adding more cells. The mesh is made finer and finer until grid independence is reached, meaning that there is no significant change in the solution when the grid is refined. The refinement processes may also include adding cells locally within boundary layers to capture significant flow features. However, it is important not to "over-mesh" as this may cause the model to become inefficient and unsolvable [15].

6 Introduction to OpenFOAM

This section includes a short introduction on how to run a simulation in OpenFOAM. OpenFOAM provides you with tutorial cases to solve a wide range of problems. The easiest way to create a model in OpenFOAM is by modifying an already existing tutorial provided by OpenFOAM.

The cavity flow case is the simplest example used to illustrate incompressible and isothermal flow. The user should start out by copying the folder from `FOAM_TUTORIALS/incompressible/icoFoam/cavity` to their directory.

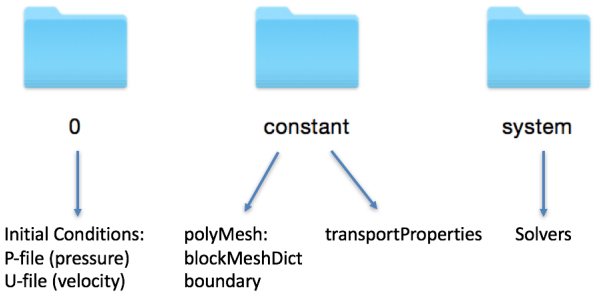


Figure 9: Structure of an OpenFOAM case.

An OpenFOAM case consists of three main folders, a 0 folder, a constant folder and a system folder. The 0 folder contains two separate files with initial conditions for the pressure, P, and the velocity, U. The constant folder contains a polyMesh folder, with files holding information about the mesh, and a file with transportProperties. The system folder contains information about the solvers and the simulation time.

The tutorial case named cavity can be run with a new geometry by replacing the polyMesh folder and updating the initial conditions to fit the new geometry. The different mesh generation methods that can be used to replace the geometry will be discussed in the next section. The following procedure can be used to run a case with a new geometry in OpenFOAM.

1. Generate the mesh by using blockMesh, snappyHexMesh or Gmsh.

2. Run the command "checkMesh" in the terminal. This command checks if the new mesh is valid and verifies that none of the grid quality thresholds are crossed.

3. Run the solver command in the terminal. The solver "icoFoam" can be used for laminar and isothermal flow. Solving the case will create more time directories in the case folder (1 folder, 2 folder etc. in addition to the 0 folder).

4. Run the command "touch case.foam". This generates a new file with the name "case.foam" that can be opened in ParaView for post processing.

The following is the checkMesh output for the cavity case.

```
Max aspect ratio = 1 OK.
Minimum face area = 2.5e-05. Maximum face area = 5e-05.
Face area magnitudes OK.
Min volume = 2.5e-07. Max volume = 2.5e-07.  Total volume = 0.0001.
Cell volumes OK.
Mesh non-orthogonality Max: 0 average: 0
Non-orthogonality check OK.
Face pyramids OK.
Max skewness = 1e-08 OK.
Coupled point location match (average 0) OK.
Mesh OK.
```

The checkMesh command checks for non-orthogonality, skewness, aspect ratio and smoothness (difference between min and max volume), and makes sure that the mesh satisfies all the properties of a valid mesh by printing "Mesh OK" at the end.

Paraview is an open-source, visualization application for post-processing provided with OpenFOAM. ParaView can also be useful if the command "checkMesh" fails. For example, if some of the cells in the mesh cross the threshold for non-orthogonality, running the checkMech command will add these cells to a special set of cells, called nonOrthoFaces. By typing the command "foamToVTK - faceSet nonOrthofaces" the Visualization Toolkit (VTK) in OpenFOAM creates a VTK file. This VTK file can be opened in ParaView to visualize the "bad" cells. ParaView also contains a mesh quality filter that can be used to evaluate the quality of the cells by adjusting the threshold for different metrics. ParaView does not use the same mesh quality metrics as OpenFOAM, but it is a useful feature to get a general overview of the mesh quality.

7 Mesh Generation Methods

Three different mesh generation methods have been investigated in this project. The first two methods, blockMesh and snappyHexMesh, are mesh utilities supplied with OpenFOAM. OpenFOAM can convert meshes from a wide range of external software's. The main focus in this project has been on the external software Gmsh, since it was found to be a fast and user-friendly method.

7.1 Block Mesh

BlockMesh is the most basic mesh generation tool in OpenFOAM and can be used to generate simple block meshes for hexahedron geometries. The cavity tutorial case is used to explain this utility. The cavity case is a 2D simulation with incompressible and isothermal flow in a cavity with a moving wall at the top.

All the information about the mesh can be found in the blockMeshDict, located in the constant/polymesh folder. To begin with, the mesh size can be scaled with the command "convertToMeters". Thereafter, all the corners of the hexahedron, called vertices, are defined. Starting from 0, the vertexes are defines as seen in Figure 10.

