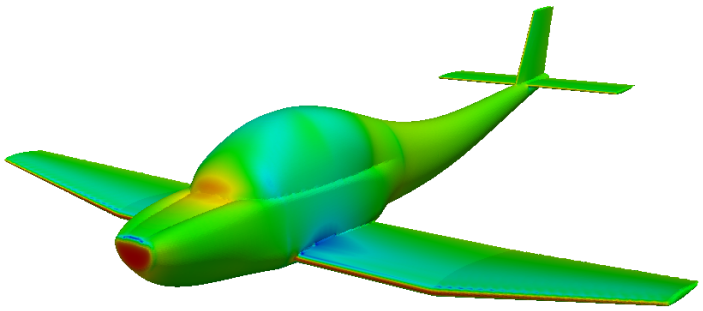
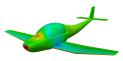
OpenFOAM GUIDE FOR BEGINNERS

OpenFOAM®



Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa
UNIVERSITAT POLITÈCNICA DE CATALUNYA



This guide has been developed by:

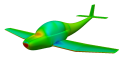
Jordi Casacuberta Puig

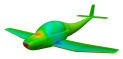
In association with:

Pedro Javier Gamez and Gustavo Raush

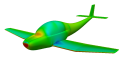
The Foam House ETSEIAT-UPC
Barcelona June 2014

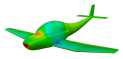
Contents





List of Figures





List of Tables

1. Introduction

1.0.1 About *OpenFOAM*®

OpenFOAM is first and foremost a *C++ library*, used primarily to create executables, known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific problem in continuum mechanics; and *utilities*, that are designed to perform tasks that involve data manipulation. The OpenFOAM distribution contains numerous solvers and utilities covering a wide range of problems.

One of the strengths of OpenFOAM is that new solvers and utilities can be created by its users with some pre-requisite knowledge of the underlying method, physics and programming techniques involved.

OpenFOAM is supplied with pre- and post-processing environments. The interface to the pre- and post- processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in Figure 1.1 [1]:

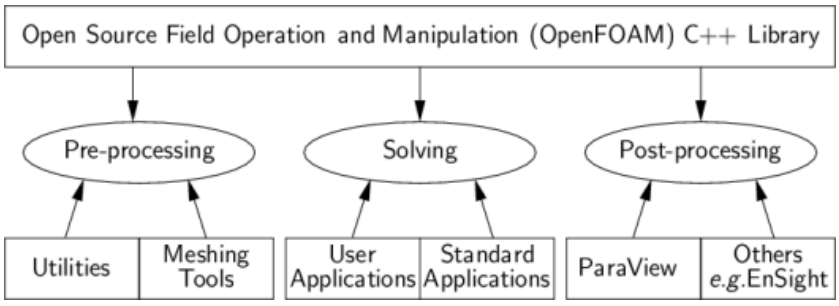
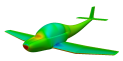


Figure 1.1: Overview of OpenFOAM structure, extracted from [1]

1.0.2 About this guide

The OpenFOAM guide developed in this project allows new users to establish and extend their OpenFOAM background once the main tutorial of the official guide is



done. Then with the present guide it will be possible to improve comprehension of the OpenFOAM structure, learn programming techniques, understand how to mesh different kinds of geometries (2D and 3D), acquire familiarity with the main pre- and post-processing OpenFOAM and ParaView capabilities, figure out which solvers and physical models are more adequate for each kind of fluid mechanics problem, and much more. It should then be much easier to use complex utilities found in Internet and follow specific tutorials focused on advanced tools.

The current guide studies and exposes five different types of solved fluid mechanics problems with high applicability potential and all of them included in the same document. It has been designed to guide the user throughout the cases, starting by simple ones and following an increasing degree of difficulty. The guide offers theoretical background before developing each case, includes OpenFOAM codes needed for the simulations, gives explanations of the main physical models required for the resolution of each case, incorporates advice against typical pitfalls, and shows different and relevant OpenFOAM utilities in each chapter.

This product is not approved or endorsed by ESI Group. There is no attempt to profit from this guide; it only aims to assist new users and facilitate learning of the main OpenFOAM characteristics.

This guide is given fully for free. The author will not be responsible for any harm of any kind that these codes and their uses may cause. Readers may use the codes under their own responsibility and risk.

Send feedback to casacuberta.puig@gmail.com if you wish. I hope you find it useful.

1.0.3 Notes

In this project, Version 2.2.1 of OpenFOAM has been used.

All the plots are in SI units, except for the airfoil and the aircraft cases, where the angle α of attack is expressed in degrees.

2. Plane-parallel plates laminar flow

2.1 Description of the case

This first tutorial studies the flow between two plane-parallel plates separated by a distance h so that their length and depth are big compared to their heigh. Its simple analytical solution allows the user to check the results obtained with $OpenFOAM^{®}$. Therefore it is an interesting case to start familiarization with this CFD software.

2.2 Hypotheses

- Incompressible flow
- Viscous flow
- Newtonian flow
- Bidimensional flow ($\frac{\partial}{\partial z} = 0$)
- Flow velocity parallel to the plates ($u \neq 0$ but $v, w = 0$ if the *infinite plates* assumption is made)
- Negligible gravitatory effects
- Fully developed and stationary flow ($\frac{\partial}{\partial t} = 0$)

2.3 First case: plane-parallel plates with relative movement (Couette flow)

2.3.1 Physics of the problem

In this first case, the top plate is moving with a horizontal velocity V relative to the other and there is no pressure gradient ($\frac{\partial p}{\partial x} = 0$). The axes are located in the middle of the plates.

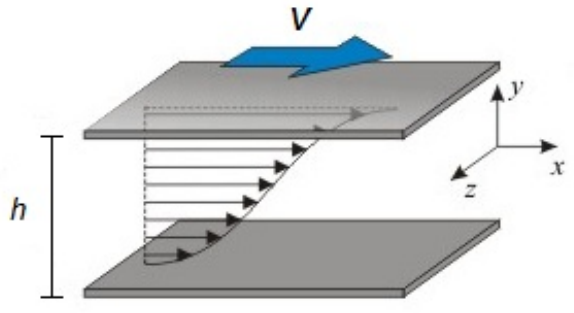


Figure 2.1: Viscous incompressible flow between two plane-parallel plates with relative movement

The continuity equation reads

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \Rightarrow \frac{\partial u}{\partial x} = 0 \Rightarrow u = u(y). \tag{2.1}$$

The momentum equation in the x-axis,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{\mu}{\rho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{2.2}$$

is reduced to

$$\frac{d^2 u}{dy^2} = 0 \Rightarrow u = C_1 y + C_2.$$

Applying the boundary conditions $u = V$ in $y = \frac{h}{2}$ and $u = 0$ in $y = -\frac{h}{2}$, the analytical solution takes the form

$$u = \frac{V}{h}y + \frac{V}{2}, \quad -\frac{h}{2} \leq y \leq \frac{h}{2}. \tag{2.3}$$

This is the general solution of the *Couette flow* due to a mobile wall. Hence the

velocity profile is linear.

For the resolution with *OpenFOAM*®, h is given a value of 0.1 m and the plates' length (l) is set to 2 m. The mobile wall has a horizontal velocity of $V = 1$ m/s.

2.3.2 Pre-processing

The first step to start the *OpenFOAM*® simulation is to create a directory which will contain all the required files. Some of them are going to be created by the user to setup the pre-processing, some of them will be automatically generated by the application and some of them will be necessary during post-processing.

The user has to open the Linux terminal and write

```
mkdir -p FoamCases/ppWall
```

With this instruction, the user creates a general directory called FoamCases containing the ppWall subdirectory, where all the files for the *Couette flow* simulation will be stored. From now on, the global FoamCases directory will contain all the cases that will be solved in this guide.

Within ppWall, there must be three main subdirectories: *0*, *constant* and *system*. The first one controls the initial field data ($t = 0$); the second one contains a full description of the case mesh and the physical properties for the application concerned, and the third one controls the setting parameters associated with the solution procedure itself.

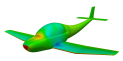
To create all of them, the user has to access ppWall:

```
cd FoamCases/ppWall
```

and type

```
mkdir 0 constant system
```

To check the results, the user can observe the new subdirectories by typing the `ls` command in the terminal.



