Introductory meshing course with OpenFOAM

Sánchez-Cruz Ángel

June 2024

Contents

Contents

Chapter 1

What is OpenFOAM and why learn to use it?

OpenFOAM (Open Field Operation and Manipulation) is an open-source software package used for computational fluid dynamics (CFD) and other types of simulations. It provides a wide range of solvers for simulating complex physical phenomena, such as fluid flow, heat transfer, chemical reactions, solid mechanics, electromagnetism, particle dynamics, multiphase flow, etc.

OpenFOAM is highly customizable and allows users to modify and extend its functionality through its extensive C++ library. Users can create new solvers, utilities, and libraries tailored to specific simulation needs by leveraging OpenFOAM's object-oriented design. The source code is openly available, enabling modifications to existing solvers and the development of custom applications. This flexibility makes OpenFOAM suitable for a wide range of applications and research areas, allowing for advanced and specialized simulations.

1.1 Download alternatives

https://openfoam.org/

Figure 1.1: OpenFOAM.org logo

Is the primary source for free and open-source downloads of OpenFOAM. It provides access to standard versions of OpenFOAM and official documentation.

https://www.openfoam.com

Figure 1.2: OpenFOAM.com logo

Offers downloads of OpenFOAM and also provides additional commercial services related to support and implementation of OpenFOAM. It's ideal if you're looking for professional support options and training.

https://www.cfdsupport.com/openfoam-for-windows.html

Figure 1.3: CFD Support logo

This site offers a version of OpenFOAM specifically designed for Windows users. You can download OpenFOAM here and explore its functionality within the Windows environment.

Chapter 2

Installation of OpenFOAM in Windows

This chapter guides readers step-by-step through the process of installing OpenFOAM, using the CFD support page as the primary resource. It covers everything from system prerequisites to downloading the software and initial setup. Readers will learn how to configure the necessary development environment and perform platform-specific configurations. Additionally, useful tips and solutions to potential installation issues are provided, ensuring that users can effectively and efficiently start using OpenFOAM.

2.1 Basic System Requirements:

To efficiently install and run OpenFOAM, it is important to have a computer that meets certain basic requirements. Here are the recommended requirements, though they do not necessarily need to be met:

- Operating System:
  - Linux: OpenFOAM is primarily developed for Linux. The most commonly used distributions are Ubuntu and CentOS.

  - **Windows**/Mac: OpenFOAM can run on Windows and Mac through Docker containers or virtual machines, although using Linux is recommended for optimal performance.

- Processor (CPU):
  - A multi-core processor, preferably with 4 cores or more. OpenFOAM can take advantage of multi-core architecture to parallelize computations and improve performance.

- Memory (RAM):
  - Minimum: 8 GB of RAM.
  - Recommended: 16 GB or more, especially for larger and more complex problems.

- Storage:
  - Disk Space: At least 20 GB of free space for software installation and simulation files. More space may be required depending on the size of the cases being run.
  - An SSD is preferable to improve read/write speeds, which can accelerate simulations and handling of large files.

- Graphics Card (optional):
  - Not strictly necessary, but a dedicated GPU can be useful if you plan to perform complex result visualizations or use hardware acceleration for certain operations.

Notes

OpenFOAM is an open-source software widely used in the computational fluid dynamics community, offers different download pages for various reasons, including stable and development versions, different distributions maintained by various organizations, and customized versions tailored to specific needs.

Keys and commands to use

- CTRL key + up scroll Increases the font size in the console.
- CTRL key + down scroll Decreases the font size in the console.
- TAB key Auto-completes the information, ensuring that we are working in the correct directory or using the correct command.
- ENTER key Used to send commands and confirm actions.
- cd Change directory, allows navigation forward in the directory structure by specifying the name of a directory you want to move into.
- cd .. Navigates backward in the directory structure, moving you to the parent directory of the current directory.
- clear Clear all previous lines of text in the terminal.
- ls List the contents of a directory.
- paraFoam.exe Opens ParaView software and load the current project.
- blockMesh.exe Creates the numerical mesh that defines the geometry of the simulation domain.
- surfaceFeatures.exe Extract geometric features from a surface, such as contour lines or sharp edges, which are important for defining the geometry and mesh quality.
- snappyHexMesh.exe This command runs the meshing process.
- snappyHexMesh > snappyHexMesh.txt 2>&1 This command runs the meshing process and all the information that would normally be displayed in the terminal (both standard output and error messages) will be saved in "snappyHexMesh.txt".

Keys and commands to use

- checkMesh.exe Used to verify the quality of generated the mesh for a CFD simulation.
- checkMesh > checkMesh.log Writes a log file with the checkMesh instruction.
- decomposePar.exe Decompose a computational domain into smaller subdomains, allowing a simulation case to run in parallel across multiple processors.
- mpiexec.exe -np X snappyHexMesh -parallel Is used to run the snappyHexMesh meshing utility in parallel across X processors.
- mpiexec.exe -np X snappyHexMesh -parallel > snappyHexMesh.log Is used to run the snappyHexMesh meshing utility in parallel across X processors and writes a log file in the directory folder.
- reconstructParMesh.exe Is used to reconstruct a single, complete mesh from multiple decomposed subdomains after a parallel meshing or simulation process.
- reconstructParMesh.exe -latestTime Is used to reconstruct the mesh corresponding to the latest time step from a parallel simulation or meshing process.

2.2 From CFD Support

CFD SUPPORT is a private engineering firm founded in 2009 by two engineers and PhD graduates from the Von Karman Institute for Fluid Dynamics in Prague, Czech Republic. Specializing in engineering simulations, the company operates at the intersection of physics, mathematics, and software engineering. Their team comprises skilled engineers, mathematicians, and programmers dedicated to meeting the diverse demands of computational fluid dynamics (CFD) through collaborative expertise and flexible solutions [1].

Figure 2.1: CFD Support logo

1. Access to CFD Support, go to Resources / Download Software / OpenFOAM for windows and fill in the required fields to enable the download button.
2. Decompress the software, run as administrator, perform the default installation without modifying check-boxes.
3. Verify the installation on your PC at This computer / C: / OpenFOAM / 20.09 / User-dev / run.
4. Place the shortcut icon for the software in an accessible location, launch it, and confirm access to the "run".
5. Finally, use the command "paraFoam.exe" to open ParaView.

Notes

This software is a revised version, so it is configured to install the necessary packages.

Chapter 3

Basic folder structure for simulations in OpenFOAM

```
MyCase (working directory)/
|-- 0/
|   |-- U
|   |-- p
|   |-- nut, k, epsilon, etc
|-- constant/
|   |-- polyMesh/
|   |   |-- boundary
|   |   |-- points, faces, etc.
|   |-- transportProperties
|   |-- turbulenceProperties
|-- system/
|   |-- controlDict
|   |-- decomposeParDict
|   |-- fvSchemes
|   |-- fvSolution
|   |-- snappyHexMeshDict
|   |-- some functions ...
```

|----------

Recommendation

- Creating cases at 1:1 scale, meaning using the real dimensions of the object or system being simulated, has several advantages compared to using smaller scales. Here are some of the main benefits:
  - **Boundary Conditions:** Boundary conditions and flow properties can be applied more accurately since no scaling is needed.
  - **Realistic Results:** The simulation results tend to be more realistic and directly applicable to real-world situations.
  - **Proportionality:** Working at a 1:1 scale avoids proportionality issues that can arise when scaling dimensions, material properties, and boundary conditions.
  - **No Adjustments Needed:** There's no need to adjust or transform the simulation results to match real dimensions.
  - **Ease of Implementation:** Defining geometry, material properties, and boundary conditions is more straightforward without additional conversions or calculations for scaling.
  - **Experimental Validation:** It facilitates the comparison and validation of simulation results with experimental data or real-world observations.
  - **Reproducibility:** Results can be more easily reproduced and verified by other researchers or engineers working on the same problem.
  - **Detailed Resolution:** It allows capturing detailed and complex flow characteristics that might be lost when using smaller scales.
- However, it's important to note that working at a 1:1 scale can also increase computational requirements due to the need for a finer mesh to capture all the flow details and geometries. This can lead to longer computation times and greater hardware resources. Therefore, balancing the need for accuracy with available computational resources is crucial.

Chapter 4

Introduction to SnappyHexMesh utility

SnappyHexMesh is a three-dimensional (3D) mesh generation tool used in computational fluid dynamics (CFD) and is part of the open-source software OpenFOAM (Open Field Operation and Manipulation).

4.1 Features of SnappyHexMesh:

- **Automatic Mesh Generation:** SnappyHexMesh generates meshes from CAD (Computer-Aided Design) geometries, allowing a more automated process compared to other meshing tools.

- **Hexahedral Meshes:** The name "HexMesh" refers to the ability to generate meshes predominantly composed of hexahedral elements, which are cubes or rectangular blocks in 3D.

- **Local Refinement:** It allows local refinement of the mesh, meaning smaller and more detailed elements can be created in areas of specific interest, such as near complex surfaces or in regions where large flow gradients are expected, maintaining the integrity of the original geometry.

- **Use of Phases:** The mesh generation process in SnappyHexMesh is carried out in several phases:
  - **0. Background mesh:** An initial coarse mesh, forming the base for further refinement.
  - **1. Castellated mesh:** Refines the background mesh and cuts cells to match the geometry's outer shape, identifying shared cells.
  - **2. Snapping (internal or external flow):** Adjusts the mesh to closely fit the surface of the geometry, improving accuracy, depending on whether internal or external flow will be studied.
  - **3. Add layers:** Adds prismatic boundary layers to the mesh to better capture near-wall flow details.

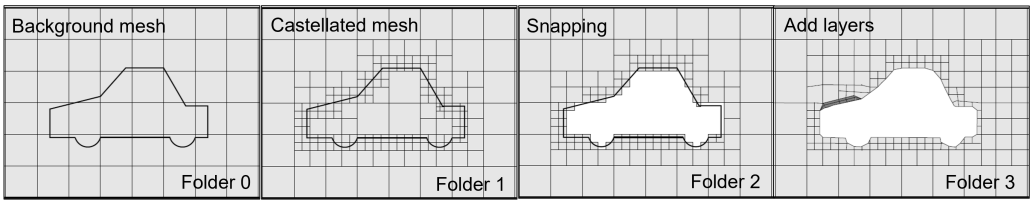


Figure 4.1: Meshing process [2].

Each phase of the process generates a folder, and each folder contains a subfolder named "polymesh", which houses the files comprising the mesh from that particular phase.

The number of phases required depends on the complexity of the mesh being generated.

- **Compatibility with OpenFOAM:** Being part of OpenFOAM, meshes generated with SnappyHexMesh integrate seamlessly with OpenFOAM's CFD solvers, facilitating the setup and execution of simulations.
- **Import and export meshes:** SnappyHexMesh allows importing geometries and meshes in various standard formats such as STL, facilitating interoperabil-

ity with other simulation software. This capability is crucial for using externally generated complex geometries or exporting meshes generated in SnappyHexMesh to other simulation environments, such as ANSYS Fluent or other solvers that support these formats.

- **Parallelization:** SnappyHexMesh and OpenFOAM are designed to leverage parallel computing capabilities, which significantly accelerates the mesh generation and CFD simulations on systems with multiple CPU cores or computer clusters. Parallelization in SnappyHexMesh distributes the mesh generation workload across multiple processors or nodes, enhancing efficiency and reducing runtime for complex problems and large datasets.

4.2 Basic folder structure for mesh creation

Here is the basic structure for creating meshes in SnappyHexMesh:

```
ExampleMeshX (working directory)/
|-- 0/
|-- constant/
|   |-- triSurface/
|   |   |-- STL files.
|-- system/
|   |-- blockMeshDict
|   |-- controlDict (Essential, not modified for BGM)
|   |-- decomposeParDict (Not essential, necessary to parallelization)
|   |-- snappyHexMeshDict
|   |-- surfaceFeaturesDict
|   |-- fvSchemes   (Essential, not modified for BGM)
|   |-- fvSolution  (Essential, not modified for BGM)
|   |-- meshQualityDict
|----------------------
```

Folders "0", "constant" and "system" are essential to perform any process in OpenFOAM.

Here is an example of a script header:

```
/*--------------------------------*- C++ -*----------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: Source
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "FolderLocation";
    object      SpecificFileToModify;
}
```

In OpenFOAM file headers, what typically changes are the specific details of the class, location and object, while the general structure of the comment and the information about OpenFOAM remain consistent.

Chapter 5

Case 1: Meshing of a hexahedron inside a wind tunnel

5.1 Creating a uni-block Background Mesh

The Background mesh (BGM) refers to the initial mesh that serves as the foundation for the detailed mesh generation process in tools like SnappyHexMesh. This stage establishes a basic structure onto which additional transformations and refinements are applied to capture the complex geometry of the simulation domain. It is crucial for ensuring an accurate representation of the domain in CFD simulations, providing an initial platform upon which more advanced phases such as "castellated mesh", "snap", and "add layers" are built.

5.1.1 Steps to generate a uni-block Background mesh

1. Create a folder named "ThisMeshingCourse" inside the "run" folder: This computer / C: / OpenFOAM / 20.09 / User-dev / run / ThisMeshingCourse. In this folder we will save the meshes of this course (ExampleMesh1, ExampleMesh2, etc...).

2. Modify the "blockMeshDict" file, located in sub-folder "system", according to Figure 5.1 and specifying desired patches to bound-

ary conditions [3].