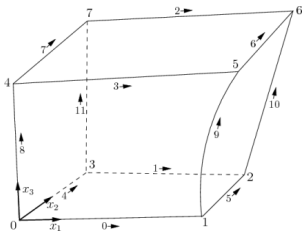


Figure 10: Definition of vertexes and patches in the blockMeshDict [6].

The blockMeshDict is written as follows for the cavity case

```
convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks //defines hexahedron block
(
    hex (0 1 2 3 4 5 6 7) // vertex numbers
    (20 20 1) // numbers of cells in each direction
);

boundary
(
    movingWall          //patch name
    {
        type wall;      //patch type
        faces
        (
            (3 7 6 2)   //face
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
```

```
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

mergePatchPairs        //merge patches
(
);
```

The next entry defines the hexahedron block from an ordered list of the vertex numbers and with the number of cells in each direction. The last entries define the boundary patches as a list of faces and assigns them patch names and patch types (physical properties). The faces are defined as lists of vertexes by the right-hand-rule, e.g. (0 3 2 1) means the bottom face in Figure 10. The "frontAndBack" boundary is made empty, which is required for a 2D simulation. OpenFOAM is always working as a 3D CFD tool, but by making these boundary patches empty, OpenFOAM can be "fooled" into working in 2D mode. When running the command "blockMesh", all the information about the mesh is transferred to points, faces, cells and boundary files in the constant/polymesh folder.

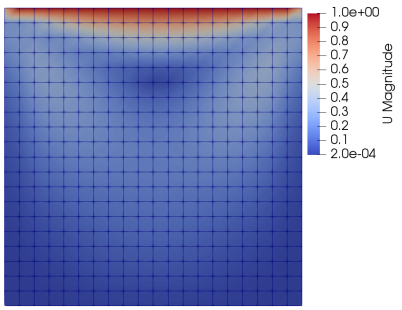


Figure 11: 2D simulation of cavity case with mesh generated with blockMesh.

The blockMesh utility is mostly used for simple geometries or as a tool for generating an initial mesh in the snappyHexMesh procedure, as you will see in next section.

7.2 Snappy Hex Mesh

SnappyHexMesh is an OpenFOAM tool used to generate meshes for more complex 3D geometries. The snappyHexMesh tutorial found at `FOAM_TUTORIALS/mesh/snappyHexMesh` can be used as an base case and should be copied from the tutorial directory.

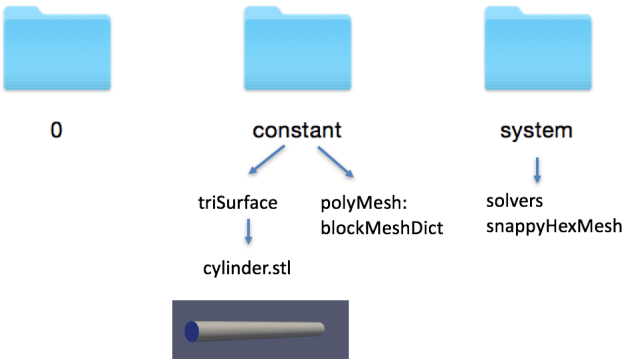


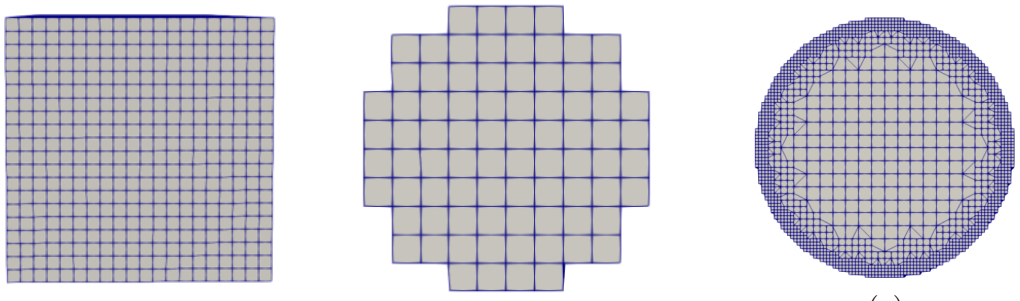
Figure 12: snappyHexMex case folder with a new geometry defined as cylinder.stl.

This utility can generate any 3D geometry from a tri-surface (surface built of triangular surfaces) in a good quality stereolithography format (STL). The STL file can be generated in an external computer-aided design (CAD) tool and added to the folder constant/triSurface. In this example, a STL file of a simple cylinder geometry is used. In the snappyHexDict file located in the system folder, the geometry is specified with its physical boundaries.

```
geometry
{
    cylinder.stl
    {
        type triSurfaceMesh;
        name cylinder;
    regions
    {
        suplat {name suplat;} //name of walls
        inlet  {name inlet;}
        outlet {name outlet;}
    }
    }

};
```

An initial mesh is created by dividing the geometry into a hexahedron grid by running the blockMesh command (defined in blockMeshDict), as seen in Figure 13(a). Thereafter, the cells intersecting with the flow boundary domain are "snapped away" by running the snappyHexMesh command, as seen in Figure 13(b).