2.3.2.1 Mesh generation

Although the physical problem has been defined two-dimensional, *OpenFOAM*® operates in a three-dimensional Cartesian coordinate system. Thus, the domain of ppWall will be a $2 \times 0.1 \times 0.01$ meters rectangular prism, where an arbitrary depth (for example 0.01 m) has been set. Later, with appropriate boundary conditions on the external x-y planes, the case will be solved bidimensionally. The domain of ppWall can be observed in Figure 2.2.

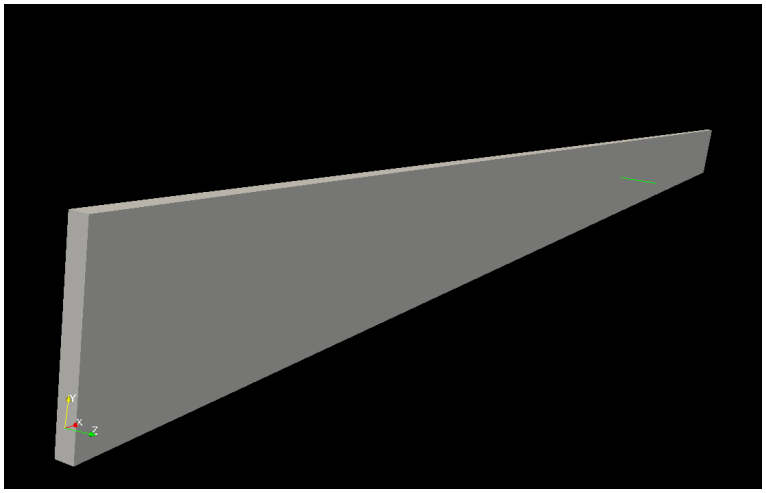


Figure 2.2: Domain of the *Couette flow* case

To solve CFD problems, it is necessary to discretize the domain in cells, that is, to *mesh* the geometry of study. *OpenFOAM*® has its own meshing tool blockMesh, which generates meshes from a description specified in an input dictionary called *blockMeshDict*. This dictionary is contained in a subdirectory within *constant*, called *polyMesh*.

Type

```
cd constant
```

to access the *constant* directory, and then

```
mkdir polyMesh
```

to set up the required configuration.

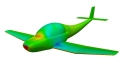
Advice:

If the user wants to move back to the previous directory, it can be done by typing

```
cd ..
```

For this case, the *blockMeshDict* entries are:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
   (0 0 0)
   (20 0 0)
   (20 1 0)
   (0 1 0)
   (0 0 0.1)
   (20 0 0.1)
   (20 1 0.1)
   (0 1 0.1)
);

blocks
(
   hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
   top
   {
       type wall;
       faces
       (
            (3 7 6 2)
       );
   }
```



```
50      bottom
51      {
52          type wall;
53          faces
54          (
55              (1 5 4 0)
56          );
57      }
58      inlet
59      {
60          type patch;
61          faces
62          (
63              (0 4 7 3)
64          );
65      }
66      outlet
67      {
68          type patch;
69          faces
70          (
71              (2 6 5 1)
72          );
73      }
74      frontAndBack
75      {
76          type empty;
77          faces
78          (
79             (0 3 2 1)
80             (4 5 6 7)
81          );
82      }
83  );
84
85  mergePatchPairs
86  (
87  );
88
89  // ************************************************************************* //
```

The user has to copy the previous code in a new gedit document. gedit is a text editor that can be found as a Linux Application (it is located in Ubuntu's main repository and is installed by default). Nevertheless, it can be installed again by typing in the terminal

```
sudo apt-get install gedit
```

There are other editors of choice to edit the case files, such as emacs, vi, kate, etc.

Caution:

> The numbers on the left do not have to be included in the code; they are used in this guide to facilitate citation

Once the *blockMeshDict* code has been copied, it is necessary to save it inside *constant/polyMesh* by selecting this directory using the *Save as...* option in the gedit menu. To summarize, there must be three directories inside ppWall up to now, namely *0*, *system* and *constant*, the latter with another subdirectory called *polyMesh*, which has a gedit sheet inside named *blockMeshDict* with the previous code copied.

Advice:

> To facilitate and optimize your programming procedure, it is recommended that you have templates of $OpenFOAM^{®}$ files for the different types of configurations. The header, which needs to appear in all the files, is basically equal for all of them. It can be found in Internet or in $OpenFOAM^{®}$ official tutorials

The specifications in the *blockMeshDict* file are the following:

- convertToMeters
  It is a scaling factor for the coordinates of the vertices. The products of the coordinates by this factor will be understood as the desired spatial magnitudes expressed in SI units.

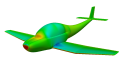
  Example: For a vertex coordinate equal to (20 0 0) *and a convertToMeters factor equal to* 0.1*, the point is located* 2 *m from the origin in the* x*-direction.*

- vertices
  It contains the list of vertices of the block or blocks in which the case geometry is divided. In the current case, the eight coordinate triples of the vertices of the $2 \times 0.1 \times 0.01$ meter rectangular prism of the domain have been written.

  Caution:

  > The order in which the vertex coordinates are written is relevant. The first coordinate triple becomes vertex number 0, the second one becomes vertex number 1, etc. This numbering will be used in future instructions

- blocks
  It contains the block definitions. Each block definition is a compound entry



consisting of a list of the vertex labels which define the block, a vector giving the number of cells required in each direction and the type and list of cell expansion ratio in each direction. There are as many block definitions as blocks the user wants to create.

Example: For the case of ppWall *where only one block is needed, the three sub-instructions are:*