- **Consider the origin of geometry as the BGM origin.**
- **The vertex order must follow the counterclockwise convention.**

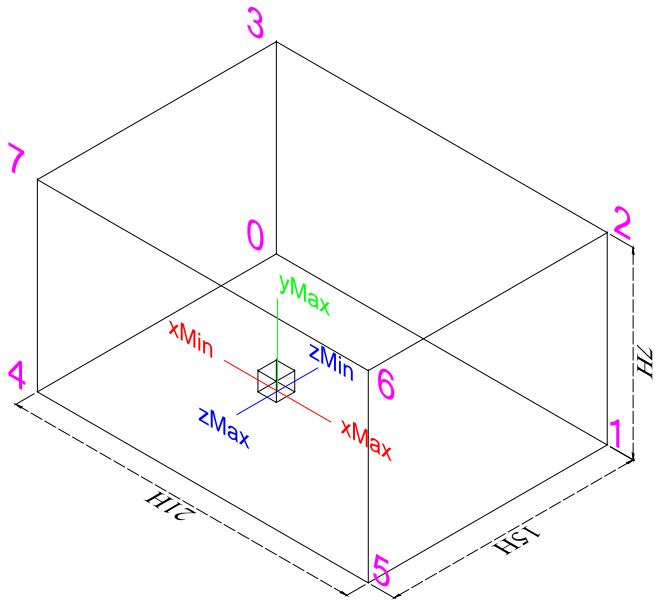


Figure 5.1: Uni-block Background Mesh sketch.

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
#inputSyntax slash;

backgroundMesh
{
    xMin   -1.1;
    xMax   3.1;
    yMin   0;
    yMax   1.4;
    zMin   -1.5;
    zMax   1.5;
    xCells xxx;  // Number of divisions on the x-axis
    yCells yyy;  // Number of divisions on the y-axis
    zCells zzz;  // Number of divisions on the z-axis
}

convertToMeters 1; // Scaling factor

vertices /* 0,1,2,3,4,5,6,7.  $! allows for the interpolation of
variables, as well as the inclusion of mathematical expressions
or references to other variables. */
(
  ($!backgroundMesh/xMin $!backgroundMesh/yMin $!backgroundMesh/zMin)
  ($!backgroundMesh/xMax $!backgroundMesh/yMin $!backgroundMesh/zMin)
  ($!backgroundMesh/xMax $!backgroundMesh/yMax $!backgroundMesh/zMin)
  ($!backgroundMesh/xMin $!backgroundMesh/yMax $!backgroundMesh/zMin)
  ($!backgroundMesh/xMin $!backgroundMesh/yMin $!backgroundMesh/zMax)
  ($!backgroundMesh/xMax $!backgroundMesh/yMin $!backgroundMesh/zMax)
  ($!backgroundMesh/xMax $!backgroundMesh/yMax $!backgroundMesh/zMax)
  ($!backgroundMesh/xMin $!backgroundMesh/yMax $!backgroundMesh/zMax)
);

blocks //Segments per axis
(
    hex (0 1 2 3 4 5 6 7)
    (
        $!backgroundMesh/xCells
        $!backgroundMesh/yCells
        $!backgroundMesh/zCells
    )
    simpleGrading (1 1 1) /* It is a tool for adjusting the density of the
    structured mesh, allowing adaptation of spatial resolution according
    to specific simulation requirements.
    It will be modified in future sections. */
);

boundary
(
    inlet
```

```
64      {
65          type patch; // See basic OF concepts chapter
66          faces
67          (
68              (0 3 7 4)
69          );
70      }
71
72      outlet
73      {
74          type patch;
75          faces
76          (
77              (1 5 6 2)
78          );
79      }
80
81      ground
82      {
83          type wall; // See basic OF concepts chapter
84          faces
85          (
86              (0 1 5 4)
87          );
88      }
89
90      boundaryright
91      {
92          type symmetry; // See basic OF concepts chapter
93          faces
94          (
95              (4 7 6 5)
96          );
97      }
98
99       boundaryleft
100     {
101         type symmetry;
102         faces
103         (
104             (0 1 2 3)
105         );
106     }
107
108      boundarytop
109     {
110         type symmetry;
111         faces
```

```
        (
            (3 2 6 7)
        );
    }
);

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

3. •In the working directory, execute the command "blockMesh.exe".

   •Use the "paraFoam.exe" command to visualize the BGM, explore the "Mesh Regions" in the "Properties" panel, visualize the "Axes Grid" and use "Camera Parallel Projection" in the "View" panel and enable "Surface With Edges" in the "Display" panel.

   •A new folder named "polymesh" has been created within the "constant" sub-folder. Here is the "boundary" file, contained in the polymesh folder.

   •To modify the BGM, it is necessary to delete the "polymesh" folder and repeat the previous step.

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |

   OpenFOAM for Windows 20.09 (v1)
   Built by CFD Support, www.cfdsupport.com (based on Symscape).
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

6
```

```
(
    inlet
    {
          type            patch;
        nFaces          2625;
        startFace       812700;
    }
    outlet
    {
        type            patch;
        nFaces          2625;
        startFace       815325;
    }
    ground
    {
        type            wall;
        inGroups        List<word> 1(wall);
        nFaces          7875;
        startFace       817950;
    }
    boundaryright
    {
        type            symmetry;
        inGroups        List<word> 1(symmetry);
        nFaces          3675;
        startFace       825825;
    }
    boundaryleft
    {
        type            symmetry;
        inGroups        List<word> 1(symmetry);
        nFaces          3675;
        startFace       829500;
    }
    boundarytop
    {
        type            symmetry;
        inGroups        List<word> 1(symmetry);
        nFaces          7875;
        startFace       833175;
    }
)

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

Play around with the following parameters:

- **xCells, yCells and zCells**

- **simpleGrading (It will be reviewed in future sessions)**

- **Create a copy of the main mesh folder, so as not to lose progress.**

Notes

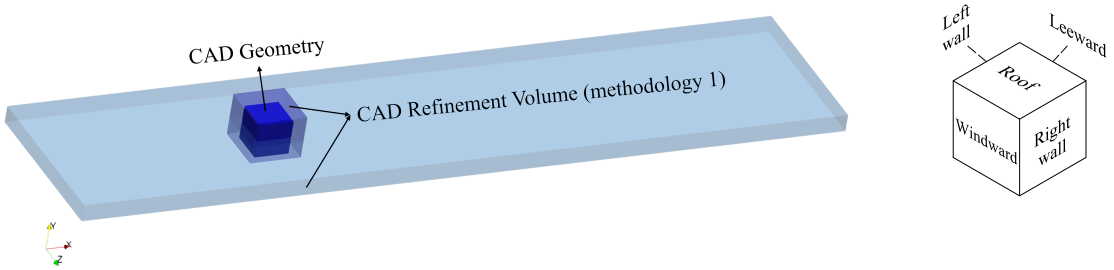
- The working directory is the place where all the files and directories necessary to run a simulation are stored.

Recommendation

- Avoid creating folders or files with spaces or special characters such as: ! @ # $ % ^ & * ( ) [ ] { } ; ' " , < > ? This can potentially cause issues with certain commands and scripts within the OF environment.

5.2 Importing geometry into OpenFOAM

The extraction of surface features from CAD geometry is crucial for generating high-quality meshes that accurately capture important geometric details, thereby improving the fidelity of simulations.



(a) CAD geometry and refinement volumes (b) Surfaces

Figure 5.2: CAD geometry and refinement volumes and surface definition

5.2.1 Steps to import geometry into OpenFOAM

1. According to the diagram in section 4.2, in "constant", sub-folder "triSurface", place the geometry files in STL format inside: This computer / C: / OpenFOAM / 20.09 / User-dev / run / ThisMeshingCourse / ExampleMeshX / constant / triSurface, to convert the geometry into a format that OpenFOAM can read.

2. Modify the "surfaceFeaturesDict" file, located in sub-folder "system".

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  dev
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      surfaceFeaturesDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
16
17 surfaces // STL files representing specific parts of the geometry
18 (
19   "Barlovento.stl"
20   "Sotavento.stl"
21   "Cubierta.stl"
22   "MDerecho.stl"
23   "MIzquierdo.stl"
24   "Solid1.stl"
25   "Vref1.stl"   // Refinement volume, methodology 1
26   "Vref2.stl"   // Refinement volume, methodology 1
27   "Vref3.stl"   // Refinement volume, methodology 1
28   "Vref4.stl"   // Refinement volume, methodology 1
29   "Vreftop.stl" // Refinement volume, methodology 1
30 );
31
32 includedAngle 150; /* Sets a criterion for identifying geometric features
33 on those surfaces based on the angle between adjacent faces */
34
35 // ************************************************************************* //
```

3.

- In the working directory, execute the command "surfaceFeatures.exe".
- A new folder named "extendedFeatureEdgeMesh" has been created within the "constant" sub-folder. Here are whole parts of geometry.
- Verify that in the "triSurface" folder, ".eMesh" files have been created for the whole parts of geometry.
- **To add more parts of geometry, it is necessary to delete the folder "extendedFeatureEdgeMesh" and ".eMesh" files in "triSurface" and repeat the previous step.**

Notes

- The names assigned to each part of the mesh must be consistent in the scripts and other files.

Recommendation

- From the CAD software, export STL files in meters, ASCII format, and preserve the origin point of the geometry in the global space (*do not convert STL output data to positive space*) SolidWorks example

5.3 snappyHexMeshDict file

The snappyHexMeshDict file is a configuration file used by OpenFOAM's snappyHexMesh utility to generate complex 3D meshes from CAD geometries. It defines various meshing parameters and controls, including geometry input, castellated mesh settings, snapping settings, and boundary layer addition. The file also includes quality control criteria and flags for additional outputs. Key sections often include geometry, castellatedMeshControls, snapControls, and addLayersControls, as shown in Fig. 4.1, each specifying detailed settings for different stages of the meshing process. **Proper configuration of this file is crucial for creating high-quality meshes tailored to specific simulation needs.**

5.3.1 Steps to modify the snappyHexMeshDict file without surface layers

1. Enable or disable mesh processes  as follow:

```
/*--------------------------------*- C++ -*----------------------------------*\
=========                 |
\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
 \\    /   O peration     | Website:  https://openfoam.org
  \\  /    A nd           | Version:  dev
   \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
 version     2.0;
 format      ascii;
 class       dictionary;
 object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Which of the steps to use?
castellatedMesh true;  //or false
snap            true;  //or false
addLayers       false; //or true
```

2. Define the surfaces and volumes:

The "geometry" section in the snappyHexMeshDict file is a **crucial part of the meshing process in OpenFOAM's "snappyHexMesh" utility.** It defines all the surface meshes that will be used to generate the computational mesh. Each entry specifies an STL file, indicating a part of the geometry to be meshed, along with its type ("triSurfaceMesh") and a user-friendly name for easy reference. This section ensures that snappyHexMesh can accurately interpret and utilize the provided geometrical surfaces, forming the foundation for the mesh generation. Properly configuring this section is essential for achieving a high-quality mesh tailored to your simulation needs.

The following code shows how to define parts of geometry refinements and refinement volumes (the first of two methodologies for creating refinement volumes):

```
geometry // Definition of all surfaces to be worked on.
{
 Solid1.stl
 {
  type triSurfaceMesh;
  name Solid1;
 }
 Barlovento.stl
 {
  type triSurfaceMesh;
  name Barlovento;
 }
 Sotavento.stl
 {
  type triSurfaceMesh;
  name Sotavento;
 }
 Cubierta.stl
 {
  type triSurfaceMesh;
  name Cubierta;
 }
 MIzquierdo.stl
 {
  type triSurfaceMesh;
  name MIzquierdo;
 }
 MDerecho.stl
 {
  type triSurfaceMesh;
  name MDerecho;
 }
 Vref1 // Methodology 1
 {
  type triSurfaceMesh;
  file "Vref1.stl";
 }
 Vref2 // Methodology 1
 {
  type triSurfaceMesh;
```

```
    file "Vref2.stl";
   }
   Vref3 // Methodology 1
   {
    type triSurfaceMesh;
    file "Vref3.stl";
   }
   Vref4 // Methodology 1
   {
    type triSurfaceMesh;
    file "Vref4.stl";
   }
   Vreftop // Methodology 1
   {
    type triSurfaceMesh;
    file "Vreftop.stl";
   }
   searchableBox_1 // Methodology 2
   {
    type searchableBox;
    min (-0.3 0 -0.3);
    max ( 0.3 1.4 0.3);
   }
  };
```

3. Modify the castellatedMeshControls and snapControls:

This part of snappyHexMesh file is responsible for defining how the initial background mesh is refined and adjusted based on the specified geometry to create a structured mesh. Key parameters include limits on the number of cells, refinement criteria, and specifications for capturing geometric features. By setting these controls, you can ensure that the mesh accurately represents complex geometries while maintaining computational efficiency.

- **Mesh refinement.**

  Refinement refers to the process of adjusting the mesh of a simulation domain to improve resolution in specific areas.

This adjustment can be crucial for capturing fine details of the flow and obtaining more accurate results in simulations. Refinement can be applied globally to the entire mesh or locally to specific regions where important phenomena are expected to occur.

In OpenFOAM, there are several methods for mesh refinement, most common are:

- **Global refinement:** Uniformly increasing the resolution of the entire mesh.
- **Local refinement:** Increasing the resolution in specific regions of the simulation domain, such as near walls, in areas with large gradients of variables of interest, or around complex geometries.

Figure 5.3 shows a schematic of the refinement principle.

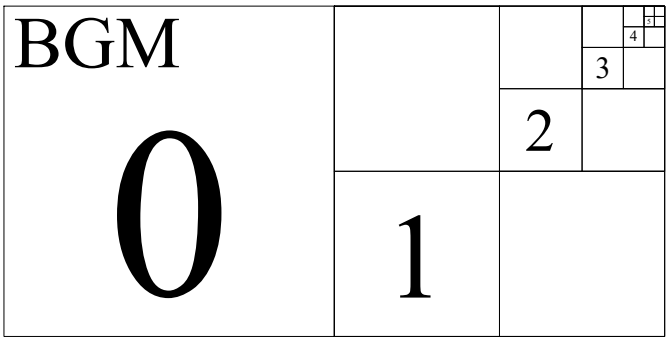
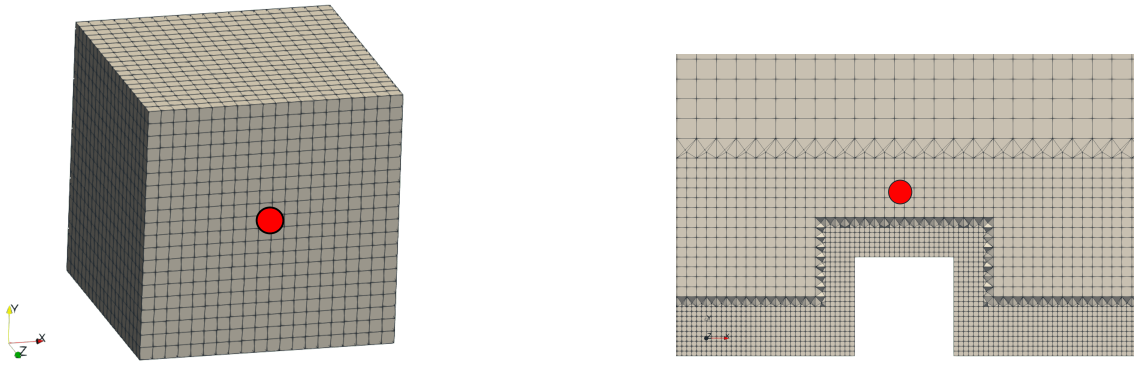


Figure 5.3: Refinement levels

- **locationInMesh definition.**

It is essential to be clear about the type of flow to be studied, whether **internal or external flow.** Based on this decision, a point must be positioned inside or outside the mesh as described in Fig.5.4:



(a) Internal Flow (b) External Flow

Figure 5.4: Meshes to internal and external flow

To generate an internal or external mesh, follow these instructions: Open the "snappyHexMeshDict" file, located in the sub-folder "system", scroll down to "locationInMesh" and modify this point coordinates. **The point defined in locationInMesh must not be on any surface of the geometry.**

The following code shows how to modify the castellatedMeshControls:

```
castellatedMeshControls /* Creates a structured mesh from a background mesh.
Determine how this initial mesh is refined and adjusted to the geometry. */
{

 /* The maximum number of cells allowed in each processor.
 If this number is exceeded, snappyHexMesh stops local refinement. */
 maxLocalCells 2000000;

 /* The total maximum number of cells allowed in the entire mesh. */
 maxGlobalCells 15000000;

 // Minimum number of cells to refine between refinement levels.
 minRefinementCells 10;

 // Number of cells between refinement levels.
```

```
16  nCellsBetweenLevels 2;
17
18
19  /* Explicit feature edge refinement:
20  Defines the feature files (usually .eMesh files) that contain
21  the edges of the geometry that need to be captured precisely. */
22  features
23  (
24
25  );
26
27  // Defines the geometry surfaces and the refinement levels applied to them.
28  refinementSurfaces
29  {
30   "(Barlovento|Sotavento|Cubierta|MIzquierdo|MDerecho)"
31   {
32    level (2 2); // Minimum and maximum refinement levels
33    patchInfo
34    {
35     type wall;
36    }
37   }
38  }
39
40  /* Feature angle used to detect sharp edges in the geometry.
41  If the angle between two adjacent cells is greater than this value,
42  it is considered a feature that needs to be refined. */
43  resolveFeatureAngle 30;
44
45  // Defines regions in space and the refinement levels applied
46  //to these regions.
47  refinementRegions
48  {
49   Vref1
50   {
51    mode inside;
52    levels ((1 1));
53   }
54   Vref2
55   {
56    mode inside;
57    levels ((2 2));
58   }
59   Vref4
60   {
61    mode inside;
62    levels ((2 2));
63   }
```

```
    Vreftop
    {
     mode inside;
     levels ((1 1));
    }
    searchableBox_1
    {
     mode inside;
     levels ((1 1));
    }
   }


   // Mesh selection.
   locationInMesh (2.5 1 1); //External
   // locationInMesh (0 0.01 0); //Internal

   // Allows or disallows the existence of free-standing faces in mesh zones.
   allowFreeStandingZoneFaces true;
  }

  snapControls /* Controls the second snapping phase.
  The mesh created during the castellation phase is adjusted to better align
  with the surface geometry. This involves moving mesh points to the nearest surface
  and refining to capture the geometry accurately. */
  {
   /* Number of iterations for internal smoothing to reduce non-orthogonality at
   the face of refinement (effectively making the faces non-planar). Due to complex
   models, the numerical mesh is rarely orthogonal. So the non-orthogonality correction
   must be made for stability and accuracy. */
   nSmoothPatch 3;

   /*  Relative distance for points to be attracted by surface feature point
   or edge. True distance is this factor times local maximum edge length. */
   tolerance 1.0;

   /* Number of mesh displacement relaxation iterations. Increasing the number of
   solve iterations can help in achieving a more accurate alignment with the surface
   but may increase computational cost.*/
   nSolveIter 300;

   /*  Number of relaxation iterations during the snapping. If the mesh does not
   conform the geometry and all the iterations are spend, user may try to increase
   the number of iterations. Relaxation iterations help in stabilizing the snapping
   process by preventing abrupt changes in the displacement field. */
   nRelaxIter 5;

   // Feature snapping
```

```
112
113  /* The number of feature snapping iterations.
114  This helps in aligning the mesh with sharp features in the geometry. */
115  nFeatureSnapIter 10;
116
117  /* Implicit feature snapping tries to detect and snap to sharp edges
118  automatically without requiring explicit feature files. */
119  implicitFeatureSnap true;
120
121  /* When enabled, the snapping process uses the feature edges
122  defined in the features section of castellatedMeshControls. */
123  explicitFeatureSnap false;
124
125  // Detect features between multiple surfaces.
126  multiRegionFeatureSnap true;
127 }
```

4. Modify the meshQualityControls, writeFlags and mergeTolerance:

```
1  meshQualityControls // Quality criteria for the generated mesh.
2  {
3   #include "meshQualityDict"
4
5   /* Set of mesh quality parameters that can be used in specific
6   meshing phases where relaxed rules are allowed. */
7   relaxed
8   {
9    /* Non-orthogonality is a measure of how much the cell faces deviate
10   from being perpendicular to each other. High non-orthogonality can
11   affect the accuracy and stability of numerical simulations. */
12   maxNonOrtho 75;
13  }
14 }
15
16 // Advanced
17 /* Control the writing of extra information to assist in
18 post-processing and mesh analysis. */
19 writeFlags
20 (
21 scalarLevels    // write volScalarField that contains the cell ref lvls.
22 layerSets       /* write cellSets, faceSets of faces in layer.
23 This is useful for verifying that the boundary layers have been added
24 correctly and cover the intended surfaces. */
25 layerFields     /* write volScalarField for layer coverage.
26 This is particularly useful for ensuring that the boundary layers are
27 correctly formed and meet the specified thickness and growth ratios. */
28 );
```

```
29
30 /* helps manage the quality of the mesh by merging points that are within
31 a specified small distance of each other.
32 Properly tuning this parameter can greatly enhance the robustness and
33 accuracy of the resulting mesh, leading to better simulation outcomes.
34 Note: the write tolerance needs to be higher than this. */
35 mergeTolerance 1E-6;
```

5. •In the working directory, execute the command "snappyHexMesh.exe" to runs the meshing process **or use "snappyHexMesh > snappyHexMesh1-2.txt 2>&1" to runs the meshing process and all the information that would normally be displayed in the terminal (both standard output and error messages) will be saved in a file called "snappyHexMesh1-2.txt", in the working directory.**

•New folders will appear in the working directory (1 and 2), due to the processes activated and configured in step 1 of this process. The meshing processes were shown in Figure 4.1.

6. Make sure the process has completed without errors, see Figure 5.5:

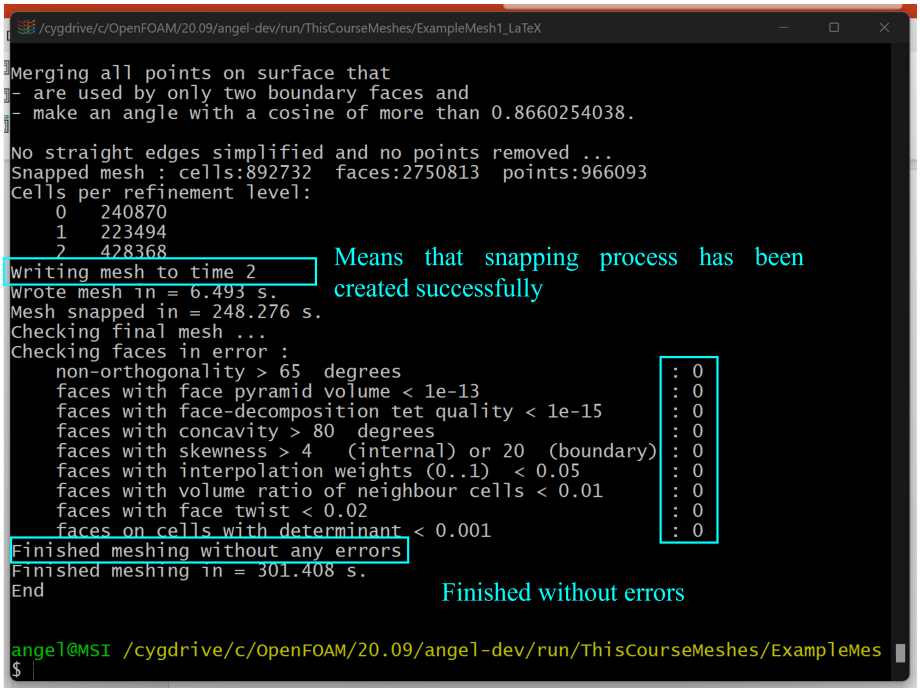
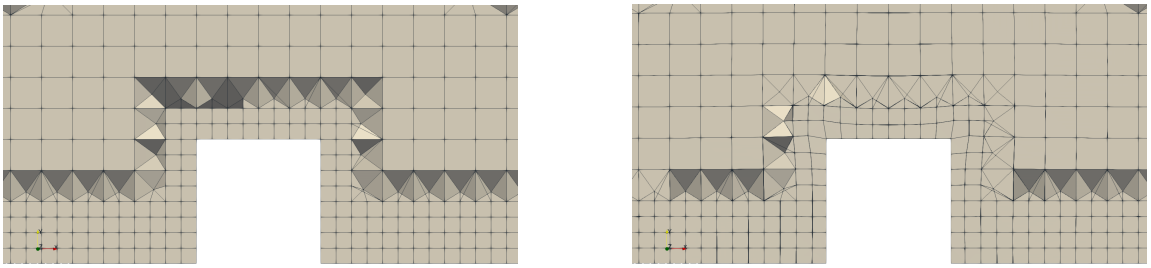


Figure 5.5: Example of successful completion of the meshing process, without adding surface layers.

Figure 5.6 shows the resulted mesh with the applied settings in the aforementioned steps.



(a) Castellated process (b) Snapping process

Figure 5.6: Castellated and Snapping processes in the mesh

7. Check the mesh quality using the "checkMesh.exe" command **or use "checkMesh > checkMesh.log" to write a log file with**

the checkMesh instruction in the working directory. See Chapter 6 for basic information about mesh quality.

Figure 5.7 shows the OpenFOAM console with the mesh stats, number of cells of each type and topology checks.

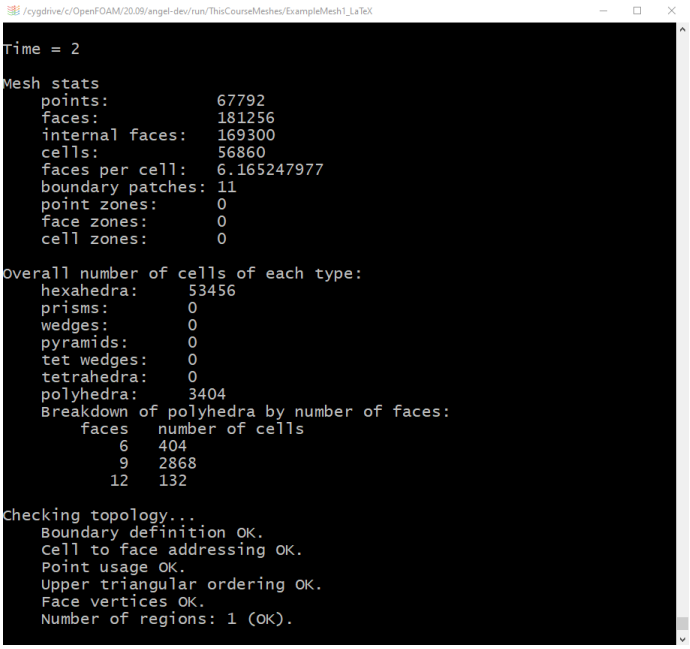


Figure 5.7: Example checkMesh 1/3.

Figure 5.8 shows the OpenFOAM console indicating the result of checking the mesh patch topology for multiply connected surfaces. The status "ok (non-closed singly connected)" means that the patch is topologically correct, well-defined, and properly connected. No issues were found with the topology of the surfaces checked, so the mesh should be ready for simulation.

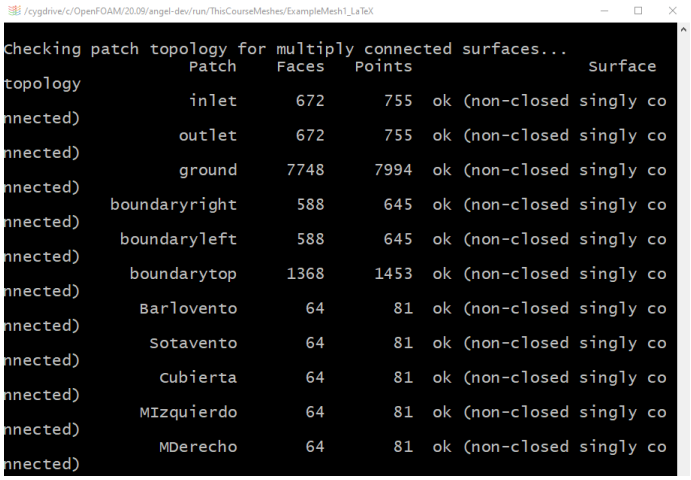


Figure 5.8: Example checkMesh 2/3.

Figure 5.9 shows the OpenFOAM console with the mesh quality. Special attention should be paid to "Max aspect ratio", "Mesh non-orthogonality" and "Max skewness" aspects. See section 6.1 for background information.

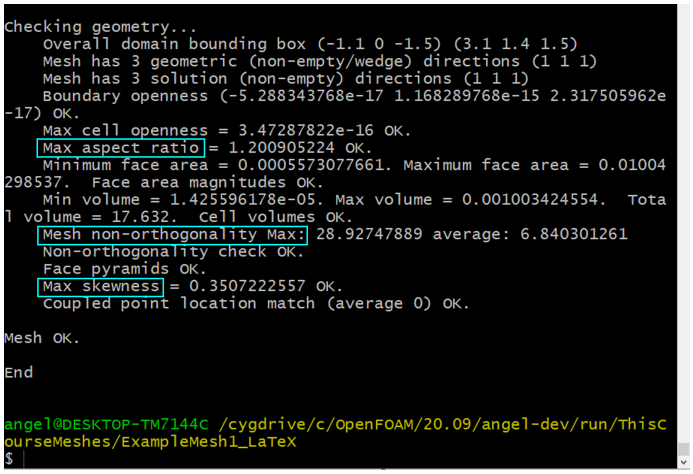


Figure 5.9: Example checkMesh 3/3.

If you receive any error messages, modify the necessary scripts and repeat step 5 as needed.

For each execution of the snappyHexMesh command, it is necessary to delete the generated temporary folders 1, 2 and 3 from the working directory. Keep folder 0.

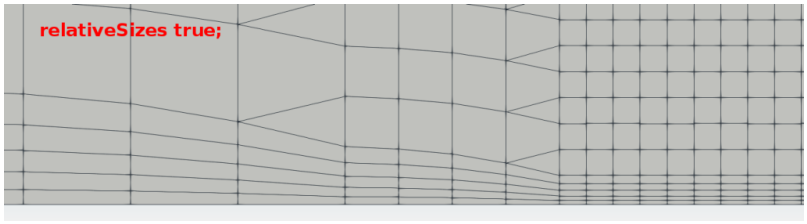
Play around with the following parameters:

- **refinementSurfaces (minimum and maximum)**
- **minRefinementCells**
- **nCellsBetweenLevels**
- **Create a copy of the main mesh folder, so as not to lose progress.**

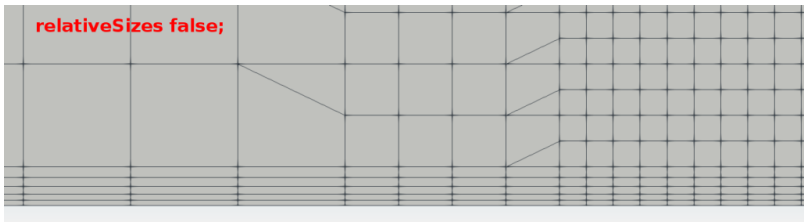
5.3.2 Steps to modify the snappyHexMeshDict file with surface layers, relativeSizes true or false

The layer addition process uses the settings in the "addLayersControls" sub-dictionary in snappyHexMeshDict, with the entries listed below. The user can choose from four different layer thickness parameters — expansionRatio, finalLayerThickness, firstLayerThickness, thickness — but must specify only 2 of them; specifying more than 2 results in an over-specified problem [4].

relativeSizes, true: When relativeSizes is enabled, this means that the sizes of the refined cells are defined relative to the not distorted base cell size in the refinement region (ratio).
relativeSizes, false: When relativeSizes is not enabled, the sizes of the refined cells are specified in absolute terms rather than relative to the base cell size (meters).



(a) Example of relativeSizes true;



(b) Example of relativeSizes false;

Figure 5.10: Example of relativeSizes: true and false [11]

Once a structured mesh with the desired refinement levels, as explained in the previous process (5.3.1), is obtained, the following steps should be undertaken:

1. Enable addLayers process in the snappyHexMeshDict file and disable castellatedMesh and snap processes, as follow:

```
castellatedMesh false;
snap            false;
addLayers       true;
```

Do not delete folders 1 and 2, created in the previous process.

2. Use the Y+ calculator to estimate the first cell height; see Chapter 8.

3. Once the height of the first cell has been estimated, use the Python function to calculate "finalLayerThickness" and "thickness" parameters, used to define the layers to add on the walls

and modify the "addLayersControl" subdictionary whether "relativeSizes" is true or false, see Chapter 9 to known how the function works. Figure 5.11 shows its usage.

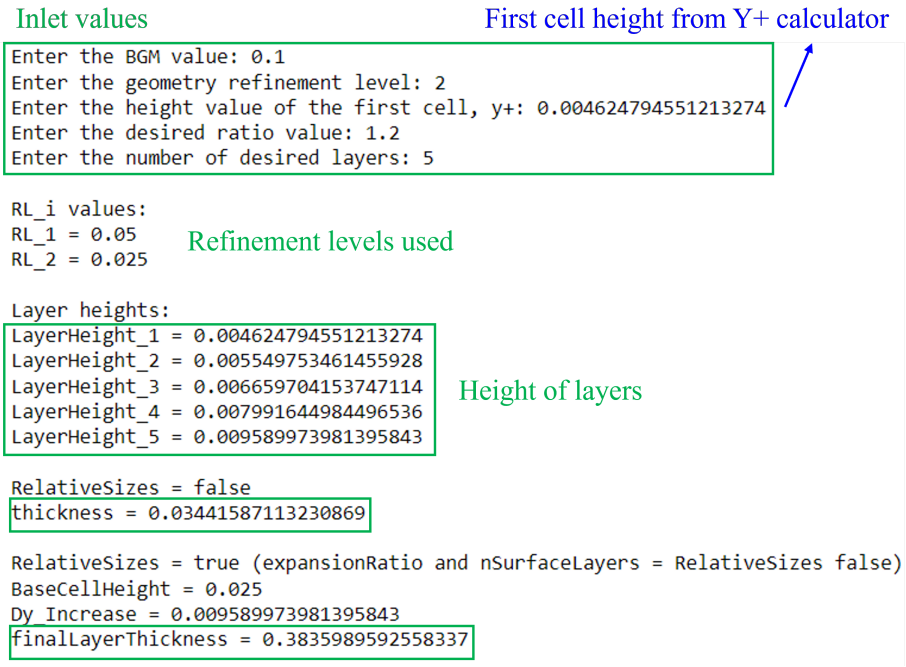


Figure 5.11: Example of usage of the Python function to calculate layer addition

4. Modify the "addLayersControls" sub-dictionary in the "snappyHexMeshDict" file.
   Modify the following script, use "true" or "false" in the "relativeSizes" line, comment or uncomment "finalLayerThickness" or "thickness" lines in the code to use relative or absolute values. Their values were calculated in the previous step 3.

   It is recommended to use both strategies: relativeSizes true and false. Sometimes one of these two strategies fits the geometry better.

```
addLayersControls
{
 /* Determines whether the layer thicknesses are specified
 relative to the surrounding cell sizes (true)
 or as absolute values (false). */
 relativeSizes false; // true or false

 layers // Specifies the surfaces and the number of layers to add.
 {
  "(Barlovento|Sotavento|Cubierta|MIzquierdo|MDerecho)"
  {
   nSurfaceLayers 5;
  }
 }

 expansionRatio 1.2; // Layer grows compared to the previous one.

 // RelativeSizes are true or false?

 /* if true: finalLayerThickness will be 0.38%, relative to
 the not distorted refinement level cell size in the wall */
 //finalLayerThickness 0.38;

 /* if false: specify the overall thickness of the cells in meters*/
 thickness 0.034;

 /* Minimum thickness of cell layer.
 If the layer cannot be above minThickness do not add layer. */
 minThickness 1e-05;

 /* The number of layers to grow when layer addition fails.
 This helps to fill gaps and ensure smooth layer transitions. */
 nGrow 0;

 // Advanced settings

 /* The angle between surface normals at which the layer addition switches
 from extrusion to a boundary layer type.
 This helps to handle sharp corners. */
 featureAngle 60;

 /* The number of relaxation iterations during layer addition.
 This helps to improve the quality of the layers. */
 nRelaxIter 5;

 /* Number of smoothing iterations of surface normals.
 Smoothing surface normals helps to avoid abrupt changes
 that can lead to poor quality layers. */
```

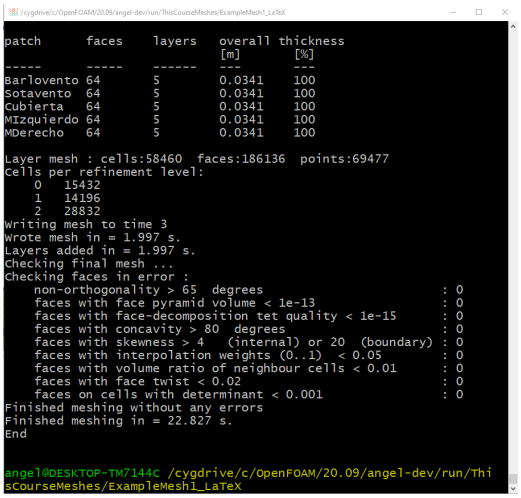
```
    nSmoothSurfaceNormals 1;

    // Number of smoothing iterations of interior mesh movement direction.
    nSmoothNormals 3;

    // Smooth layer thickness over surface patches
    nSmoothThickness 10;

    // The maximum allowable thickness ratio of a layer face to its neighboring face.
    maxFaceThicknessRatio 0.5;

    /* The maximum allowable ratio of layer thickness to the distance
    from the nearest surface. */
    maxThicknessToMedialRatio 0.3;

    // The minimum angle for the median axis to avoid collapse of layer cells.
    minMedianAxisAngle 90;

    // The number of buffer cells that should not extrude.
    nBufferCellsNoExtrude 0;

    /* The number of iterations for layer addition. More iterations
    can lead to better quality but higher computational cost. */
    nLayerIter 50;

    /* Increasing the number of relaxation iterations can improve
    mesh quality, particularly in complex geometries where
    layers might otherwise intersect or create poor-quality cells. */
    nRelaxedIter 20;
}
```

5. •In the working directory, execute the command "snappyHexMesh.exe" to runs the meshing process **or use "snappyHexMesh > snappyHexMesh3.txt 2>&1" to runs the meshing process and all the information that would normally be displayed in the terminal (both standard output and error messages) will be saved in a file called "snappyHexMesh3.txt", in the working directory.**

   •A new folders will appear in the working directory (3), with the last meshing process.

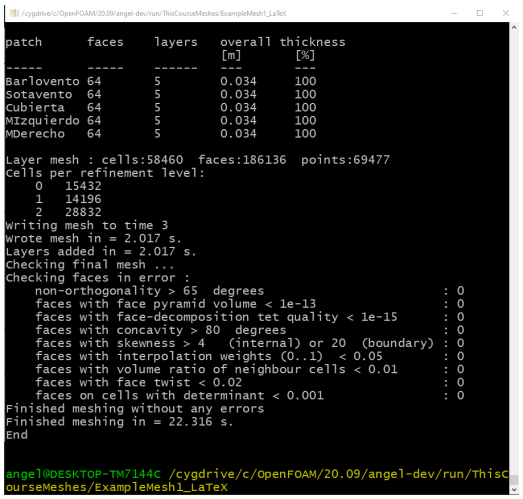
Figure 5.12 shows the results of using relativeSizes true and false. Both strategies

result in a similar mesh, due to the geometry shape. Slight differences can be seen in the "overall" console output. 0.0341 m vs 0.0340 m when relativeSizes are true and false, respectively.



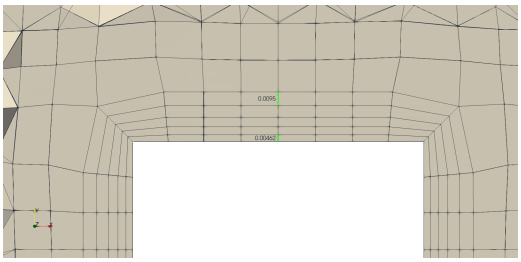
```
patch      faces    layers   overall thickness
                             [m]        [%]
-----      -----    ------   ---        ---
Barlovento 64       5        0.0341     100
Sotavento  64       5        0.0341     100
Cubierta   64       5        0.0341     100
MIzquierdo 64       5        0.0341     100
MDerecho   64       5        0.0341     100

Layer mesh : cells:58460  faces:186136  points:69477
Cells per refinement level:
    0   15432
    1   14196
    2   28832
Writing mesh to time 3
Wrote mesh in = 1.997 s.
Layers added in = 1.997 s.
Checking final mesh ...
Checking faces in error :
    non-orthogonality > 65  degrees                      : 0
    faces with face pyramid volume < 1e-13               : 0
    faces with face-decomposition tet quality < 1e-15    : 0
    faces with concavity > 80  degrees                   : 0
    faces with skewness > 4   (internal) or 20  (boundary) : 0
    faces with interpolation weights (0..1)  < 0.05      : 0
    faces with volume ratio of neighbour cells < 0.01    : 0
    faces with face twist < 0.02                         : 0
    faces on cells with determinant < 0.001              : 0
Finished meshing without any errors
Finished meshing in = 22.827 s.
End

angel@DESKTOP-TM7144C /cygdrive/c/OpenFOAM/20.09/angel-dev/run/Thi
sCourseMeshes/ExampleMesh1_LaTeX
```

(a) Console, using relativeSizes true;

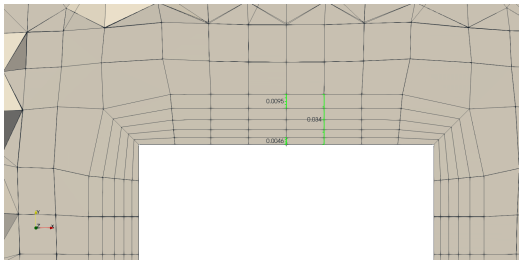


```
patch      faces    layers   overall thickness
                             [m]        [%]
-----      -----    ------   ---        ---
Barlovento 64       5        0.034      100
Sotavento  64       5        0.034      100
Cubierta   64       5        0.034      100
MIzquierdo 64       5        0.034      100
MDerecho   64       5        0.034      100