(a) (b) (c)

Figure 13: Mesh after running the command (a) blockMesh and (b) snappyHexMesh. The final result (c) after refinement.

In the snappyHexMeshDict file there are a number of refinement parameters and snapControl parameters that can be used to refined the mesh. All these parameters are well explained in this file. Region-wise refinement was applied to the walls of the cylinder and the snapControls were improved by increasing the number of iterations. The final result can be seen in Figure 13(c). In addition, layers of cells can be added to specific boundary patches.

SnappyHexMesh is a utility that can generate very complex geometries automatically from surface geometries. However, some tuning of snapControl parameters and refinement parameters are needed.

7.3 Gmsh

The external software Gmsh makes it easier to visualize the mesh as it is generated compared to the previous methods. Geometries can be created by using the graphical user interface (GUI) or by using Gmsh's own scripting language. The next section includes two examples of meshes created in Gmsh, using a combination the GUI and Gmsh code. Figure 14 shows the GUI, that is intuitive and easy to use for simple geometries. However, the need for programming the geometry increases with the complexity of the geometry. Gmsh has basic build-in functions such as Cos, Sine and Log, and loops such as If and For.

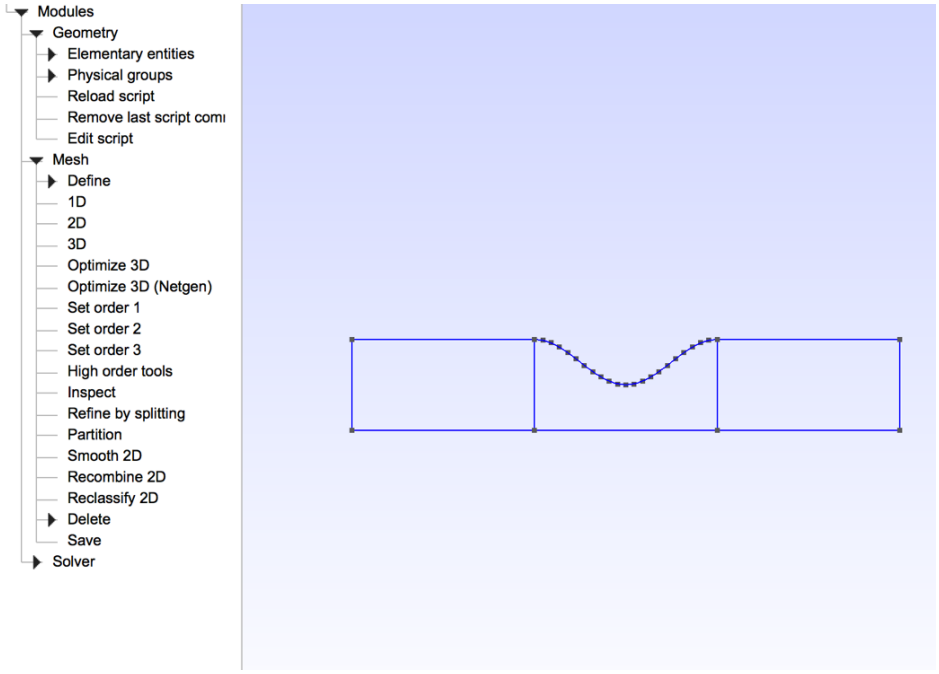


Figure 14: Graphical interface in Gmsh

Starting out with the cavity base case, the constant folder should be deleted while keeping the transportProperties file. The new mesh, called pipe2D.msh, can then be added to the folder, as seen in Figure 15,

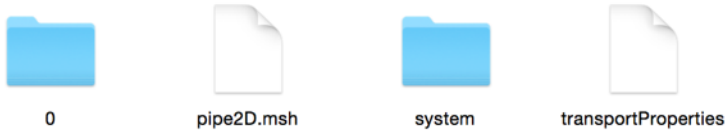


Figure 15: Cavity case folder before running the command "gmshToFoam".

By running the command "gmshToFoam pipe2D.msh", a new constant folder is created with all the information about the new mesh. The transportProperties file should then be moved back to the constant folder before running checkMesh and the solver command.

Gmsh is a powerful tool for creating block-structured meshes. It can be difficult to create a structured mesh from a single zone when the geometry is complex. The geometry domain can therefore be divided to into smaller hexahedron subvolumes, where structured meshes are created. For example, the mesh quality metrics of a cylinder can be increased by dividing the geometry into new subdivisions, as shown in Figure 16.