```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
```

The first factor of the entry (**vertex numbering**) follows specific rules:

- Each number represents one vertex, for which the number is the position (starting from zero) in which the vertex is written in the vertices instruction
- Each block has a a local coordinate system (x_1 x_2 x_3) that must be right-handed
- The axis origin is the first entry in the block definition (vertex 0 in the example)
- The x_1 direction is described by moving from vertex 0 to the vertex written in the second position (vertex 1 in the example)
- The x_2 direction is described by moving from vertex 1 to the vertex written in the third position (vertex 2 in the example)
- Vertices 0, 1, 2, and the vertex written in the fourth position (vertex 3 in the example) define the plane $x_3 = 0$
- The vertex written in the fifth position (vertex 4 in the example) is found by moving from vertex 0 in the x_3 direction
- The vertices occupying the sixth, seventh and eighth positions (vertices 5, 6 and 7 in the example) are similarly found by moving in the x_3 direction from vertices 1, 2 and 3 respectively

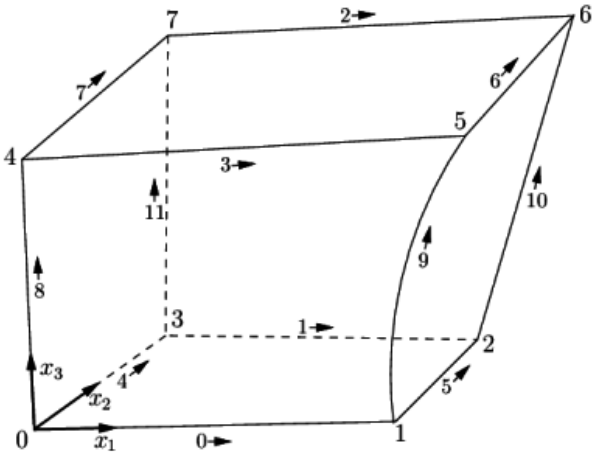


Figure 2.3: Specifications of a single block

The second factor of the entry (**number of cells**) gives the number of cells in each of the x_1 x_2 x_3 directions for that block.

Example: In the current ppWall *case, if the instruction is* (20 20 0) *then the* 2 *meter length of the prism in the* x_1*-direction of the local axis will be divided in* 20 *cells of* 0.1 *meters each. Note that, as the case is two-dimensional, there is no need of meshing along the* x_3*-axis.*

The third factor of the entry (**cell expansion ratios**) gives the cell expansion ratios for each direction in the block. The expansion ratio enables the mesh to be graded or refined in specified directions. The ratio is that of the width of the end cell δ_e along one edge of a block to the width of the start cell δ_s along that edge, as shown in Figure 2.4. In the current case, the expansion ratio equals 1 in the three spatial directions. Thus, no graded mesh will be created.

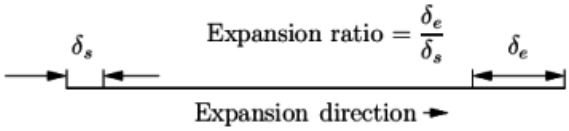
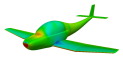


Figure 2.4: Mesh grading along a block edge

- edges

  Each edge joining two vertex points is assumed to be straight by default. However, each edge may be specified to be curved by entries in a list named



edges. In the current case all the edges are not curved, so the list may be omitted.

- boundary

  It contains the keywords (user's choice) which define each one of the regions (or patches) in which the whole boundary is divided and information about these regions (type of patch and faces contained in the region).

  Example: For the case of ppWall, there are five patches which need to be identified and determined. top and bottom refer to the walls with relative movement; inlet and outlet are the regions by which the flow enters and leaves, and frontAndBack refers to the two external x-y planes which are enclosed in a same set. This can be done because these two faces determine the bidimensional behaviour, thus they can be identified as a whole group. This patch distribution is shown in Figure 2.5.

  Once all the regions have been identified, it is necessary to determine their patch type. Although there are seven different types of patch descriptions, in ppWall one finds wall, patch and empty. The first one is applied in regions whose behaviour is characterized by the physical boundary conditions of a wall (for instance the plane-parallel plates of the *Couette flow*). The second one, patch, is used as a generic patch when it does not contain any geometric or topological information about the mesh.

  Example: In the ppWall case, the regions associated with the generic patch type are inlet and outlet. They contain no geometric information of the mesh but cannot be associated to the wall boundary conditions. The third type, empty, is applied for the front and back planes of a 2D geometry.

  Advice:

  > Use logical and easily interpretable names for the patches (following the physical specifications). It will help in setting the boundary conditions as well as finding programming errors

  The last step is to determine which faces of the block or blocks are associated to each one of the patches. After assigning the keyword of the patch and specifying the patch type, the faces have to be enclosed between parentheses. Each parenthesis contains the four vertices of the face associated to that patch, and in every patch type there are as many parentheses as faces are associated to this region.

  Caution:

The order in which the four vertices are written in each parenthesis is meaningful. This order must be such that, looking from inside the block and starting from any vertex, the face must be traversed in a clockwise direction to define the other vertices

- mergePatchPairs
  A mesh can be created using more than one block. When a connection between blocks is needed, there are two possibilities. The first is **face matching**, by which the connected faces are formed from the same set of vertices. In that case, the two patches that form the connection should simply be ignored from the patches list. The second possibility is **face merging**, by which a group of faces from a patch from one block are connected to another group of faces from a patch from another block, to create a new set of internal faces. In the ppWall case there is only one block, thus it may be omitted.

After completing *BlockMeshDict*, the patch configuration that will be used for setting the boundary conditions and to run the case can be seen in Figure 2.5.

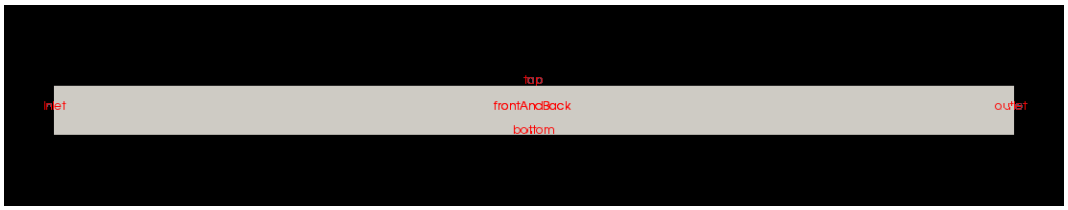


Figure 2.5: Patches defined in the domain of ppWall

The user will be able to view the mesh in 2.3.3 once all the required files are set. However, the mesh is presented in advance in Figure 2.6 for a better understanding of the case.

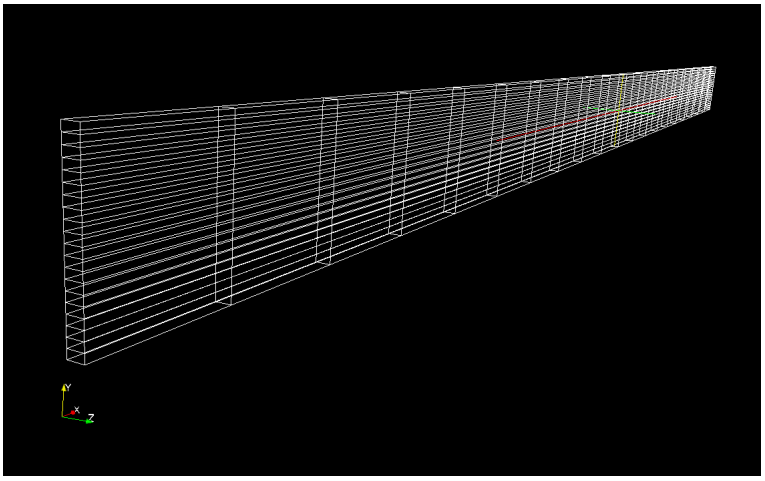


Figure 2.6: Initial mesh of the ppWall case

2.3.2.2 Boundary and initial conditions

Once the mesh is generated, the next step is to set up the initial field conditions of the case. The case is set up to start at time $t = 0$ s, so the initial field data are stored in the *0* subdirectory, as explained in 2.3.2. The *0* subdirectory contains two files, *p* and *U*, representing the pressure (p) and velocity ($\mathbf{U}$) fields.

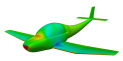
As done with *blockMeshDict* in 2.3.2.1, the user has to copy two gedit sheets inside the *0* subdirectory. The first must be named *p* and contains the following instructions:

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      p;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     top
```

```
24      {
25          type          zeroGradient;
26      }
27
28      bottom
29      {
30          type          zeroGradient;
31      }
32
33      inlet
34      {
35          type          fixedValue;
36          value         uniform 0;
37      }
38
39      outlet
40      {
41          type          fixedValue;
42          value         uniform 0;
43      }
44
45      frontAndBack
46      {
47          type          empty;
48      }
49  }
50
51  // ************************************************************************* //
```

The second file must be named *U* and contains the following instructions:

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volVectorField;
13      object      U;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions          [0 1 -1 0 0 0 0];
18
19  internalField       uniform (0 0 0);
20
21  boundaryField
22  {
23      top
24      {
25          type            fixedValue;
```



```
26          value           uniform (1 0 0);
27      }
28
29      bottom
30      {
31          type            fixedValue;
32          value           uniform (0 0 0);
33      }
34
35      inlet
36      {
37          type            zeroGradient;
38      }
39
40      outlet
41      {
42          type            zeroGradient;
43      }
44
45      frontAndBack
46      {
47        type              empty;
48      }
49  }
50
51
52  // ************************************************************************* //
```

The main specifications in the p and U files are:

- dimensions
  It specifies the dimensions of the field. Each position in the brackets represents a basic SI or USCS unit: mass, length, time, temperature, quantity, current and luminous intensity, as can be seen in Figure 2.7:

No.	Property	SI unit	USCS unit
1	Mass	kilogram (kg)	pound-mass (lbm)
2	Length	metre (m)	foot (ft)
3	Time	——— second (s) ———	
4	Temperature	Kelvin (K)	degree Rankine (°R)
5	Quantity	kilogram-mole (kgmol)	pound-mole (lbmol)
6	Current	——— ampere (A) ———	
7	Luminous intensity	——— candela (cd) ———	

Figure 2.7: Base units and nomenclature for SI and USCS, extracted from [1]

  Each position contains the exponent of the corresponding unit, which can be negative or positive.

*Example: As the velocity field (**U**) has units $m/s = m^1 s^{-1}$ and the length and time occupy the second and third positions respectively, the first line of the U file must be*

```
dimensions [0 1 -1 0 0 0 0]
```

Caution:

> As can be seen in the p file, the units are [0 2 -2 0 0 0 0], which represents $m^2 s^{-2}$. However, it is known that in the SI the units of the pressure are $Pa = kg^1 m^{-1} s^{-2}$. This is due to the fact that *OpenFOAM*® works with *kinematic pressure* $\left(\frac{p}{\rho}\right)$. As a consequence, this fact has to be considered in all the pressure values introduced in the pre-processing or obtained in the post-processing.

- internalField

  This specification contains the internal field data, which represents the value of the internal field at the initial time of the simulation. It can be started at a random initial value, which will evolve through the future timesteps according to the boundary conditions.

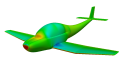
  Example: Since the case deals with an incompressible flow, the absolute value of the pressure field is not relevant, so it is set to uniform 0 for convenience.

- boundaryField

  It is the instruction for the boundary field data that includes boundary conditions and data for all the boundary patches. Although there are a great number of conditions to be applied at the boundaries, in the current case three of them have been used.

  The patches defined as top and bottom in the real physical problem are considered as solid walls. Therefore, the boundary conditions referring to velocity and pressure must be set accordingly: they must have an imposed fixed velocity value, while the pressure will vary according to the values of the internal field. In *OpenFOAM*®, the boundary condition to specify a fixed value in a patch is fixedValue, followed by an instruction indicating if it is unifom or nonuniform and giving its numerical value. This input may be a scalar or a vector, depending on the nature of the field.

  Example: To indicate that the superior plate (top patch) is moving at a speed of 1 m/s in the direction of the x axis, the instruction is