Layer mesh : cells:58460  faces:186136  points:69477
Cells per refinement level:
    0   15432
    1   14196
    2   28832
Writing mesh to time 3
Wrote mesh in = 2.017 s.
Layers added in = 2.017 s.
Checking final mesh ...
Checking faces in error :
    non-orthogonality > 65  degrees                      : 0
    faces with face pyramid volume < 1e-13               : 0
    faces with face-decomposition tet quality < 1e-15    : 0
    faces with concavity > 80  degrees                   : 0
    faces with skewness > 4   (internal) or 20  (boundary) : 0
    faces with interpolation weights (0..1)  < 0.05      : 0
    faces with volume ratio of neighbour cells < 0.01    : 0
    faces with face twist < 0.02                         : 0
    faces on cells with determinant < 0.001              : 0
Finished meshing without any errors
Finished meshing in = 22.316 s.
End

angel@DESKTOP-TM7144C /cygdrive/c/OpenFOAM/20.09/angel-dev/run/ThisC
ourseMeshes/ExampleMesh1_LaTeX
```

(b) Console, using relativeSizes false;



(c) Layers, using relativeSizes true;



(d) Layers, using relativeSizes false;

Figure 5.12: Using relativeSizes: true and false

Chapter 6

checkMesh utility

The checkMesh utility is used to verify the quality of the generated mesh for a CFD simulation. When you run it, the program analyzes the mesh and provides a detailed report that includes:

- **Mesh dimensions:** Shows the number of nodes, cells, faces, and points.
- **Mesh quality:** Checks various parameters such as cell aspect ratio, skewness, and non-orthogonality.
- **Mesh topology:** Verifies the connectivity between cells and faces, and whether there are any poorly defined cells (e.g., cells with negative volumes).
- **Errors or warnings:** Identifies any issues with the mesh that could affect the accuracy or stability of the simulation.

This command is crucial to ensure that the mesh is suitable before proceeding with a simulation, as a poor-quality mesh can lead to incorrect results or numerical instability during the simulation.

6.1 Mesh quality

1. **Cell aspect ratio:**

- **Definition:** The aspect ratio of cells is the ratio of the cell's length in its longest direction to the length in its shortest direction.

$$HexahedralAspectRatio = \frac{max(x_1, x_2, ..., x_{12})}{min(x_1, x_2, ..., x_{12})} \tag{6.1}$$

$$TetrahedralAspectRatio = \frac{max(x_1, x_2, ..., x_6)}{2 \cdot \sqrt{6} \cdot r} \tag{6.2}$$

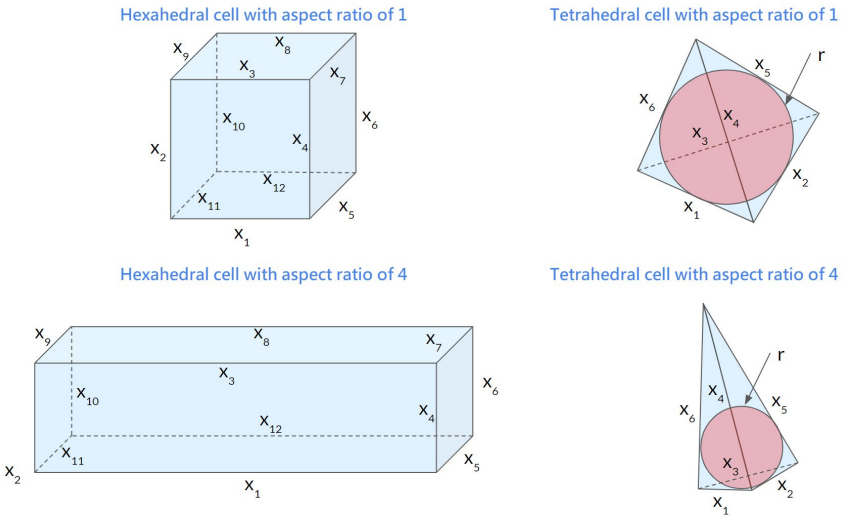


Figure 6.1: Cell aspect ratio sketch [10].

- **Importance:** An ideal cell aspect ratio is close to 1 (cubic cells). A high aspect ratio (long and thin cells) can cause interpolation errors and negatively impact the accuracy of the results.

2. **Skewness:**

- **Definition:** Skewness refers to the measure of how distorted a cell is compared to its ideal shape. Is the normalized distance between a line that connects two adjacent cell centroids and the distance from that line to the shared face's center.

$$Skewness = \frac{|\vec{s}|}{|\vec{d}|} \tag{6.3}$$

Where $|\vec{s}|$ is the norm of the skewness vector and $|\vec{d}|$ is the norm of the vector that connects the adjacent cell centroids, as shown in the Figure 6.2:

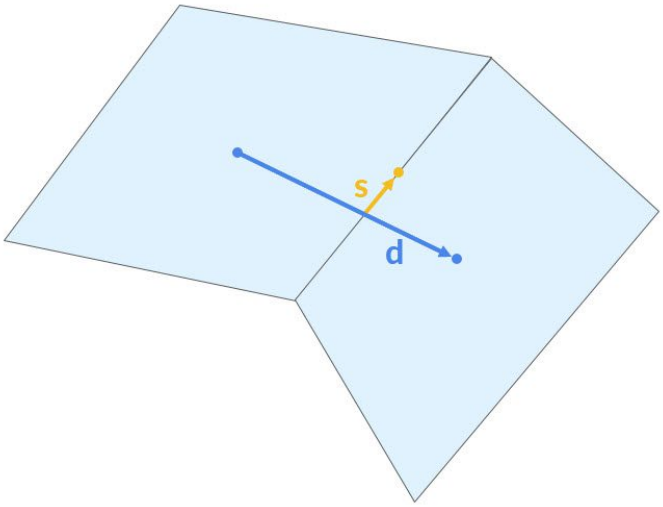


Figure 6.2: Skewness sketch [10].

- **Importance:** Low skewness is desirable because it ensures that the mesh cells are symmetric and well-aligned. High skewness can lead to interpolation errors and inaccuracies in gradient calculations.

3. **Non-orthogonality:**

- **Definition:** Non-orthogonality measures the angle between the normal vector of a cell face and the vector connecting the face center to the cell center. The range of non-orthogonality is between 0 (ideal) and 90 (worst). 0 indicates the mesh being orthogonal. Two perfect hexes aligned with each other have non-orthogonality equal to 0, see Figure 6.3.

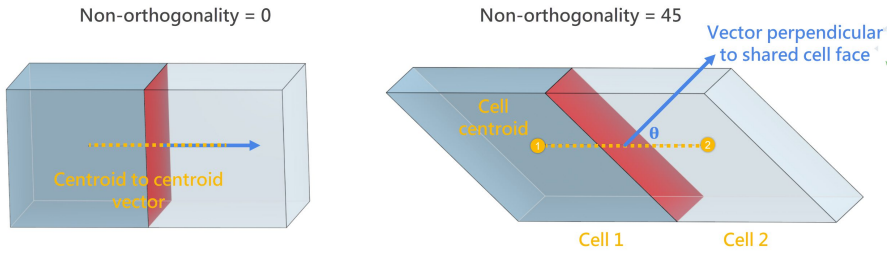


Figure 6.3: Non-orthogonality sketch [10].

- **Importance:** High non-orthogonality indicates misalignment between these vectors, which can cause instability and numerical errors during the simulation. It is critical to keep this parameter under control to ensure simulation quality.

These parameters are fundamental in assessing the quality of the mesh, as they directly influence the accuracy, stability, and efficiency of CFD simulations. A good mesh should have optimal values in these parameters to ensure reliable results.

RECOMMENDATION

- Max aspect ratio: $1 < x > 200$
- Max non-orthogonality: < 70
- Max skewness: < 4

Chapter 7

First cell height estimation, background

In computational fluid dynamics (CFD) simulations, the height of the first cell in the mesh is crucial for accurately capturing the physical phenomena near the walls. Here are some key reasons why it is important:

1. **Boundary layer capture:** The boundary layer is the region of the flow near a surface where viscosity effects are significant. The height of the first cell must be small enough to resolve the velocity and temperature gradients within the boundary layer accurately.

2. **Wall conditions:** To correctly apply wall boundary conditions (such as the wall function or turbulence models), the first cell needs to be within the appropriate range of the y^+ value (a dimensionless parameter representing the distance from the wall in terms of viscous length units).

3. **Turbulence models:** Many turbulence models, such as the k-ϵ or k-ω models, have specific requirements for the first cell location. If the first cell is not at the correct position (usually at a specific y^+), turbulence models may not work properly and could produce inaccurate results.

4. **Resolution of pressure and temperature gradients:** A refined mesh near

the walls ensures that pressure and temperature gradients are captured accurately, which is essential for predicting phenomena such as flow separation, heat transfer, and wall friction.

5. **Numerical stability:** A well-designed mesh with an appropriate first cell height contributes to the stability and convergence of the numerical solver, reducing the risk of errors and increasing the accuracy of the simulation.

In summary, the height of the first cell in the mesh is a determining factor in the quality and accuracy of a CFD simulation, especially in regions near solid surfaces where viscous effects are significant.

A fisherman uses a fishing net, which is a type of mesh structure. If he is trying to catch small fish, then the mesh size needs to be small enough to capture them. In this case, large fish will also be caught. Similarly, if we intend to resolve the effects near the wall, that is, in the viscous sublayer, then the mesh size must be small and dense enough near the wall to capture almost all the effects.

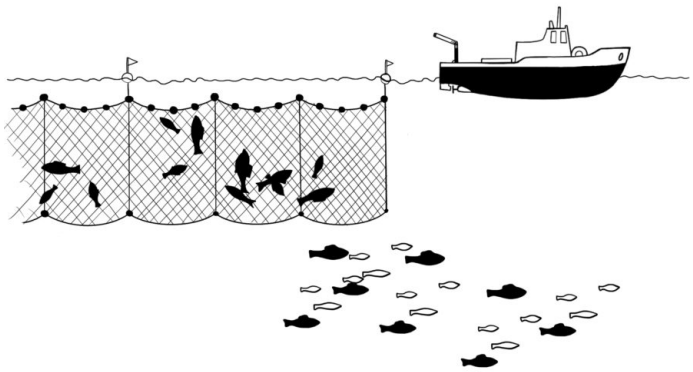


Figure 7.1: Mesh size analogy

Estimation of first cell height is a first guess.

In this chapter, we will provide a concise overview of fundamental concepts essential for understanding the significance of the first cell height in computational fluid dynamics (CFD) simulations.

7.1 Turbulent boundary layer

The turbulent boundary layer is a region of fluid near a solid surface where the flow is dominated by turbulence, meaning chaotic and disordered movements of fluid particles. This layer develops when the flow transitions from being laminar (smooth and orderly) to turbulent due to the instability of the flow at high velocities or rough surfaces, as shown in Figure 7.2 [7].

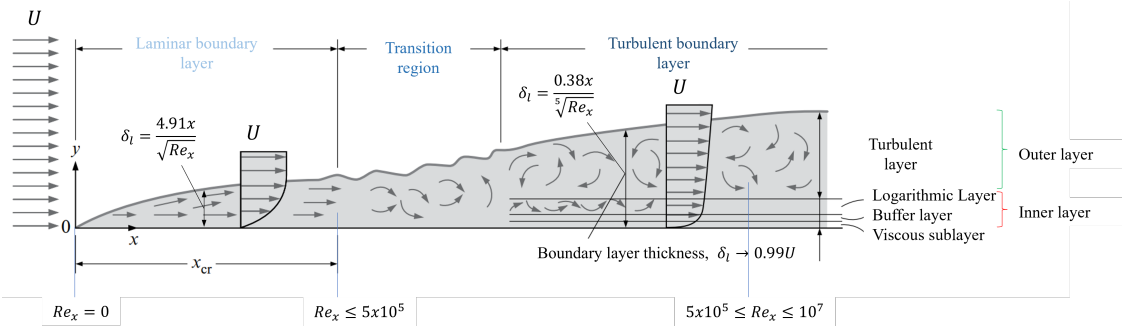


Figure 7.2: Development and structure of a turbulent boundary layer

7.1.1 Development of the Turbulent Boundary Layer

1. **Laminar boundary layer:** Near the leading edge of a surface (e.g., an airplane wing), the flow is initially laminar.

2. **Transition to turbulence:** As the flow moves further along, small disturbances can grow and lead to a transition to turbulence.

3. **Turbulent boundary layer:** Eventually, the flow becomes fully turbulent, characterized by vortices and fluctuations in velocity and pressure.

The thickness of the boundary layer varies as a function of the Reynolds number:

$$\frac{\overline{\delta_l}}{L_c} \sim \frac{1}{\sqrt{Re_x}} \tag{7.1}$$

where Re_x is the Reynolds number based on the distance from the leading edge of the plate (x) [5]:

$$Re_x = \frac{\rho U L_c}{\mu} = \frac{U L_c}{\nu} \tag{7.2}$$

where:

- μ is the dynamic viscosity of the fluid.
- L_c is the distance from the leading edge of the plate.
- ν is the kinematic viscosity of the fluid.

7.1.2 Basic Structure of the Turbulent Boundary Layer

1. **Viscous sub-layer (or laminar sub-layer):** Very close to the wall, the flow remains mainly laminar and viscosity dominates.
2. **Buffer layer:** Further from the wall, where vortices and turbulence begin to form.
3. **Logarithmic layer:** Where the average flow velocity can be described by a logarithmic law.
4. **Outer layer:** The farthest part from the wall, where larger turbulent structures dominate.

Viscous sub-layer, Buffer layer and Logarithmic layer constitute what is known as the "Inner layer".

7.2 Universal law of the wall

Before World War II, Johann Nikuradse took velocity measurements in smooth pipes at different distances from the inner surface and published his results in 1933. Previously, in 1775, Chézy had proposed the first empirical friction law when determining the cross-section of channels to supply water to Paris. In the 1860s, Bazin also took velocity measurements in channels but couldn't find an exact equation. **It was Nikuradse, along with Prandtl, who normalized the axes in terms of y^+ and u^+, making all the data sets match and writing an equation to describe the velocity near the wall** [9].

The "universal law of the wall" in (CFD) is an empirical relationship that describes how fluid flow behaves in the immediate vicinity of the wall. It is called "universal" in the sense that its form remains similar in different turbulent flow situations near solid surfaces, although the exact values of the constants may vary slightly depending on the type of flow and specific conditions.

The law of the wall divides the region near the wall into three sub-regions, known as the "Inner layer" of the turbulent boundary layer.

The typical formulation of the **law of the wall in the logarithmic layer** is:

$$u^+ = \frac{1}{\kappa} \ln(y^+) + B \tag{7.3}$$

where:

- u^+ is the dimensionless velocity.
- y^+ is the dimensionless distance from the wall.
- κ (kappa) is the von Kármán constant, approximately equal to 0.41.
- B is an empirical dimensionless constant, approximately equal to 5.2.

The relationship between the dimensionless velocity u^+ and the dimensionless distance from the wall y^+ is often visualized in a graph, see Figure 7.3; is of great help to understanding the transition between different flow regions near the wall:

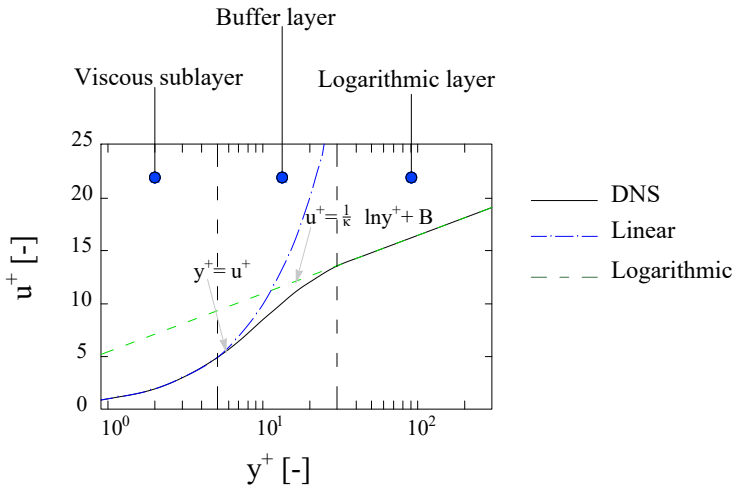


Figure 7.3: Universal law of the wall

Table 7.1: y^+ values, according to universal law of the wall

Layer	y^+ **value**
Viscous sublayer	$1 < y^+ < 5$
Buffer layer	$5 < y^+ < 30$
Logarithmic layer	$30 < y^+ < 300$

7.2.1 u^+ parameter