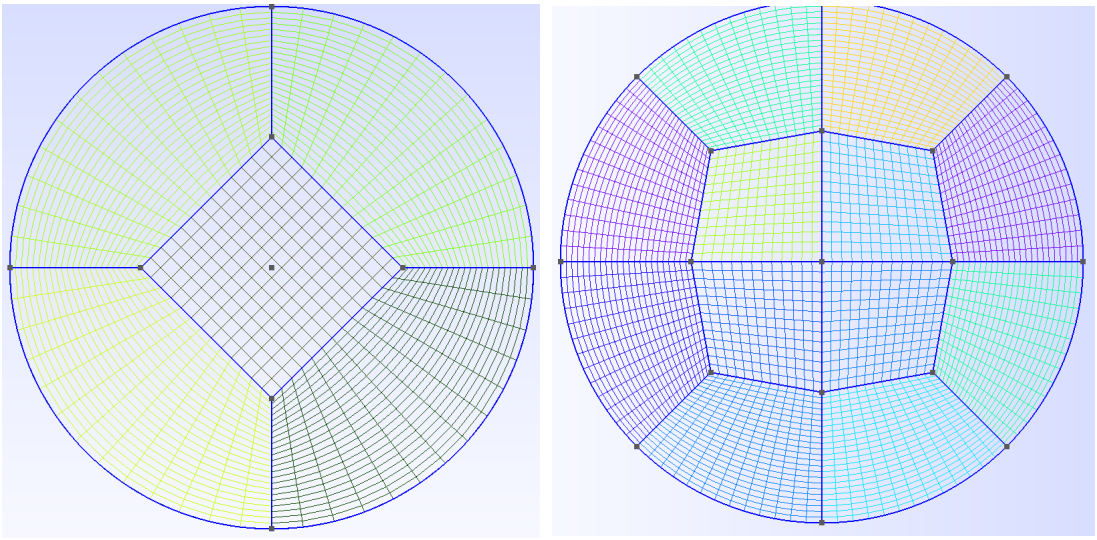


Figure 16: Block-structured cylinder created by using Gmsh.

8 Examples from Gmsh

8.1 2D Pipe

The following 2D example is based on the tutorial [16] and creates a mesh for a pipe with a sudden contraction and sudden expansion. To begin with, the geometric parameters are defined in the Gmsh code.

```
SetFactory("OpenCASCADE");
//OpenCASCADE gives access to usual solid geometry operations;
// ******* Define Geometry Parameters ************************
ls = 5;
Xi = 100;
Xo = 100;
L = 100.0;
x0 = Xi + L/2.0;
R = 50.0;
f0 = 0.5;
Ri = 15;
Z = 5;
```

The points making up the main structure, Figure 17, are defined in the code by their 3D location in space, where *ls* is the size of the points. Straight lines are created by connecting the points in the GUI interface or by declaring new lines in the code. When using the Gmsh GUI, points and lines are label automatically with the next available number in ascending order.

```
// ****** Define Geometry ***********
Point(1) = {0, 0, 0, ls};
Point(2) = {Xi, 0, 0, ls};
Point(3) = {Xi, R, 0, ls};
Point(4) = {0, R, 0, ls};
Point(5) = {Xi + L, 0, 0, ls};
Point(6) = {Xi + L + Xo, 0, 0, ls};
Point(7) = {Xi + L + Xo, R, 0, ls};
Point(8) = {Xi + L, R, 0, ls};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};
Line(5) = {5, 6};
Line(6) = {6, 7};
Line(7) = {7, 8};
Line(8) = {8, 5};
Line(9) = {2, 5};
```

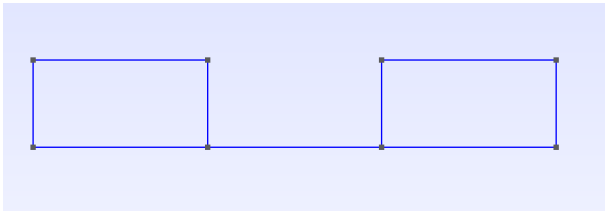


Figure 17: Points and lines defined in Gmsh code.

The curved line is created in the code by using the function $f(x) = R * (1 - f0/2 * (1 + Cos(2.0 * Pi * (x - x0)/L))$, where L is the length of the line, $x0$ is the maximum position and $f0$ is a fraction from 0-1. At this stage, it becomes easier to code the structure than manually adding the points from the GUI. A for-loop is applied to declare all the points along the curved line. The command "newp" returns the next available number that can be used to number points, while "newl" returns the next available number for lines. Finally, a new curved line, "Spline", is created between the new points.

```
pList[0] = 3; // First point label
nPoints = 21; // Number of discretization points
    For i In {1 : nPoints}
      x = Xi + L*i/(nPoints + 1);
      pList[i] = newp; //returns the next available point number.
      Point(pList[i]) = {x,
                    ( R * (1 - f0/2 *(1 + Cos(2.0*Pi * (x-x0)/L) ) )),
                    0,
                    ls};
    EndFor
    pList[nPoints+1] = 8; // Last point label
 Spline(newl) = pList[];
```

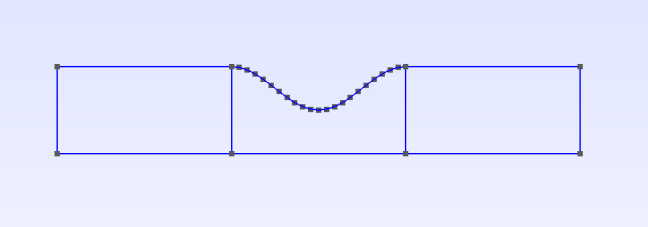


Figure 18: Curved lined added from the Gmsh code.