```
type fixedValue

value uniform (1 0 0)
```

as shown at lines 23 *through* 27 *in the* *U* *file.* All the velocity conditions in the top and bottom patches must have instructions like this (as can be seen at lines 29 through 33, where the instruction for the bottom plate is the same but with a numerical value of (0 0 0), meaning that the plate is motionless).

Regarding the pressure on top and bottom, as the user does not have to specify a value but it will be adapted at each timestep to the internal field, the zeroGradient boundary condition must be set. This means that the normal gradient of the pressure is zero in that patch. To impose zeroGradient, the instruction written in the required patches must be

```
type zeroGradient
```

as can be seen at lines 23 through 31 in the *p* file.

Caution:

> When introducing values for a vector field such as **U**, the vectors are being referred to the global coordinate system, not to the local coordinate system of the block

The patches defined as inlet and outlet in the real physical problem are the sections by which the fluid enters and leaves the space between the plates in the direction of the superior plate's displacement. Up to this point the First case and the Third case of ppWall differ. In the current case, the entire flow movement is caused by a relative displacement between the top plate and the bottom plate, whereas in the Third case a pressure gradient in the x-direction is superposed to the relative movement of the plates, contributing to the global flow dynamics. For this reason, to avoid any pressure gradient along the field, the inlet pressure and the outlet pressure must have the same value (for instance, zero). As it happened with the boundary conditions of the *U* file, to impose a fixed pressure value in a patch, the instruction is

```
type fixedValue
```

and to specify which and what kind (uniform or nonuniform) of fixed value is imposed, the instruction must be complemented with

```
value uniform 0
```

(note that now the field is scalar, with an imposed value of zero). The result can be seen in the *p* file at lines 33 through 43.

To determine the velocity boundary conditions in the inlet and outlet patches, as the value is unknown and it will be adapted to the internal field conditions, zeroGradient is used. Although the velocity is a vector field, the instruction is set exactly in the same way as it appears with pressure in the top and bottom patches of the *p* file, which is

```
type zeroGradient
```

This is shown at lines 35 through 43 of the *U* file.

Finally, it is necessary to impose boundary conditions to frontAndBack. As explained in 2.3.2.1, this patch does not contribute to the physical problem itself, but converts the case into a bidimensional one. On the faces forming this region, the velocity and pressure values are the ones that would exist if the case were extended infinitely on the normal direction of the front and back faces. In all cases, all the patches related to a 2D behaviour use the empty boundary condition. Therefore the common instruction to be used is

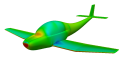
```
type empty
```

and it is set in the same way for the pressure (lines 45 through 48) and the velocity (lines 45 through 48).

2.3.2.3 Physical properties

The physical properties of the case are stored in dictionaries whose names are given the suffix ...*Properties*. The ppWall case will be solved using icoFoam as a solver, which is used for transient, incompressible, laminar flows of Newtonian fluids.

For an icoFoam case, the only property that must be specified is the kinematic viscosity wich is stored from the *transportProperties* dictionary. To set it within the case directory, access *constant* by typing



```
cd constant
```

Up to now, inside *constant* it should only appear the *polyMesh* subdirectory (it can be observed by typing `ls` in the console). At this moment, next to *polyMesh* (not within) the user has to create a new gedit sheet called *transportProperties* with the following code inside:

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              nu [0 2 -1 0 0 0 0] 0.01;

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

The first line of the file shows the only property that must be specified (nu refers to ν, the symbol by which the kynematic viscosity is represented). Next to it appear the units of the property (expressed following the same method as shown in 2.3.2.2). Finally, it is necessary to set the value of the property. The case will be run with a Reynolds number of 10 (laminar flow). The Reynolds number is defined as

$$Re = \frac{d|\mathbf{U}|}{\nu} \tag{2.4}$$

and as the characteristic length is $d = h = 0.1$ m and the characteristic velocity is $|\mathbf{U}| = 1$ m/s, ν has to be 0.01 in order to achieve a Reynolds number equal to 10. Note that this is a relatively high value of kinematic viscosity (water at 5ºC has a kinematic viscosity of $1.5 \times 10^{-6}\ m^2/s$). In order to avoid initial inestabilities, the Reynolds number is set very low, and therefore the viscous forces have to be much higher than the inertial forces.

2.3.2.4 Control

Input data related to the control of time and reading and writing of the solution data are read in from the *controlDict* dictionary. It can be seen as the case control file, and it is located in the *system* directory. From the case directory, type

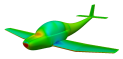
```
cd system
```

and, as done in previous sections, copy inside a gedit file named *controlDict* containing the following instructions:

Advice:

> In order to expedite these *copy/paste* procedures, the Linux commands `cp`, `mv` and `rm` can be used through the terminal. Write `mv --help` in the console for more information

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      controlDict;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 application         icoFoam;
19
20 startFrom           startTime;
21
22 startTime           0;
23
24 stopAt              endTime;
25
26 endTime             5;
27
28 deltaT              0.005;
29
30 writeControl        timeStep;
31
32 writeInterval       20;
33
34 purgeWrite          0;
35
```



```
writeFormat      ascii;

writePrecision   6;

writeCompression off;

timeFormat       general;

timePrecision    6;

runTimeModifiable true;

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

The specifications in the *controlDict* are:

- application
  It specifies the solver used for the current case. According to the physical characteristics of the *Couette flow* problem and the icoFoam working conditions given in 2.3.2.3, this application fits well for running the case.

- startFrom
  Indicates that the simulation will start at the time specified in startTime.

- startTime
  It is the time at which the simulation begins. If the user wishes to start the run at time $t = 0$, then $OpenFOAM^{®}$ needs to read field data from a directory named *0*, although it is not strictly necessary.

  Example: If the user had put p and U fields data in a directory named 1 instead of 0, the startTime would have to be set at 1, and therefore the simulation would have started at time $t = 1$.

  Advice:

  > At these first steps of the $OpenFOAM^{®}$ training, use *0* as field data directory name and startTime equal to 0 to avoid confusions

- stopAt
  Indicates that the simulation will finish at the time specified in endTime.

- endTime
  It is the time at which the simulation finishes. For the current case, the aim is to reach the steady state, which means stationary flow, as it indicates one of the hypotheses of the analytical resolution. Although icoFoam is a transient

solver (by considering that $\frac{\partial}{\partial t}$ schemes $\neq 0$), flow can reach the steady state if the case is set consequently (basically when no turbulence, time-variable boundary conditions or specific effects like von Kármán vortex sheet exist).

As a general rule, the fluid should pass through the domain 10 times to reach steady state in laminar flow. This would imply an endTime equal to 20. However, it has been seen that 5 is sufficient.

- deltaT

  It is the timestep. To achieve temporal accuracy and numerical stability when running icoFoam, a maximum Courant number of less than 1 is required. The Courant number is defined for one cell as

$$Co = \frac{\delta t |\mathbf{U}|}{\delta x} \tag{2.5}$$

  where δt is the time step, $|\mathbf{U}|$ is the magnitude of the velocity through that cell and δx is the cell length. The flow velocity varies across the domain and the user must ensure that $Co < 1$ everywhere. Therefore, it is necessary to set δt based on the worst case: the *maximum* Co corresponding to the combined effect of a large flow velocity and small cell size.

  Example: In the ppWall case, all cells have the same length in the x_1-direction and in the x_2-direction (as the mesh is not graded). Thereby, the minimum cell length is

$$\frac{h}{n} = \frac{0.1}{20} = 0.005 \text{ m}$$

  The maximum flow velocity appears at the top plate, where $|\boldsymbol{U}| = V = 1$ m/s, thus the maximum δt is

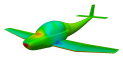
$$\delta t = \frac{Co\ \delta x}{|\mathbf{U}|} = \frac{1 \times 0.005}{1} = 0.005 \text{ s}$$

- writeControl

  It sets how the user decides the times at which the results are written. In the current case, the timeStep option is chosen, meaning that results are written at every nth time step, where n is specified under the writeInterval keyword.

- writeInterval

  The user has to decide how many results are written. Up to now, it is known that the simulation starts at time $t = 0$ s and finishes at time $t = 5$ s. However,



it is also necessary to define at what times between 0 s and 5 s the field data is given. At the end of the simulation, during the post-processing, the results of the simulation will be shown to the user at every time interval specified in the writeInterval instruction. So, with timeStep on, this time interval is defined by the product of deltaT times the input in the writeInterval instruction.

Example: If the user wants to write the results at times $t = 0.1, 0.2, \ldots, 5$ *s, and considering the fact that* deltaT $= 0.005$ *s,* writeInterval *must be* 20.

The remaining instructions are secondary and are not going to be explained in the guide. They refer to the writing format, allow data compression, etc.

2.3.2.5 Discretization and linear-solver settings

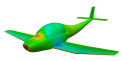
There must be two more files in the *system* directory. The first is called *fvSchemes* and specifies the choice of finite volume discretization schemes (for each one of the terms of the differential equations governing the problem). The second is called *fvSolution* and contains the specifications of the linear equation solvers and tolerances and other algorithm controls. Due to their complexity, it is not necessary to discuss all their entries at this initial stage of the *OpenFOAM*® training.

Copy the following codes in two files within *system*, the first called *fvSchemes* and the second called *fvSolution*.

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      fvSchemes;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 ddtSchemes
19 {
20     default         Euler;
21 }
22
23 gradSchemes
24 {
25     default         Gauss linear;
```