The parameter u^+ is the dimensionless velocity in the turbulent boundary layer and is used to describe the behavior of fluid flow in the vicinity of a solid surface, such as a wall, and is defined as:

$$u^+ = \frac{u}{u_*} \tag{7.4}$$

where:

- u is the local velocity of the flow parallel to the wall.
- u_* is the friction velocity, which is a measure of the tangential velocity of the flow at the wall and is defined as:

The friction velocity u_* can be calculated using the wall shear stress τ_w [5]:

$$u_* = \sqrt{\frac{\tau_w}{\rho}} \tag{7.5}$$

where:

- τ_w is the wall shear stress.
- ρ is the fluid density.

The wall shear stress τ_w can be estimated for a turbulent boundary layer over a flat plate using empirical correlations, such as the one from the law of the wall:

$$\tau_w = \rho {u_*}^2 \tag{7.6}$$

7.2.2 y^+ parameter

A crucial aspect of modeling the boundary layer is calculating the height of the first cell from the wall. This value is directly related to the dimensionless parameter y^+, which is a measure of the distance from the wall in terms of viscous length units. Choosing the appropriate first cell height ensures that the no-slip wall conditions and velocity gradients are accurately represented, which is essential for turbulence models.

Some mature turbulence models, such as $k - \epsilon$, are only valid in fully developed turbulence regions and do not perform well near the wall. To address the near-wall region, two methods are typically proposed [9]:

1. **Integrating the turbulence to the wall:** Turbulence models are modified to resolve the viscosity-affected region with the mesh extending all the way to the wall, including the viscous sublayer. **When using a low Reynolds number** turbulence model to solve the near-wall region, **the first cell center must be placed in the viscous sublayer** (preferably $y^+ = 1$), which requires a large number of mesh cells and substantial computational resources.

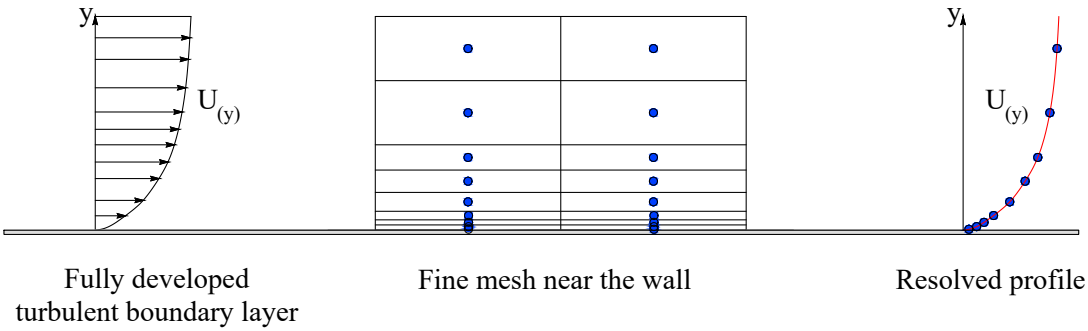


Figure 7.4: Sketch of boundary layer resolution

- Use low-Re model like $k - \omega$.
- When you are interested in the forces on the wall (aerodynamics analyses for F1 cars, blade optimization in turbo-machinery, etc.).

2. **Use the so-called wall functions, which can model the near-wall region:** Wall functions are empirically derived equations used to satisfy the physics in the near-wall region. **The first cell center needs to be placed in the log-law region to ensure accuracy. Wall functions bridge the inner region between the wall and the fully developed turbulence region, to reproduce what occurs between the wall and the wall-adjacent cell center.** Using the wall functions approach eliminates the need

to resolve the boundary layer, significantly reducing the mesh size and computational domain.

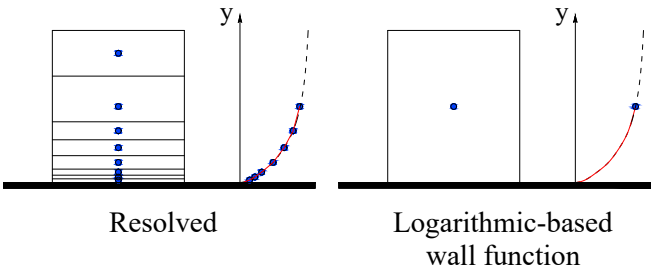


Figure 7.5: Sketch of resolved boundary layer and using a wall function

- First grid cell needs to be $30 < y^+ < 300$ (If this is too low, the model is invalid. If this is too high, the wall is not properly resolved.
- Use high-Re model (Standard $k-\epsilon$, RNG $k-\epsilon$).
- Use when you are more interested in the mixing rather than the forces on the wall.

The dimensionless wall distance y^+ is a **crucial parameter in the correct resolution of the boundary layer in CFD simulations.** It is defined as [5]:

$$y^+ = \frac{y_P u_* \rho}{\mu} = \frac{y_P u_*}{\nu} \tag{7.7}$$

where:

- y_P is the distance from the wall to the center of the first cell.
- u_* is the friction velocity.
- ρ is the density.
- μ is the dynamic viscosity of the fluid.

- ν is the kinematic viscosity of the fluid ($\nu = \mu/\rho$).

Since y_P is the distance from the wall to the center of the first cell. The first cell height can be estimated as follow:

$$y_H = 2y_P \tag{7.8}$$

Given the complexity of obtaining exact values for u_* and τ_w, practical approaches often involve using known or estimated y^+ values to back-calculate the first cell height y_H.

7.2.2.1 y^+ Example Calculation

Let's assume the following parameters for our example:

- Desired y^+ value: 30 (a typical target for wall-resolved LES or RANS simulations).
- Kinematic viscosity ν: $1.5 \times 10^{-5}\,\mathrm{m}^2/\mathrm{s}$ (typical for air at room temperature).
- Fluid density ρ: $1.225\,\mathrm{kg}/\mathrm{m}^3$.
- Estimated wall shear stress τ_w: $0.1\,\mathrm{N}/\mathrm{m}^2$.

First, calculate the friction velocity u_τ:

$$u_* = \sqrt{\frac{\tau_w}{\rho}} = \sqrt{\frac{0.1}{1.225}} \approx 0.285\,\mathrm{m/s}$$

Next, rearrange the y^+ formula to solve for the distance to the center of the first cell y_P:

$$y_P = \frac{y^+ \nu}{u_*}$$

Substitute the known values:

$$y_P = \frac{30 \times 1.5 \times 10^{-5}}{0.285} \approx 1.58 \times 10^{-3}\,\mathrm{m} = 1.58\,\mathrm{mm}$$

Since y_H is the height of the first cell, and y_P is the distance to the center of the first cell, we have:

$$y_H = 2y_P = 2 \times 1.58 \times 10^{-3}\,\text{m} \approx 3.16\,\text{mm}$$

So, for a target y^+ of 30, the height of the first cell y_H should be approximately 3.16 mm.

This example illustrates the process of calculating the first cell height based on a desired y^+ value, ensuring that the boundary layer is adequately resolved in the simulation.

7.2.3 Wall Shear Stress on a Flat Plate

The wall shear stress (τ_w) on a flat plate in a turbulent boundary layer can be estimated using the following equation [5]:

$$\tau_w = \left(\frac{1}{2}\rho U^2\right) C_f \tag{7.9}$$

where:

- τ_w is the wall shear stress.
- ρ is the fluid density.
- U is the free-stream velocity.
- C_f is the skin friction coefficient.

7.2.3.1 Skin Friction Coefficient

The skin friction coefficient (C_f) can be determined using an empirical correlation for fully developed turbulent flow over a flat plate:

$$C_f = (2log_{10}(Re_x) - 0.65)^{-2.3} \tag{7.10}$$

where Re_x is the Reynolds number based on the distance from the leading edge of the plate (x), see Equation 7.2

Some other empirical C_f are provided, depending on the flow type [6].

Internal flow:

$$C_f = 0.079 \times Re^{-0.25} \tag{7.11}$$

External flow:

$$C_f = 0.058 \times Re^{-0.20} \tag{7.12}$$

7.2.3.2 Wall Shear Stress on a Flat Plate Example Calculation

Let's assume the following parameters:

- $U = 10\,\mathrm{m/s}$.
- $\rho = 1.225\,\mathrm{kg/m^3}$.
- $\nu = 1.5 \times 10^{-5}\,\mathrm{m^2/s}$.
- $L = 1\,\mathrm{m}$.

Calculate Re_x:

$$Re_x = \frac{UL}{\nu} = \frac{10 \times 1}{1.5 \times 10^{-5}} = 6.67 \times 10^5$$

Determine C_f:

$$C_f = (2\log_{10}(Re_x) - 0.65)^{-2.3} = \left(2\log_{10}(6.67 \times 10^5) - 0.65\right)^{-2.3}$$

Calculating the logarithm and the skin friction coefficient:

$$C_f = \left(2\log_{10}(6.67 \times 10^5) - 0.65\right)^{-2.3} \approx (2 \times 5.824 - 0.65)^{-2.3} \approx (11.648{-}0.65)^{-2.3} \approx 0.003$$

Calculate τ_w:

$$\tau_w = \frac{1}{2} C_f \rho U^2 = \frac{1}{2} \times 0.003 \times 1.225 \times 10^2 \approx 0.184\,\mathrm{N/m^2}$$

So, the wall shear stress τ_w is approximately $0.184\,\mathrm{N/m^2}$.

This example illustrates the process of calculating the wall shear stress on a flat plate using the turbulent boundary layer equation.

7.3 Reasons why u^+ and y^+ are Wall Functions

7.3.1 Reasons for u^+ being a Wall Function

1. **Normalization of Velocity:**
   - u^+ is defined as the local flow velocity normalized by the friction velocity u_*. This normalization makes u^+ a dimensionless parameter, allowing for comparison across different flow conditions and systems.
2. **Relation to Wall Shear Stress:**
   - The friction velocity u_* is directly related to the wall shear stress τ_w, which is a crucial factor in determining the flow behavior near the wall. This relationship ensures that u^+ accurately represents the effects of the wall on the flow.
3. **Universal Law of the Wall:**
   - u^+ is a key component of the universal law of the wall, which describes how the velocity profile near the wall varies with distance. This law is essential for understanding and predicting the transition between different flow regions close to the wall.
4. **Turbulence Model Integration:**
   - Turbulence models use u^+ to validate and adjust their predictions in the near-wall region, ensuring that velocity gradients and no-slip conditions are accurately captured.

7.3.2 Reasons for y^+ being a Wall Function

1. **Normalization of Distance:**

   - y^+ is defined as the distance from the wall normalized by the viscous length scale ν/u_*, where ν is the kinematic viscosity. This normalization makes y^+ a dimensionless parameter, enabling consistent analysis of flow behavior near the wall.

2. **Dimensionless Representation:**

   - By representing the distance from the wall in dimensionless terms, y^+ facilitates the comparison of flow characteristics across different systems and scales, aiding in the development and validation of CFD models.

3. **Characterization of Flow Regions:**

   - y^+ is used to distinguish between different regions within the turbulent boundary layer, such as the viscous sublayer ($y^+ < 5$), buffer layer ($5 < y^+ < 30$), and logarithmic layer ($y^+ > 30$). Understanding these regions is critical for accurate modeling and simulation of near-wall turbulence.

4. **Boundary Layer Resolution:**

   - In CFD simulations, the height of the first cell from the wall is often chosen based on a target y^+ value to ensure that the boundary layer is adequately resolved. This approach helps maintain accurate representation of wall effects and velocity gradients in the simulation.

Chapter 8

First cell height estimation, using a web calculator

This y^+ calculator estimates the necessary height of the first mesh cell from the wall to achieve a target y^+, based on flat-plate boundary layer theory. The calculations are derived from the 5th edition of Frank M. White's "Fluid Mechanics".

Type the following link into your internet browser to access to y^+ calculator: https://volupe.com/support/all-y-calculator-when-meshing-a-geometry-for-cfd-analysis/.

Type the following link into your internet browser to access to the instructions and tips of y^+ calculator: https://volupe.com/simcenter-star-ccm/y-calculator-launched-at-volupes-webpage/.

Last access date: 07/08/2024.

First cell height in prism layer

Define your parameters to obtain the value for the height of the first cell in the prism layer.

Free stream velocity U [m/s]

20

Density ρ [kg/m^3]

1.204

Dynamic viscosity μ [kg*m/s = N*s/m^2 = Pa*s]

1.813E-5

Characteristic length L [m]

0.2

Desired y+ [dimensionless]

150

Select if your simulation has internal or external flow

- [ ] Internal
- [x] External

Output from calculator

Kinematic viscosity ν [m^4/s]

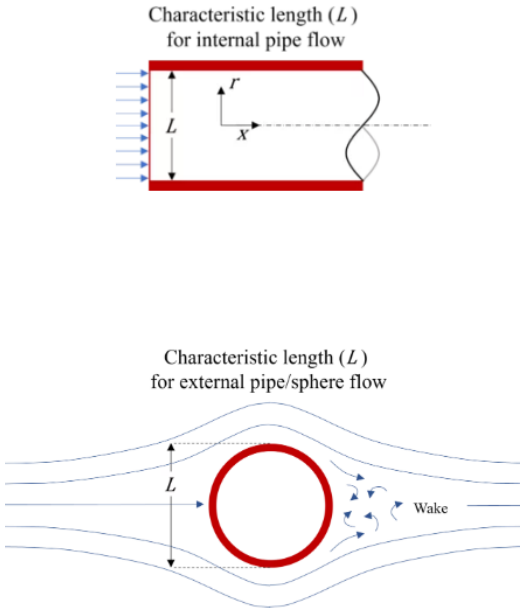
0.000015058139534883723

Reynolds number Re_x [dimensionless]

265637.0656370656

First cell height [m]

0.004624794551213274



(a) Inlet and output parameters (b) Characteristic lengths schemes

Figure 8.1: First cell height estimation, using a web tool

Chapter 9

Python function to calculate layer addition

The code provides a way to calculate the geometric characteristics of the prismatic layers in a mesh generated by snappyHexMesh, which is crucial for accurate boundary layer simulation in Computational Fluid Dynamics (CFD).