The next step consists of adding mesh points along the lines to obtain a structured mesh. The function "Transfinite Line" controls the number of mesh points along the lines, while "Ceil" rounds up to the nearest integer. Three plane surfaces can then be created from Gmsh GUI by manually selecting the line loops. "Transfinite Surface" forces the plane surfaces to create a structured triangular mesh, while "Recombine Surface" combines triangles to quadrangles.

```
//create mesh points
Transfinite Line {9, 10} = Ceil(L/ls) Using Progression 1;
Transfinite Line {4, -2, 8, -6} = Ceil(R/ls) Using Progression 1.1;
Transfinite Line {1, 3} = Ceil(Xi/ls) Using Progression 1;
Transfinite Line {5, 7} = Ceil(Xo/ls) Using Progression 1;
//create plane surfaces
Line Loop(1) = {4, 1, 2, 3};
Plane Surface(1) = {1};
Line Loop(2) = {2, 10, 8, -9};
Plane Surface(2) = {2};
Line Loop(3) = {8, 5, 6, 7};
Plane Surface(3) = {3};
Transfinite Surface {1,2,3}; //structured triangles
Recombine Surface {1,2,3};  //combine triangles to quadrangles
```

The "Extrude" function forces the three surfaces to extend in the z-direction to create a 3D mesh.

```
//Create 3D geometry
    Extrude {0,0,Z} {
      Surface{14,12,16}; Layers{1}; Recombine;
    }
Coherence; //remove duplicates lines
Mesh 3;  // Generalte 3D mesh
```

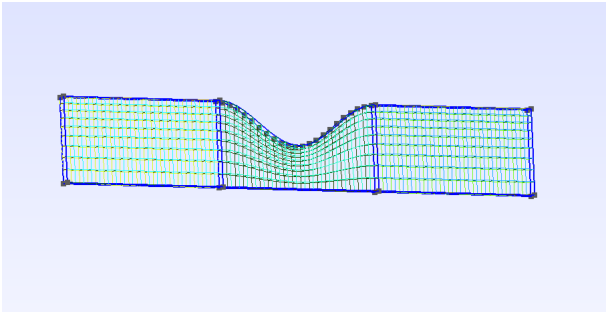


Figure 19: Structured mesh consisting of hexahedron.

The last step consists of declaring the physical boundaries and volumes to OpenFOAM and saving a msh file. The physical boundaries and volumes are read to OpenFOAM when running the command "gmshToFoam pipe2D.msh".

```
//***** Define Physical Geometry ***********
Physical Surface("symmetryLine") = {51, 37, 73};
Physical Surface("frontAndBack") = {60, 38, 82, 16, 14, 12};
Physical Surface("wall") = {59, 29, 81};
Physical Surface("inlet") = {47};
Physical Surface("outlet") = {77};
Physical Volume("volume") = {2, 1, 3};
```

Moving to the OpenFOAM case folder, the boundary patches can be found in the boundary file in the constant/polyMesh folder. The only change that needs to be done in this file, is changing the frontAndBack patch type from "patch" to "empty". This change makes it possible to run the OpenFOAM case in 2D.

```
    frontAndBack
    {
        type            empty; // from 3D to 2D
        nFaces          1026;
        startFace       960;
    }
    wall
    {
        type            patch;
        physicalType    patch;
        nFaces          57;
        startFace       1986;
    }
    symmetry
    {
        type            patch;
        physicalType    patch;
        nFaces          57;
        startFace       2043;
    }
```

```
    inlet
    {
        type            patch;
        physicalType    patch;
        nFaces          9;
        startFace       2100;
    }
    outlet
    {
        type            patch;
        physicalType    patch;
        nFaces          9;
        startFace       2109;
    }
```

The next step is to define the inlet conditions for all the boundary patches in the 0-folder. Listed below are the pressure inlet conditions in the p-file.

```
dimensions      [0 2 -2 0 0 0 0]; //units
internalField   uniform 0;
boundaryField
{
inlet
    {
        type            zeroGradient;

    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    wall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;

    }
    symmetry{
       type             zeroGradient;
   }
}
```

The inlet conditions for the velocity are specified in the u-file.