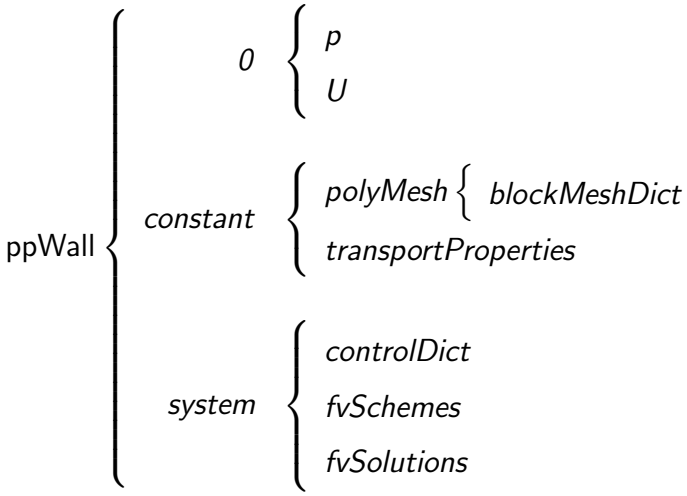
```
26      grad(p)         Gauss linear;
27  }
28
29  divSchemes
30  {
31      default         none;
32      div(phi,U)      Gauss linear;
33  }
34
35  laplacianSchemes
36  {
37      default         none;
38      laplacian(nu,U) Gauss linear orthogonal;
39      laplacian((1|A(U)),p) Gauss linear orthogonal;
40  }
41
42  interpolationSchemes
43  {
44      default         linear;
45      interpolate(HbyA) linear;
46  }
47
48  snGradSchemes
49  {
50      default         orthogonal;
51  }
52
53  fluxRequired
54  {
55      default         no;
56      p               ;
57  }
58
59
60  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "constant";
14      object      fvSolution;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  solvers
19  {
20      p
```



```
21      {
22          solver          PCG;
23          preconditioner  DIC;
24          tolerance       1e-06;
25          relTol          0;
26      }
27
28      U
29      {
30          solver          PBiCG;
31          preconditioner  DILU;
32          tolerance       1e-05;
33          relTol          0;
34      }
35  }
36
37  PISO
38  {
39      nCorrectors     2;
40      nNonOrthogonalCorrectors 0;
41      pRefCell        0;
42      pRefValue       0;
43  }
44
45  // ************************************************************************* //
```

Up to now, all the required files and dictionaries to run the case have been set. As a final summary and if all the steps have been followed properly, the user should have the following scheme in the ppWall directory:



2.3.2.6 Creating the mesh

The last step before completing the pre-processing is the generation of the mesh. Although its dictionary has been set in 2.3.2.1, the mesh is not physically created.

To create it, the user must run blockMesh on the *blockMeshDict* file. This is done by typing in the terminal

```
blockMesh
```

within the ppWall directory, and if no error(s) appear(s), some text is going to start running at the console, finishing with `End`. This means that the mesh has been created successfully. The user can check the instructions appearing in the terminal, which contain information about the patches, number of cells, number of faces, etc.

Caution:

> The user should note that within *constant/polyMesh* five new files have appeared. They contain information about the elements of the mesh and are generated automatically while running blockMesh

To obtain a more accurate background about the mesh quality and its geometrical properties, type

```
checkMesh
```

Advice:

> It is possible to redirect the information of blockMesh and checkMesh (as well as other instructions) to a file by typing
>
> ```
> blockMesh > log.blockMesh
> ```
>
> or
>
> ```
> checkMesh > log.checkMesh
> ```
>
> By doing this, two new files are created in the directory where the instructions are typed, named *log.blockMesh* and *log.checkMesh*. The user can check these files as much as necessary

checkMesh contains information about the mesh quality. Factors as Mesh non-orthogonality or Max skewness are examples of indicators of this quality (regarding cell geometry and connections between them). If the mesh is set correctly, at the end it should appear

```
Mesh OK.
```

2.3.3 Viewing the mesh

Before the case is run, it is a good idea to view the mesh in order to check for any errors. The mesh is viewed in ParaView, the post-processing tool supplied with *OpenFOAM*®. It can be started by typing in the terminal from within the case directory

```
paraFoam
```

This launches the ParaView window as shown in Figure 2.8. To the left of the user's screen there are the Pipeline Browser and the Object Inspector subwindows. The first one shows the tree with the development of the case geometry and its treatment. It provides an easy way to select and deselect its subparts and the specific utilities used during the post-processing. In the second one, the user can find the main viewing settings for the case geometry and mesh.

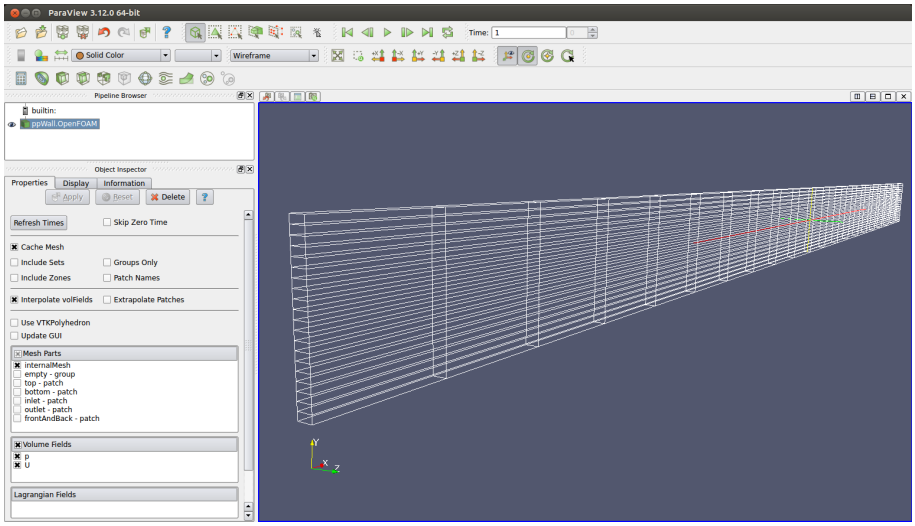


Figure 2.8: ParaView's window with the initial ppWall mesh

Caution:

> Before clicking the Apply button, the user needs to select some geometry from the Mesh Parts panel. It is located in the middle of the Object

Inspector subwindow. There, the user indicates what is going to be represented. It can be one or more patches (top, inlet, etc.) or the whole domain. By clicking the box adjacent to the Mesh Parts panel title, all the components are selected.