```
# Function to calculate the addition of prisms layers in
#snappyHexMesh utility in OpenFOAM (Version1). Sanchez-Cruz Angel.

def calculate_values(bgm, refinements, y_cell, ratio, num_cells):

# Calculate the RL_i list
rl_values = []
rl = bgm / 2.0
rl_values.append(rl)

for i in range(1, refinements):
rl = rl / 2.0
# Each refinement divides the previous value by 2
rl_values.append(rl)

# Get the CoreCell value
core_cell = rl_values[-1]  # Last value of the RL_i list

# Calculate cell heights
cell_heights = []
height = y_cell  # Initial height without rounding
cell_heights.append(height)
```

```
23
24  for _ in range(1, num_cells):
25  height = height * ratio  # Each layer multiplies the
26  #previous height by the ratio
27  cell_heights.append(height)
28
29  # Get the Dy_Increase value
30  dy_increase = cell_heights[-1]
31  # Last value of the cell heights list
32
33  # Calculate thickness
34  thickness = sum(cell_heights)
35  # Sum of the heights without rounding
36
37  # Calculate finalLayerThickness
38  final_layer_thickness = dy_increase / core_cell if core_cell !
39  = 0 else None
40
41  return rl_values, cell_heights, thickness, core_cell, dy_increase,
42  final_layer_thickness
43
44  # Example usage:
45  bgm = float(input("Enter the BGM value: "))
46  refinements = int(input("Enter the geometry refinement level: "))
47  y_cell = float(input("Enter the height value of the first cell, y+: "))
48  ratio = float(input("Enter the desired ratio value: "))
49  num_cells = int(input("Enter the number of desired layers: "))
50
51  rl_values, cell_heights, thickness, core_cell, dy_increase,
52  final_layer_thickness = calculate_values(bgm, refinements,
53  y_cell, ratio, num_cells)
54
55  # Display results
56  print("\nRL_i values:")
57  for i, value in enumerate(rl_values, start=1):
58  print(f"RL_{i} = {value}")
59
60  print("\nLayer heights:")
61  for i, height in enumerate(cell_heights, start=1):
62  print(f"LayerHeight_{i} = {height}")
63
64  print(f"\nRelativeSizes = false")
65  print(f"thickness = {thickness}\n")
66  print(f"RelativeSizes = true (expansionRatio and nSurfaceLayers =
67  RelativeSizes false)")
68  print(f"BaseCellHeight = {core_cell}")
69  print(f"Dy_Increase = {dy_increase}")
70  print(f"finalLayerThickness = {final_layer_thickness}" if
```

```
71  final_layer_thickness is not None else "finalLayerThickness =
72  Undefined (CoreCell is 0)")
```

9.1 Function description:

Here is a breakdown of what each part of the function does:

- **Calculation of the `RL_i` list:**
  - `rl_values = []`: Initializes an empty list to store the `RL_i` values.
  - `rl = bgm / 2.0`: Calculates the first `RL_i` value by dividing the `bgm` parameter (likely the background mesh size) by 2.
  - In a loop, the previous `rl` value is divided by 2 and added to the `rl_values` list until the number of refinements specified by `refinements` is completed.
  - The last value of `RL_i` is stored in `core_cell`, which represents the smallest cell size after all refinements.
- **Calculation of cell heights:**
  - `cell_heights = []`: Initializes an empty list to store the heights of the cell layers.
  - `height = y_cell`: Uses `y_cell` as the height of the first cell (without rounding).
  - In a loop, each subsequent layer height is calculated by multiplying the previous height by the `ratio`. This `ratio` is the expansion factor between layers. The values are stored in `cell_heights`.
- **Calculation of `Dy_Increase`:**
  - `dy_increase = cell_heights[-1]`: Takes the last cell height from the `cell_heights` list as the `Dy_Increase` value.

- **Calculation of the total layer thickness (`thickness`):**
  - `thickness = sum(cell_heights)`: Sums all the layer heights to obtain the total thickness of the prismatic layer.
- **Calculation of `finalLayerThickness`:**
  - `final_layer_thickness = dy_increase / core_cell if core_cell != 0 else None`: Calculates the relative thickness of the last layer based on the smallest cell (`core_cell`).
    If `core_cell` is 0, `final_layer_thickness` is set to `None`.
- **Return of results:**
  - The function returns a tuple with the calculated values: `rl_values`, `cell_heights`, `thickness`, `core_cell`, `dy_increase`, and `final_layer_thickness`.

Example usage: The user is prompted to input five values: BGM value, geometry refinement level, first cell height, desired ratio value, and number of desired layers. The `calculate_values` function is called with these inputs, and the results are printed in an organized manner.

Chapter 10

Parallelization of Mesh Creation in OpenFOAM

The parallelization of mesh creation in OpenFOAM refers to the process of dividing the mesh generation task into multiple subprocesses that can be executed simultaneously on different processor cores or across different machines in a cluster. This is particularly useful for speeding up the mesh creation time for large or complex meshes.

In OpenFOAM, parallelization can be achieved using tools such as "decomposePar", which splits the computational domain into several parts, each assigned to a different processor. After the mesh generation, the tool "reconstructPar" can be used to reconstruct the parallelized mesh into a single complete mesh if needed.

This approach allows the work to be performed more quickly by leveraging the available hardware resources, which is crucial when working with CFD simulations that require a detailed mesh and, therefore, high computational power.

10.1 How to Determine the Number of Processors to Use

The choice of how many processors to use when parallelizing a task in OpenFOAM, such as mesh creation, depends on several factors:

1. **Number of Available Cores**: The maximum number of processors you can use depends on the number of cores available on your machine or in the cluster you are using.

2. **Mesh Size and Complexity**: For a larger or more complex mesh, more computational power is generally required, justifying the use of more cores. However, if the mesh is small, using too many processors might be inefficient due to the communication overhead between processors.

3. **Available Memory**: Each parallel process requires a certain amount of memory. Make sure your system has enough RAM to handle the number of processors you plan to use.

4. **Scaling Efficiency**: There is a point of diminishing returns where adding more processors does not necessarily speed up the process due to the parallelization overhead (communication and synchronization between cores). This is known as inefficient scaling.

5. **Practical Testing**: It is often useful to perform tests with different numbers of processors to find the optimal balance between runtime and resource usage. For example, you might start by testing with 2, 4, 8, 16 processors, and so on, and then measure the runtime for each case.

In summary, you should balance the number of processors with resource availability, task complexity, and scaling efficiency to determine the optimal number of processors to use.

10.2 Steps to parallelize the meshing process

1. Check the number of processors on your computer. For windows system, use the following key combination: CTRL + SHIFT + ESC. And go to the "Performance" tab. The number of cores available will be the number of divisions that can be done to parallelize the process.

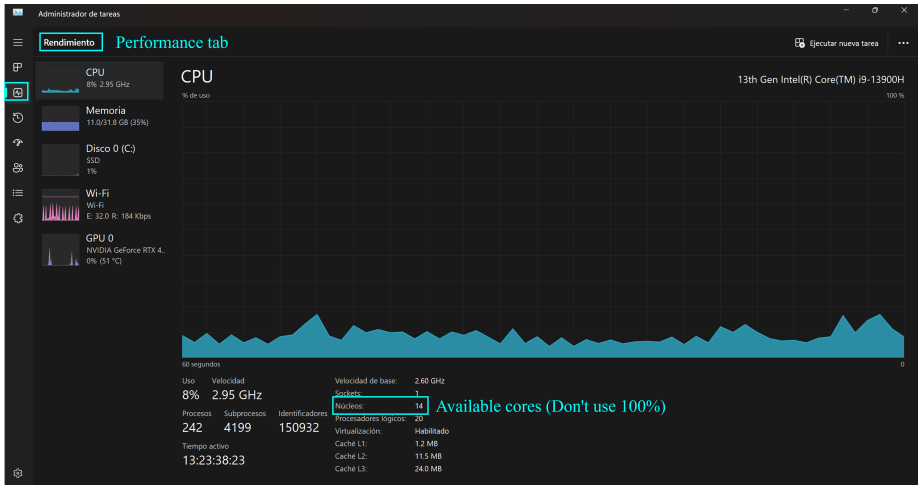


Figure 10.1: PC cores visualization in Windows

- **Not to use 100% of the cores.**
- **Use even numbers of cores.**
- **Apply practical testing, point 5 of section 10.1.**

2. Configure the "decomposeParDict" file , located in the "system" folder: This computer / C: / OpenFOAM / 20.09 / User-dev / run / ThisMeshingCourse / ExampleMeshX / system / decomposeParDict:

```
1  /*---------------------------*- C++ -*-----------------------------*\
2  =========                 |
3  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4  \\    /   O peration     | Website:  https://openfoam.org
```

```
\\  /    A nd           | Version:  dev
\\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
 version     2.0;
 format      ascii;
 class       dictionary;
 location    "system";
 object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

/* Number of subdomains into which the total domain will be
divided for parallel simulation. */
numberOfSubdomains 4; // Is the multiplication of n (x y z)

// Specifies the decomposition method to be used.
method simple; //ptscotch; //hierarchical; //metis; // parMetis;

// Specific parameters for the simple method.
simpleCoeffs
{
 // defines the partitioning of the domain in each direction (x, y, z).
 n               (2 1 2);
 /* Number used to avoid rounding issues when cells are assigned to
 subdomains. If delta is too large, it might cause the decomposition to be
 less balanced, meaning some cells might be incorrectly assigned,
 leading to an uneven workload across processors. */
 delta           0.001;
}