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
inlet
    {
        type            fixedValue;
        value           uniform (1 0 0);

    }

    outlet
    {
        type            zeroGradient;

    }

    wall
    {
        type            fixedValue;
        value           uniform (0 0 0);

    }

    frontAndBack
    {
        type            empty;

    }
    symmetry{
       type             slip;
   }
}
```

Once the boundaries are declared, the solver command "icoFoam" can be run in the terminal. When the simulation is completed, the command "touch pipe2D.touch" generates a file that can be opened in ParaView, where the pressure profile, Figure (20), and the velocity profile, Figure (21), can be visualized.

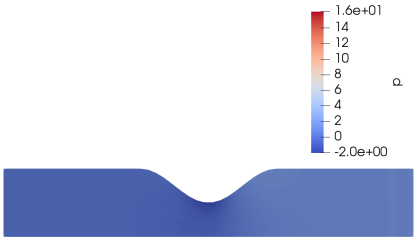


Figure 20: Pressure profile

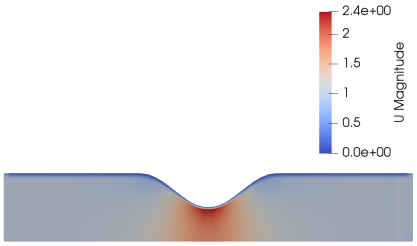


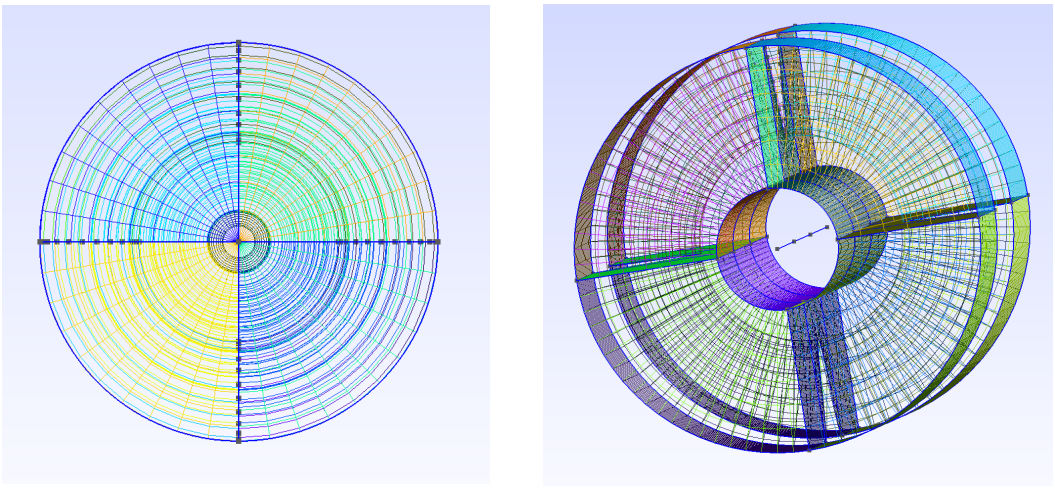
Figure 21: Velocity profile

8.2 3D Pipe

The previous example was modified for a 3D simulation. Instead of using the Extrude function in the z-direction, the function was used to rotate each surface 90 degrees, four times. The rotation is performed in stages since Gmsh cannot rotate the geometry the whole angle of 360 degree in one step. The notation Extrude $\{\{1,0,0\},\{0,0,0\},Pi/2\}$, specify the x-axis as the axis of revolution, by specifying a vector parallel to it, $\{1,0,0\}$, and any point on the axis, $\{0,0,0\}$, while $Pi/2$ sets the rotation angle.

```
// Rotate 360 degrees to create 3D geometry
For i In {1:3}
out1[]=Extrude{{1,0,0},{0,0,0},Pi/2}{Surface{i};Layers{20};Recombine;};
out2[]=Extrude{{1,0,0},{0,0,0},Pi/2}{Surface{out1[0]};Layers{20};Recombine;};
out3[]=Extrude{{1,0,0},{0,0,0},Pi/2}{Surface{out2[0]};Layers{20};Recombine;};
out4[]=Extrude{{1,0,0},{0,0,0},-Pi/2}{Surface{i};Layers{20};Recombine;};
EndFor
```

By writing "out1[]=Extrude..", out1[0] will contain the "top" of the newly created surface, that can be extruded in the next step. Figure 22(a) shows that all the mesh cells lead towards the center of the cylinder, creating a singularity where the solution will not converge. To avoid the singularity problem, the block-structured mesh approach is used. The initial points and lines are re-specified to create a hole in the cylinder. The new geometry is shown in Figure 22(b).



(a) (b)

Figure 22: Cylinder mesh (a) with a singularity in the center and without a singularity (b).

The inner cylinder needs to be filled with a new mesh to recreate the cylinder. This is achieved by adding a plane surface at one of the ends of the cylinder. The surface is divided into quadrilateral subsurfaces in order to create block-structured meshes.