When the Apply button is clicked, it appears the case geometry. It is initialy represented as a solid surface. To view the mesh, the user has to select the Display panel located at the top of Object Inspector and then go to the Style panel. Among some types of configurations, it is possible to choose `Wireframe` or `Surface With Edges` to view the mesh. Furthermore, it is possible to set the colour of either the surface or the mesh by going to Color (also located in the Display panel), specifiying `Color by Solid Color`, clicking `Set Ambient Color` and selecting an appropriate colour.

Advice:

The user can check the measures of the domain and the mesh by selecting the option `Show cube axes` located in the middle of the Display panel

Advice:

It is possible to initialize ParaView but allowing the terminal to be operative, by typing

```
paraFoam &
```

instead.

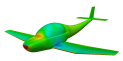
2.3.4 Running the application

To execute the icoFoam solver the user has to type in the case directory

```
icoFoam
```

The progress of the job is written to the terminal window. It tells the user the current time, maximum Courant number, initial and final residuals for all fields. As explained previously, it can be also redirected by typing

```
icoFoam > log.icoFoam
```



and then the user is able to check the results as much as desired.

Once the solver has ben run, it is possible to observe a set of new created directories within the ppWall case. Their nomenclature corresponds to each one of the time intervals defined by combining the deltaT and writeInterval instructions previously defined in *system/controlDict*. Within these new directories, it is contained the information related to **U** and p fields.

Caution:

> In case the user should want to rerun icoFoam with a different time configuration, erase all the created directories (except, of course, *0*, *constant* and *system*)

2.3.5 Post-processing

As soon as the results are written to time directories, they can be viewed using paraFoam. If previously icoFoam has been executed as a foreground process, restart paraFoam.

2.3.5.1 Viewing the results as isosurface and contour plots

To view the velocity, after clicking on Apply, the user should open the Display panel. To make a simple plot of velocity, the user should select the following: in the Style panel, choose `Surface` from the `Representation` menu and in the Color panel, select one of the two alternatives regarding **U** (the one with the cube icon or the one with the circular icon). With the first option, a single value for velocity will be attributed to each cell so that each cell will be denoted by a single colour with no grading. Instead, by applying the second option, the velocity field will be interpolated across each cell to give a continuous appearance.

Now in order to view the solution at $t = 5$ s, the user can use the `VCR Controls` of `Current Time Controls` to change the current time to 5. These are located in the toolbar below the menus at the top of the ParaView window (represented as green arrows). Using them, it is possible to access field data within the directories generated while icoFoam was running. At $t = 5$ s it is possible to observe a fully developed flow with a linear velocity trend (shown in Figure 2.9).

It is possible to include a colour bar representing the values acquired by the field with its corresponding numerical interpretation. To do this, click on the rainbow vertical bar located at the top-left corner of ParaView's window.

The following Figure shows the results of the **U** field by representing the module of the velocity.

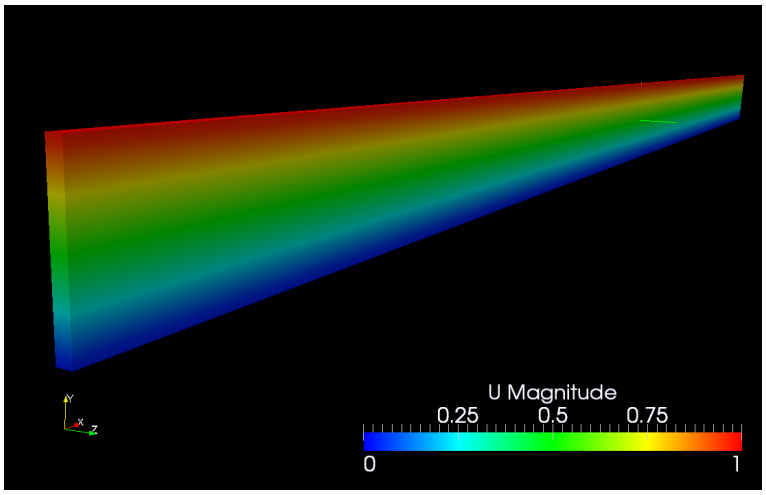


Figure 2.9: Velocity field obtained with icoFoam in the *Couette flow* case (m/s)

In the current case, it is also interesting to represent the projections of the total velocity on the x and y axis (U_x and U_y), instead of the module of the total velocity. In order to evaluate if the initial hypotheses have been correctly set (especially to prove that $v \approx 0$ if the *infinite plates* assumption is done), U_x should be almost equal to $|\mathbf{U}|$ and U_y should be nearly zero in all the domain. These variables can be shown by selecting them in the second drop-down menu next to the vertical rainbow icon (only if **U** is selected in the Display menu).

The same procedure can be done to obtain the p distribution along the domain. The user has to choose the desired representation (between the cube icon or the circular icon) in the Color panel within Display. The results are shown in Figure 2.10.

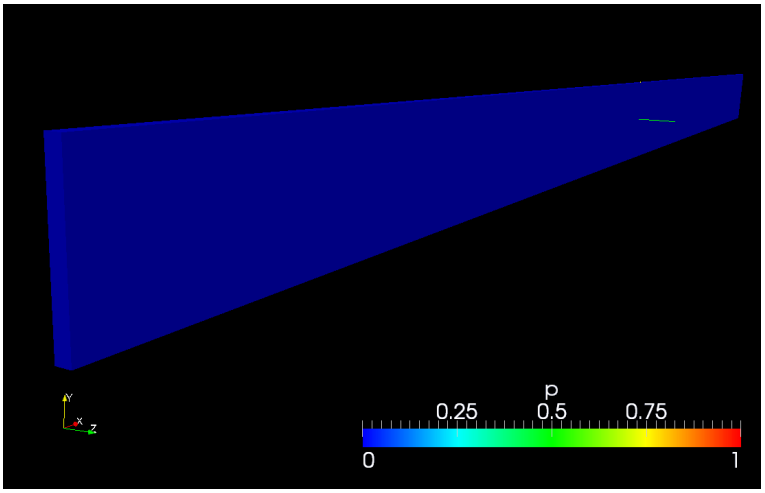


Figure 2.10: Pressure field obtained with icoFoam in the *Couette flow* case (m^2/s^2)

As one imposition in the *Couette flow* case is the fact that there is not a pressure gradient along the x-direction ($\frac{\partial p}{\partial x} = 0$), and therefore the boundary conditions have been set accordingly ($p = 0$ at the inlet and outlet), the pressure was expected to be constant in the whole domain. It can be observed in Figure 2.10.

Advice:

> Set the minimum and maximum values of legends coherently to avoid misleading conclusions derived from the colour distribution

2.3.5.2 Plotting variables in ParaView

Up to this moment, it has been possible to obtain a graphical representation of **U** and p fields. However, it has not been checked if the analytical and the numerical solutions coincide; although the velocity field seems to offer a linear trend, the user can plot it as a function of the y coordinate to validate the results.

To make a plot in ParaView, the user has to access `Plot Over Line` located in one of the menus at the top of the ParaView window. The user has to click on Filters -> Alphabetical -> Plot Over Line. Afterwards it is necessary to select the `Y Axis` option and click on Apply. The plot is shown and the user can modify its colour, variables to represent and scale as desired. In the *Couette flow* case, the representation of the variables along the y-axis is shown in Figure 2.11.

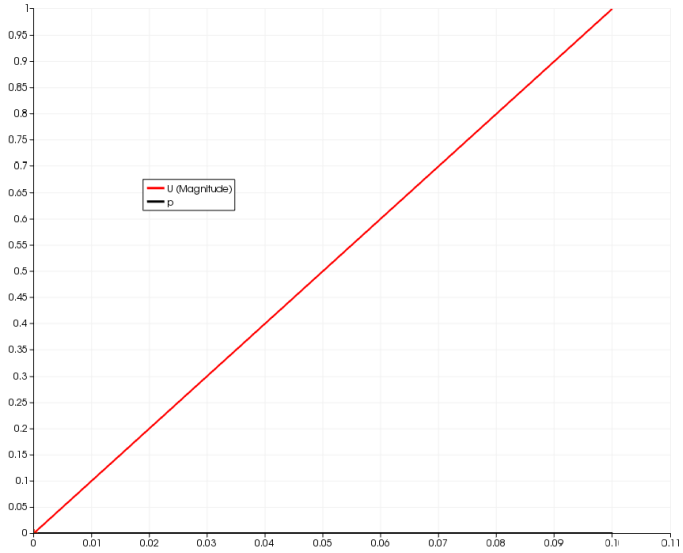


Figure 2.11: Distribution of $|\mathbf{U}|$ and p (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow* case

As it can be observed, the module of the velocity (remember that it was checked that ($|\mathbf{U}| \approx U_x$) offers a clear-cut linear trend, starting at (0 0) (the minimum velocity value a the bottom plate), finishing at (0.1 1) (the maximum velocity value at the top plate) and with a slope of 10. While, the pressure is constant, remaining zero in all the domain. Reminding Equation 2.3, the analytical solution for the horizontal velocity (expressed in axis located at the middle of the plates) is

$$u = 10y + \frac{1}{2}$$

which perfectly matches with the numerical results obtained with *OpenFOAM*®. It shows that the simulation has been correctly set, and that the assumptions done have been properly chosen.