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

metisCoeffs
{
 processorWeights ( 1 1 1 1 );
}

manualCoeffs
{
 dataFile        "";
}
```

```
53
54  // Indicates whether the decomposition is distributed in multiple machines.
55  distributed     no;
56
57  /* Specifies the root directories if the simulation is distributed
58  across multiple machines. */
59  roots           ( );
60
61  // ************************************************************* //
```

3. Execute the "decomposePar.exe" command. New folders (processorX) will be created, according to the "numberOfSubdomains" configured.

4. Execute the "mpiexec.exe -np X snappyHexMesh -parallel" command, to run the snappyHexMesh meshing utility in parallel across X processors. **or use "mpiexec.exe -np X snappyHexMesh -parallel > snappyHexMesh.log" to write a log file with the meshing instruction in the working directory.**

5. Execute the "reconstructParMesh.exe" command, to reconstruct every single process: "castellatedMesh", "snap" and "addLayers" (folders 1,2,3). **or use "reconstructParMesh.exe -latestTime" to reconstruct the mesh corresponding to the latest time step.**

Chapter 11

Basic OpenFOAM Concepts

- **patch** It's an interface or boundary between a region and its surroundings in a simulation domain. It is defined as an area of the mesh or wall where boundary conditions can be applied, specifying values for velocity, pressure, temperature, or other flow variables. Patches are used to represent the conditions at the boundaries of the simulation domain. They can represent solid walls, flow inlets, flow outlets, symmetries, material interfaces, among others. Each patch has a specific type that determines how the boundary conditions are defined at that location.

- **wall** A type of boundary condition used to represent solid surfaces within the simulation domain. These surfaces interact with the fluid flow in a specific way, usually by enforcing a no-slip condition.

- **symmetry** A type of boundary condition used to define a plane where the flow and other variables are mirrored, ensuring no fluid or scalar quantity crosses the plane. This means the velocity and gradients perpendicular to the plane are zero.

Chapter 12

ParaView basic tools

ParaView is an application for scientific data visualization and analysis, designed especially for handling data from numerical simulations and scientific experiments. It's a powerful open-source tool that enables researchers and scientists to visualize, analyze, and interpret complex 3D data.

Figure 12.1: ParaView logo

Listed below are some important features of the software.

- **Interactive Visualization:** Allows interactive visualization of complex datasets in 2D and 3D, including meshes, volumes, and particle data.
- **Support for Multiple Formats:** Can read a wide range of scientific data formats, including common formats used in computational simulation such as VTK (Visualization Toolkit), OpenFOAM, among others.
- **Analysis and Post-Processing:** Facilitates data analysis with tools to compute gradients, integrals, histograms, and more on the visualized data.

- **Advanced Rendering:** Offers advanced rendering techniques like lighting, shading, and special effects to enhance data visualization.
- **Intuitive Graphical Interface:** Has an intuitive graphical user interface (GUI) that makes it easy to explore data and configure complex visualizations.

Below are the keys and commands to use and steps to perform specific tasks in ParaView 5.8.0. Each instruction assumes that the software is already open.

KEYS AND COMMANDS TO USE

- Scroll ↑ Zoom in.
- Scroll ↓ Zoom out.
- Scroll button + move the mouse Activates the panning or 3D scene scrolling function.
- Left click + move the mouse Activates the 3D rotation.

12.1 Visualize different geometries in STL

Go to: File → Open → Navigate to the folder where the STL files are located → In "Files of type", select "All Files (*)" → Select the desired files to display → STL Reader → OK → In the "Properties" panel, click "Apply".

12.1.1 Axes grid of geometry

After completing the steps in section 12.1 → in the "View" panel, check the "Axes Grid" box.

12.1.2 Camera parallel projection

After completing the steps in section 12.1 → in the "View" panel, check the "Camera Parallel Projection" box.

12.1.3 Modify opacity to display geometries

After completing the steps in section 12.1 → in the "Pipeline Browser", select the STL geometry to modify its opacity → in the "Display" panel, adjust the "Opacity" bar.

12.1.4 Modify STL color to display geometries

After completing the steps in section 12.1 and 12.1.3 → in the "Pipeline Browser", select the STL geometry to modify its color → in the "Display" panel, "Coloring", select "Solid Color" → Press the "Edit" button 

and select a color.

12.2 Displaying the project for post-processing

Go to: File → Open → Navigate to system folder project → In "Files of type", select "All Files (*)" → Select the controlDict file → OK → Select OpenFOAMReader → OK → Click on Apply in the Properties panel. Apply the Camera parallel projection process, see last section 12.1.2 step.

12.3 Displaying the last time step

After completing the steps in section 12.2 Go to: Time drop-down bar 

→ Select the last time step.
The same result can be achieved by using the time-step buttons located next to the time drop-down bar  .

12.4 Displaying the mesh cells

After completing the steps in section 12.2 and 12.3 Go to: Display panel → Change the Representation to Surface With Edges.

12.5 Measuring a distance

After completing the steps in section 12.2, 12.3 and 12.4, Zoom in to the element to measure → Select the Ruler tool  → Place the pointer over one of the vertices of the edge to be measured, click and press the P key → Repeat the previous step for the next vertex → In the Properties panel, click on Apply. **Note:** Measurements may not be exact, but if done correctly will show a good approximation.

12.5.1 Modify the measurement distance

After completing the steps in section 12.5, Click and hold the arrowhead and drag to the desired direction → click on "Apply".

12.6 Identifying surface coordinates

After created a mesh or performed a solver and completed steps in section 12.2 and 12.4 → Click on "Last Frame" button  , to visualize the last solution → Go to Properties panel → In Mesh Regions, check the geometry patches boxes → Click on Apply → In View panel check the Camera Parallel Projection box → Zoom in to the geometry → Click on "Hover Points On" button  and click on "OK" → Move to the vertex of interest to know its coordinates. **Note:** To explore another face, click again on "Hover Points On" button and zoom in to interest face.

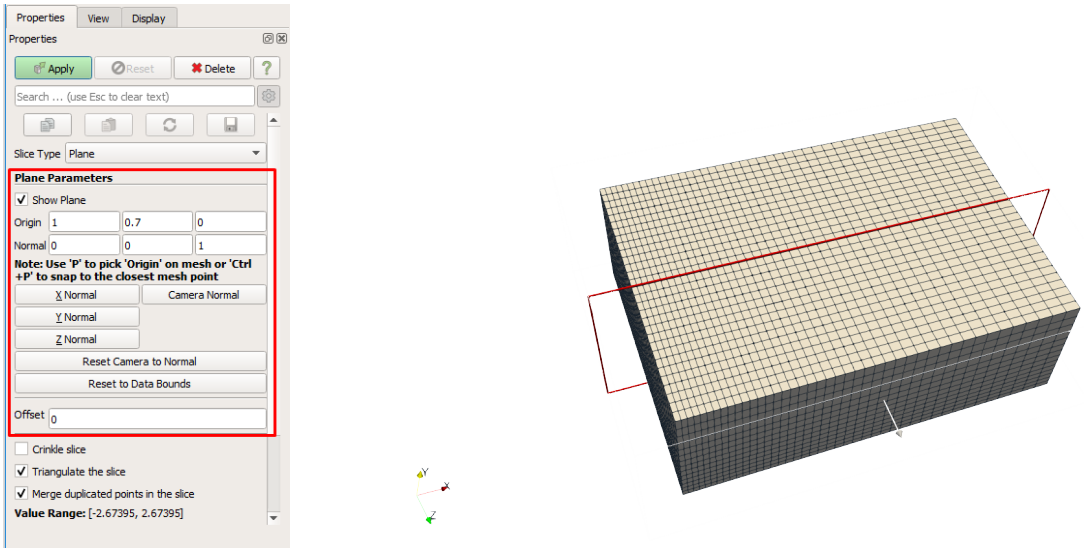
12.7 Creating a Slice

After completing the steps in section 12.2, 12.3 and 12.4 → Go to "Pipeline Browser" and select the mesh → Click on "Slice" button  → Modify the "Plane Parameters" in the "Properties" panel to obtain a desired view. See sections 12.7.1 and 12.7.2 → Click on "Apply"

12.7.1 Plane Parameters configuration, part 1

Figure 12.2a shows the first part of Plane Parameters and the settings available to display a plane.

- Box "Show Plane" enable or disable the plane to apply, as shown in Figure 12.2b
- "Origin" and "Normal" are settings to establish the desired plane
- "X Normal", "Y Normal" and "Z Normal" position the plane normal to these axes.



(a) Plane Parameters 1 (b) Mesh to apply the plane

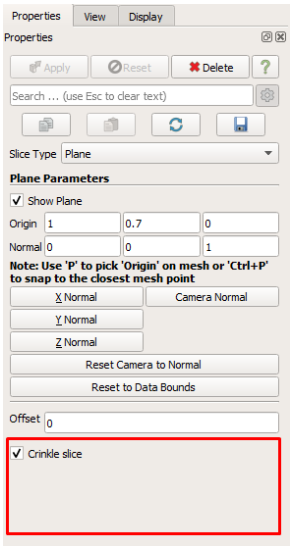
Figure 12.2: Plane Parameters configuration, part 1

12.7.2 Plane Parameters configuration, part 2

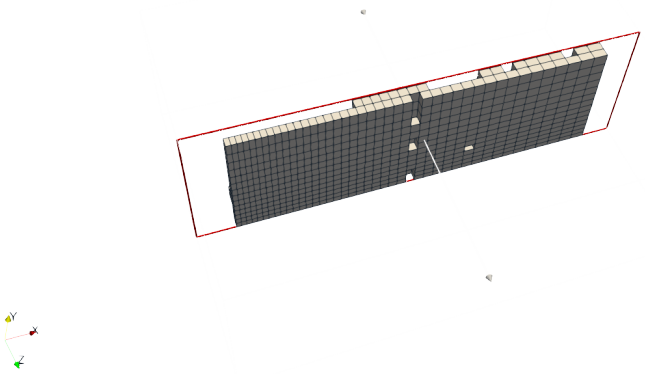
Figure 12.3a and 12.3c shows the second part of Plane Parameters.

- Box “Crinkle slice” displays a 3D representation of the plane, as shown in Figure 12.3b.

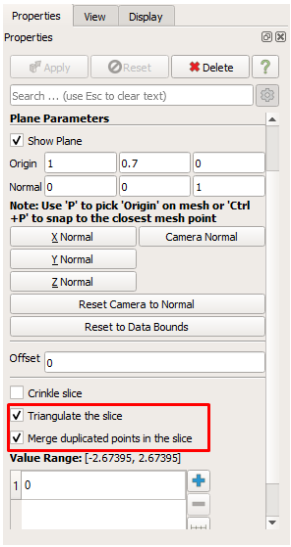
- Box “Triangulate the slice” displays a 2D representation of the plane with triangulation, as shown in Figure 12.3d.



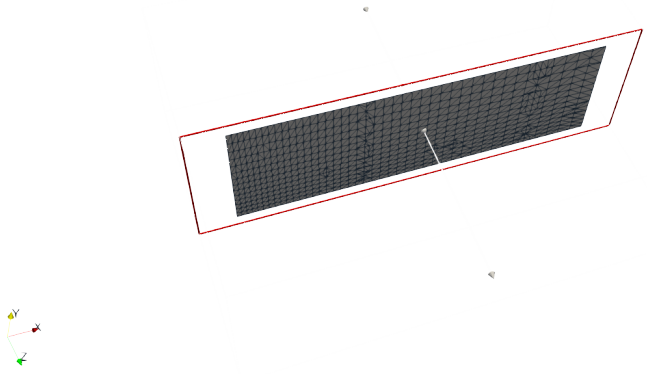
(a) Plane Parameters 2



(b) Crinkle plane



(c) Plane Parameters 3

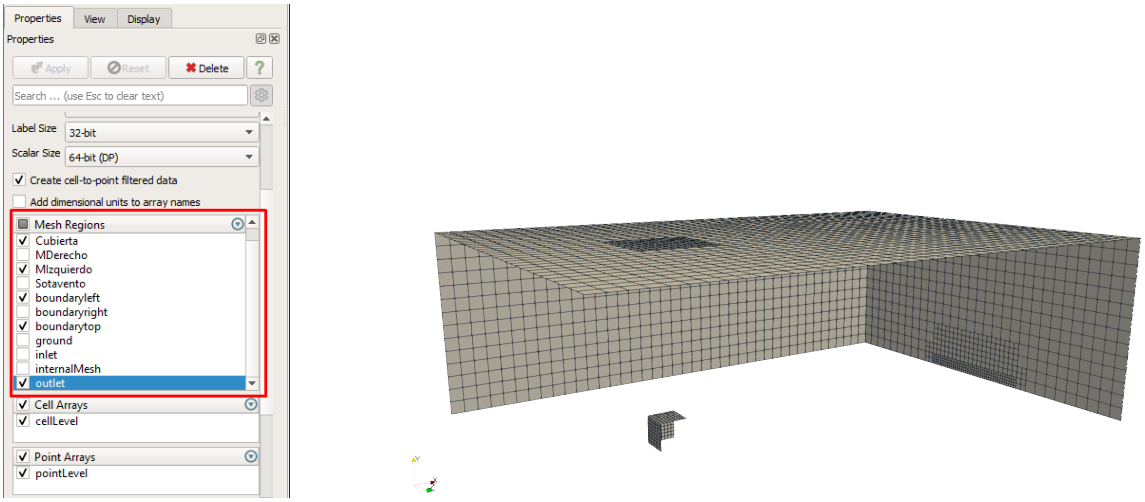


(d) Triangulated plane

Figure 12.3: Plane Parameters configuration, part 2

12.8 Visualization of patches created in the blockMesh and snappyHexMesh processes

After completing the snappy process, see section 5.3, follow steps 12.2, 12.3, 12.4 → In Properties panel check the desired patches to visualize (uncheck "internalMesh") → Click on "Apply".



(a) Patch selection (b) Patch visualization

Figure 12.4: Visualization of patches

12.9 Saving a Screenshot

Go to: File → "Save Screenshot..." → Go to the folder where you want to save the screenshot, type the desired name of the file and select the file extension in "File name:" and "Files of type:" → Click on "OK" → Modify "Size and Scaling", "Coloring" and "Compression Level".

Figures, like 12.2b use "Transparent Background" in the "Coloring" section.

Bibliography

[1] CFD SUPPORT, *CFD SUPPORT*, 2009, https://www.cfdsupport.com/, accessed: 25.06.2024.

[2] Open FOAM, *Mesh generation with the snappyHexMesh utility*, 2004, https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.4-mesh-generation-with-the-snappyhexmesh-utility, accessed: 01.07.2024.

[3] Open FOAM, *Specification of patch types in OpenFOAM*, 2004, https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.2-boundaries, accessed: 27.07.2024.

[4] Open FOAM v11 User Guide, *5.5 Mesh generation with the snappyHexMesh utility*, 2023, https://doc.cfd.direct/openfoam/user-guide-v11/snappyhexmesh, accessed: 28.07.2024.

[5] Aidan Wimshurst, *Fluid Mechanics 101 Calculators and Tools*, Fluid Mechanics 101, 2021, https://www.fluidmechanics101.com/pages/tools.html, accessed: 01.08.2024.

[6] Computational Fluid Dynamics Blog - LEAP Australia, *Tips and Tricks: Estimating the First Cell Height for correct Y+*, 2021, https://www.computationalfluiddynamics.com.au/tips-tricks-cfd-estimate-first-cell-height/, accessed: 01.08.2024.

[7] Cengel, Y. A., and Cimbala, J. M., *Mecánica de Fluidos, Fundamentos y Aplicaciones (Cuarta ed.)*, Ciudad de México: Mc Graw Hill, 2018

[8] Bert Blocken, Ted Stathopoulos, Jan Carmeliet. CFD simulation of the atmospheric boundary layer: wall function problems. *Atmospheric Environment*, 4, 238-252. 2007.

[9] SIMSCALE, *What is y+ (yplus)?*, 2018, https://www.simscale.com/forum/t/what-is-y-yplus/82394, accessed: 04.08.2024.

[10] SIMSCALE, *Mesh Quality*, 2024, https://www.simscale.com/docs/simulation-setup/meshing/mesh-quality/, accessed: 09.08.2024.

[11] CFD SUPPORT, *addLayersControls*, 2009, https://www.cfdsupport.com/openfoam-training-by-cfd-support/node128.html, accessed: 11.08.2024.