```
//Declare points on inner circle
p1 = newp; Point(p1) = {Xi+L+Xo,Ri*Cos(Pi/4),Ri*Sin(Pi/4),ls};
p2 = newp; Point(p2) = {Xi+L+Xo,Ri*Cos(Pi/4),-Ri*Sin(Pi/4),ls};
p3 = newp; Point(p3) = {Xi+L+Xo,-Ri*Cos(Pi/4),Ri*Sin(Pi/4),ls};
p4 = newp; Point(p4) = {Xi+L+Xo,-Ri*Cos(Pi/4),-Ri*Sin(Pi/4),ls};

p5 = newp; Point(p5) = {Xi+L+Xo,(Ri/2)*Cos(Pi/4),(Ri/2)*Sin(Pi/4),ls};
p6 = newp; Point(p6) = {Xi+L+Xo,(Ri/2)*Cos(Pi/4),(-Ri/2)*Sin(Pi/4),ls};
p7 = newp; Point(p7) = {Xi+L+Xo,(-Ri/2)*Cos(Pi/4),(Ri/2)*Sin(Pi/4),ls};
p8 = newp; Point(p8) = {Xi+L+Xo,(-Ri/2)*Cos(Pi/4),(-Ri/2)*Sin(Pi/4),ls};

//Declare lines on inner square
l1= newl; Line(l1) = {p7, p5};
l2= newl; Line(l2) = {p5, p6};
l3= newl; Line(l3) = {p6, p8};
l4= newl; Line(l4) = {p8, p7};
Transfinite Line{l1,l2,l3,l4}=20+1;

l5 = newl; Line(l5) = {p3, p7};
l6 = newl; Line(l6) = {p1, p5};
l7 = newl; Line(l7) = {p8, p4};
l8 = newl; Line(l8) = {p6, p2};
Transfinite Line{l5,l6,l7,l8}=3;

c1 = newc; Circle(c1) = {p1, 6, p3};
c2 = newc; Circle(c2) = {p3, 6, p4};
c3 = newc; Circle(c3) = {p4, 6, p2};
c4 = newc; Circle(c4) = {p1, 6, p2};
Transfinite Line{c1,c2,c3,c4}=20+1;

ll1 = newll; Line Loop(ll1) = {l1,l4,l3,l2};
ll2 = newll; Line Loop(ll2)={l6,c1,l5,l1};
ll3 = newll; Line Loop(ll3) = {l5,c2,l7,l4};
ll4 = newll; Line Loop(ll4) = {l8,l3,l7,c3};
ll5 = newll; Line Loop(ll5) ={l6,l2,l8,c4};

s1 = news; Plane Surface(s1)={ll1};
s2 = news; Plane Surface(s2) = {ll2};
s3 = news; Plane Surface(s3) = {ll3};
s4 = news; Plane Surface(s4)={ll4};
s5 = newc; Plane Surface(s5) = {ll5};

Transfinite Surface{s1,s2,s3,s4,s5};
Recombine Surface{s1,s2,s3,s4,s5};
```

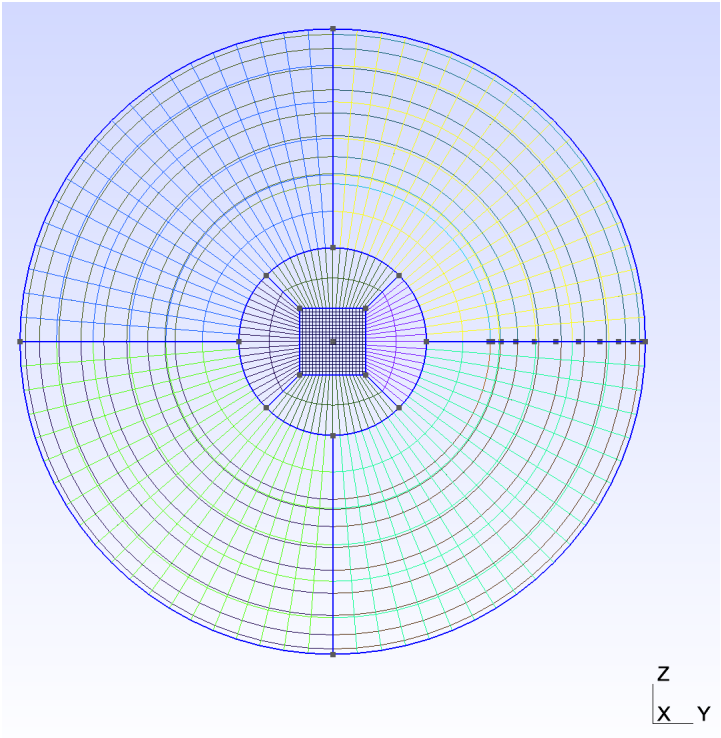


Figure 23: Structured inner cylinder.

The block-structured surface, Figure 23, is then extruded along the pipe in the x-direction. The extrusion function is used three times on all the five subsurfaces in order to get the correct number of layers. The transition between blocks should be smooth when combining different structured subdomains. OpenFOAM may otherwise believe that there is a surface between different subregions if there is a sudden jump in the mesh.