2.4 Second case: plane-parallel plates with pressure gradient (plane-Poiseuille flow)

2.4.1 Physics of the problem

In the second case of Chapter 2, there is not a relative movement between the plates, but a pressure gradient along the x-axis ($\frac{\partial p}{\partial x} \neq 0$). For the calculations, the axes are

located in the middle of the plates.

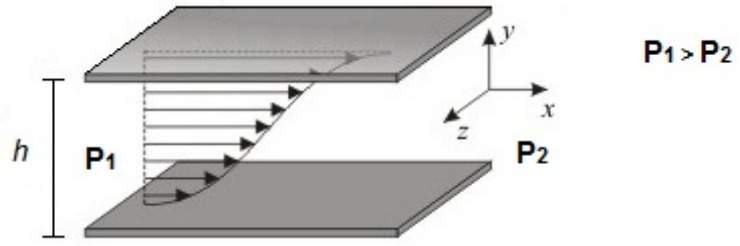


Figure 2.12: Viscous incompressible flow between two plane-parallel plates with a pressure gradient in the x-direction

By using the continuity equation (Equation 2.1),

$$u = u(y)$$

and simplifying the momentum equation in the x-axis (Equation 2.2),

$$\mu \frac{d^2u}{dy^2} = \frac{\partial p}{\partial x} \tag{2.6}$$

Equation 2.6 is slightly different to the solution of the *Couette flow* because in the current case a pressure gradient must be considered. Moreover, as $v = w = 0$ and according to the momentum equations in the y and z axis,

$$\frac{\partial p}{\partial y} = \frac{\partial p}{\partial z} = 0 \quad or \quad p = p(x)$$

A consequence of Equation 2.6 is the fact that $\frac{\partial p}{\partial x} = constant < 0$. It must be a constant because if two quantities are equal and the first is a function of y and the second is a function of x, then they must equal the same constant. Therefore, as the pressure is lineal in x, $\frac{dp}{dx}$ can be easily computed as $\frac{P_2 - P_1}{l}$.
The solution of Equation 2.6 can be found by integrating two times:

$$u = \frac{1}{\mu}\frac{dp}{dx}\frac{y^2}{2} + C_1 y + C_2 \tag{2.7}$$

Applying the boundary conditions $u = 0$ in $y = \frac{h}{2}$ and $u = 0$ in $y = -\frac{h}{2}$, the analytical solution takes the form

$$u = \frac{dp}{dx}\frac{1}{2\mu}\left(y^2 - \left(\frac{h}{2}\right)^2\right), \quad -\frac{h}{2} \leq y \leq \frac{h}{2} \tag{2.8}$$

This is the general solution of the *plane-Poiseuille flow* due to a pressure gradient: a parabolic velocity profile. Additionally, an interesting value to be computed is the maximum velocity corresponding to the vertex of the parabola, which is

$$u_{max} = -\frac{h^2}{8\mu}\frac{dp}{dx} \tag{2.9}$$

For the resolution with *OpenFOAM®*, h is given a value of 0.1 m and the plates length (l) is set to 2 m. The pressure gradient will be caused by applying vacuum at outlet while mantaining ambient pressure at inlet ($P_1 = 101325$ Pa and $P_2 = 0$ Pa).

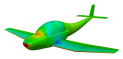
2.4.2 Pre-processing

To simulate the *plane-Poiseuille flow*, a new case is going to be created within FoamCases, named ppGrad. It is going to contain the same directories, subdirectories and files as ppWall but with some changes in the boundary conditions and the time control settings. Thus, *0/p*, *0/U* and *system/controlDict* are going to be modified. The dimensions of the domain and the mesh configuration are maintained.

2.4.2.1 Boundary and initial conditions

As in the current case there is not relative movement between plates, the first step is to set the velocity of the *top* patch equal to (0 0 0) (line 26 in *U* file).

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volVectorField;
13     object      U;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions          [0 1 -1 0 0 0 0];
```



```
18
19  internalField         uniform (0 0 0);
20
21  boundaryField
22  {
23      top
24      {
25          type              fixedValue;
26          value             uniform (0 0 0);
27      }
28
29      bottom
30      {
31          type              fixedValue;
32          value             uniform (0 0 0);
33      }
34
35      inlet
36      {
37          type              zeroGradient;
38      }
39
40      outlet
41      {
42          type              zeroGradient;
43      }
44
45      frontAndBack
46      {
47        type                empty;
48      }
49  }
50
51
52  // ************************************************************************* //
```

The second step is to apply the appropriate pressure conditions at the inlet and outlet. According to 2.4.1, the pressure gradient along the x-direction is going to be created by applying vaccum at the outlet ($p = 0$ Pa), while mantaining ambient pressure at the inlet ($p = 101325$ Pa)

Caution:

According to 2.3.2.2, the pressure at inlet must not be set as

`uniform 101325`

because *OpenFOAM*® is expecting an input of kinematic pressure $\left(\frac{p}{\rho}\right)$

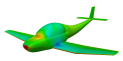
In *transportProperties*, it is now set that the kinematic viscosity (ν) equals 0.01 (m^2/s). It became so with the intention of introducing a very low Reynolds number, guaranteeing laminar flow in the whole domain. Now, a density of 1000 kg/m^3 is going to be used to introduce the desired ambient pressure value. With $\rho = 1000$ kg/m^3 and $\nu = 0.01$ m^2/s, the dynamic viscosity of this fluid equals $\mu = 10$ $Pa \cdot s$. According to these values and to have a greater physical meaning, these properties may correspond to honey.

Having said that, the user has to set the inlet pressure by introducing the instruction

```
uniform 101.325
```

at line 36 of *p* file.

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      p;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     top
24     {
25         type          zeroGradient;
26     }
27
28     bottom
29     {
30         type          zeroGradient;
31     }
32
33     inlet
34     {
35         type          fixedValue;
36         value         uniform 101.325;
37     }
38
39     outlet
```



```
40      {
41          type            fixedValue;
42          value           uniform 0;
43      }
44
45      frontAndBack
46      {
47          type            empty;
48      }
49  }
50
51  // ************************************************************************* //
```

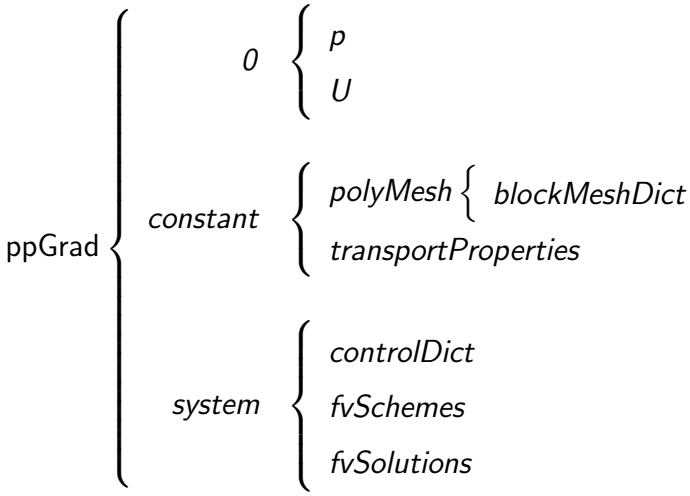
2.4.2.2 Control

Although this dictionary should not be necessarily changed when adapting the ppWall case to the ppGrad case, it has been seen that the convergence of ppGrad is faster. Therefore, to avoid time-consuming processes, endTime is going to be set to 2.5 (line 26 in *controlDict*), mantaining deltaT and writeInterval.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      controlDict;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  application         icoFoam;
19
20  startFrom           startTime;
21
22  startTime           0;
23
24  stopAt              endTime;
25
26  endTime             2.5;
27
28  deltaT              0.005;
29
30  writeControl        timeStep;
31
32  writeInterval       20;
```