```
For i In{s1:s5}
out5[] = Extrude{-Xo,0,0}{Surface{i};Layers{Ceil(Xo/ls)-1};Recombine;};
out6[] = Extrude{-L,0,0}{Surface{out5[0]};Layers{Ceil(L/ls)-1};Recombine;};
out7[] = Extrude{-Xi,0,0}{Surface{out6[0]};Layers{Ceil(Xi/ls)-1};Recombine;};
EndFor

//********* Define Physical Geometry (this is for OpenFOAM) *************
Physical Surface("inlet") = {19, 14, 9, 182, 197, 212, 152, 167, 144, 4};
Physical Surface("outlet") = {148, 48, 26, 41, 31, 36, 137, 136, 135,
134, 133};
Physical Surface("walls") = {135, 40, 25, 35, 30, 55, 60, 50, 45, 22, 7,
12, 17};
Physical Volume("volume") = {6, 19, 22, 25, 13, 16, 5, 7, 8, 10, 9, 11, 12,
2, 1, 3, 4, 24, 27, 15, 21, 18, 20, 23, 14, 26, 17};

Mesh 3;  // Generalte 3D mesh
Coherence Mesh;  // Remove duplicate entities
```

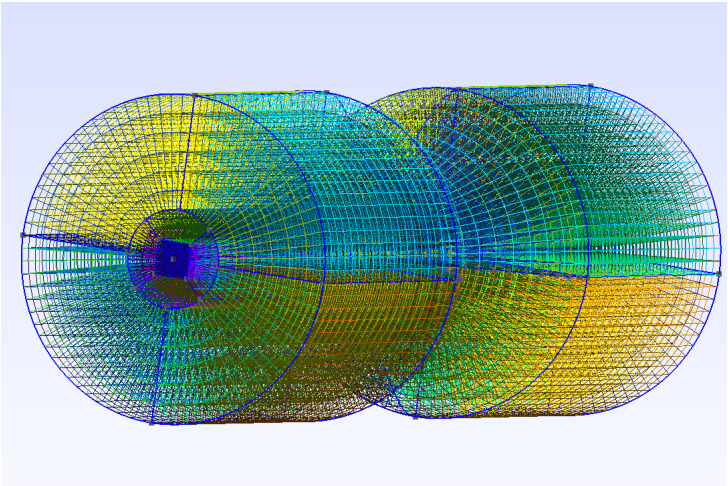


Figure 24: The final 3D mesh.

The final 3D mesh, Figure 24, can be thereafter be added to a OpenFOAM case folder, converted to OpenFOAM format and simulated by the same procedure as the previous example.

9 Conclusion

In conclusion, mesh quality is generally measured by four metrics, which are non-orthogonality, skewness, aspect ratio and smoothness. There are several ways to create meshes for OpenFOAM simulations. It is of the authors opinion that Gmsh is the most user-friendly method for structured hexahedron meshes. However, automatic generation methods, such as the utility snappyHexMesh, is needed for high complex geometries.

References

[1] C. Soccol, A. Pandey, and C. Larroch. *Fermentation Processes Engineering in the Food Industry*. CRC Press, 2013.

[2] J. Slotnick, A. Khodadoust, and A Alonso. "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences". In: *NASA/CR-2014-218178* (2014).

[3] H. G. Weller et al. "A Tensorial Approach to Computational Continuum Mechanics Using Object-oriented Techniques". In: *Comput. Phys.* 12.6 (Nov. 1998), pp. 620–631. ISSN: 0894-1866. DOI: 10.1063/1.168744. URL: http://dx.doi.org/10.1063/1.168744.

[4] C Geuzaine and Jçois Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. ISSN: 1097-0207. DOI: 10.1002/nme.2579. URL: http://dx.doi.org/10.1002/nme.2579.

[5] M et al. Wang. "Mesh partitioning using matrix value approximations for parallel computation fluid dynamics simulation". In: *Advances in Mechanical Engineering* 9 (2017).

[6] *OpenFOAM v5 User Guide*. URL: https://cfd.direct/openfoam/user-guide.

[7] *Mesh Topologies, ANSYS FLUENT*. URL: http://www.afs.enea.it/project/neptunius/docs/fluent/html/ug/node161.htm.

[8] A. Moraes and P. Lage. "Analysis of the non-orthogonality correction of finite volum discretization on unstructured meshes". In: Brazil, 2013.

[9] *The Finite Element Method*. URL: https://www.comsol.com/multiphysics/finite-element-method (visited on 11/2017).

[10] V. Ranade. *Computational Flow Modeling for Chemical Reactor Engineering*. ACADAMIC PRESS, 2002.

[11] F. Moukalled, L. Mangani, and M Darwish. *The Finite Volume Method in Computational Fluid Dynamics, An Advanced Introduction with OpenFOAM® and Matlab*. Springer, 2016.

[12] Dimitri J. *Handbook of Computational Fluid Mechanics*. Academic Press, 1996. Chap. Chapter 7: Mesh generation and adaptivity for complex geometries and flows, pp. 417–459.

[13] J Rhoads. *OpenFOAM Workshop 2014: Effects of grid quality on solution accuracy*. 2014.

[14] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: the finite volume method*. Longman Scientific & Technical Publisher, 1995.

[15] P.A Durbin. *Fluid Dynamics with a Computational Perspective*. Cambridge University Press; Reprint edition, 2014.

[16] *2D Mesh Tutorial using GMSH*. URL: http://openfoamwiki.net/index.php/2D_Mesh_Tutorial_using_GMSH.