```
33
34  purgeWrite          0;
35
36  writeFormat         ascii;
37
38  writePrecision      6;
39
40  writeCompression    off;
41
42  timeFormat          general;
43
44  timePrecision       6;
45
46  runTimeModifiable   true;
47
48  // ************************************************************************* //
```

Now the case is ready to be run. As the changes have only been applied to instructions within files, the scheme of directories and subdirectories is maintained.



2.4.3 Post-processing

2.4.3.1 Results of the simulation

Once icoFoam has ben executed and if all the files have been properly set, the results for $|\mathbf{U}|$ and p should be the following.

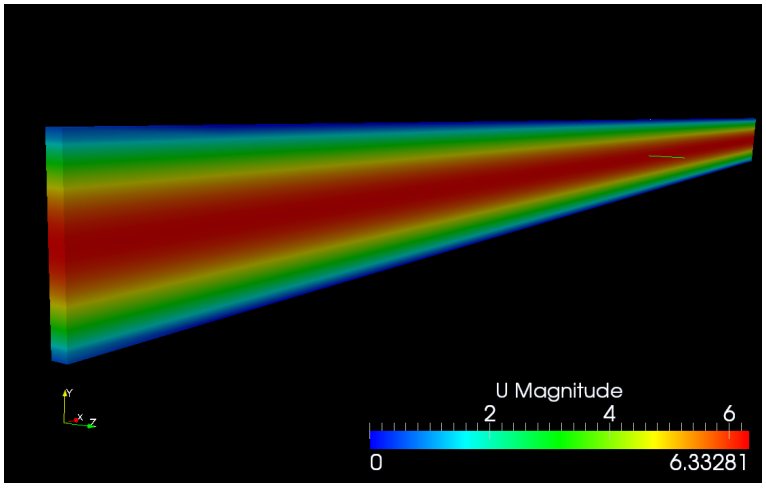


Figure 2.13: Velocity field obtained with icoFoam in the *plane-Poiseuille flow* case (m/s)

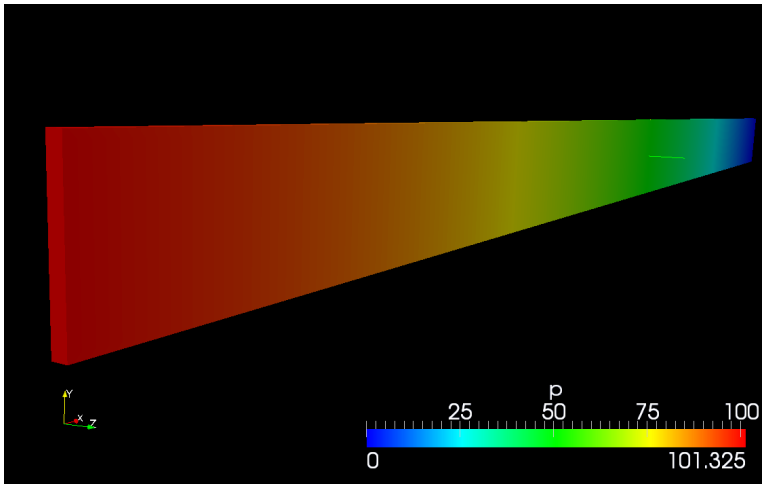


Figure 2.14: Pressure field obtained with icoFoam in the *plane-Poiseuille flow* case (m^2/s^2)

The user should have noted that the numerical results seem to fit the analytical equations; the pressure shows a linear trend and the velocity profile looks like parabolic. As done previously, the user can plot these variables to compare them with the analytical results. They are shown in Figure 2.15 and Figure 2.16.

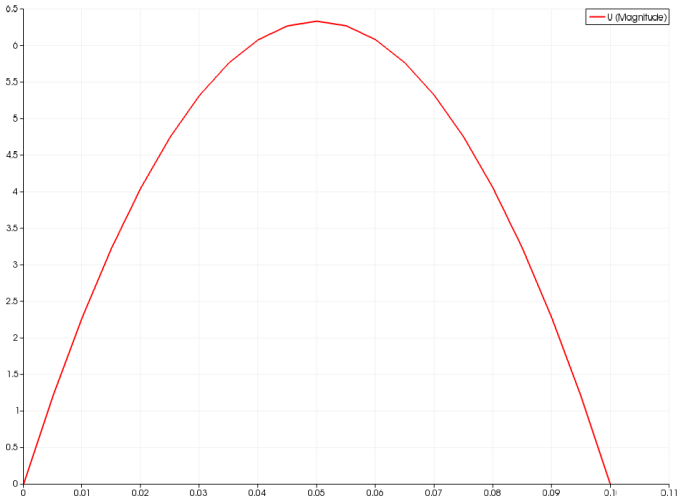


Figure 2.15: Distribution of $|\mathbf{U}|$ (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *plane-Poiseuille flow* case

Figure 2.16: Distribution of p (ordinate axis) along the x-axis (abscissa axis) obtained with the icoFoam simulation of the *plane-Poiseuille flow* case

As it can be observed, the velocity profile is parabolic with a symmetrical distribution respect to the plates. Furthermore, according to Equation 4.5, the maximum analytical speed is $u_{max} = 6.33281$ m/s, which perfectly coincides with the maximum value achieved in the *OpenFOAM*® simulation (Figure 2.13).

Regarding the pressure, as it was explained in 2.4.1, it follows a linear trend, starting at $p = 101325$ Pa and finishing at $p = 0$ Pa (Figure 2.14). Moreover, its derivative $(\frac{dp}{dx})$ is negative. As it can be seen, the analytical and numerical results coincide.

2.5 Third case: plane-parallel plates with relative movement and pressure gradient (Couette flow with pressure gradient)

2.5.1 Physics of the problem

In the third case of Chapter 2, the movement of the flow is caused by the combined effect of the relative displacement of the plates ($V \neq 0$) and the existence of a pressure gradient in the x-direction ($\frac{\partial p}{\partial x} \neq 0$).

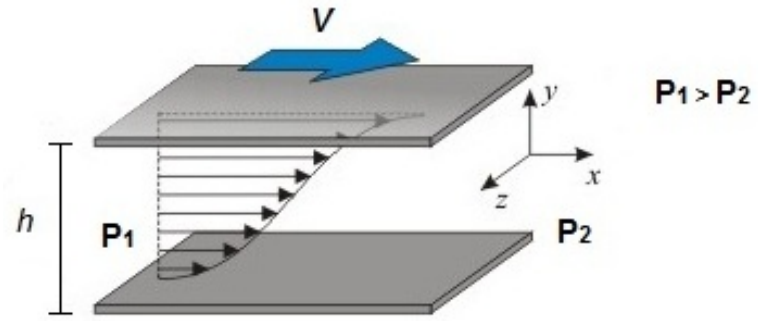


Figure 2.17: Viscous incompressible flow between two plane-parallel plates with relative movement and with a pressure gradient in the x-direction

The mathematical treatment of the case is the same as done for the *plane-Poiseuille flow* except for the fact that the boundary conditions change: with $u = V$ in $y = \frac{h}{2}$ and $u = 0$ in $y = -\frac{h}{2}$ applied in Equation 2.7, the analytical solution takes the form

$$u = \frac{1}{2\mu}\frac{dp}{dx}y^2 + \frac{V}{h}y - \frac{1}{2\mu}\frac{dp}{dx}\left(\frac{h^2}{4}\right) + \frac{V}{2}, \quad -\frac{h}{2} \leq y \leq \frac{h}{2} \tag{2.10}$$

This is the general solution of the *Couette flow* with pressure gradient: a non-symmetric parabolic arc velocity profile.

For the resolution with *OpenFOAM*®, h is given a value of 0.1 m and the plates length (l) is set to 2 m. The mobile wall has an horizontal velocity of $V = 10$ m/s and the pressure gradient will be caused by applying vacuum at outlet while mantaining ambient pressure at inlet ($P_1 = 101325$ Pa and $P_2 = 0$ Pa). The velocity of the top wall is 10 m/s instead of 1 m/s (as it happened in the *Couette flow* case)

to better appreciate the combined effect of the two kinds of boundary conditions.

2.5.2 Pre-processing

As it happens with new cases of study, the *Couette flow with pressure gradient* case is going to be contained in FoamCases, and is going to be named ppWallGrad. It will be set by modifying the ppGrad case; more specifically, its *0/U* file.

2.5.2.1 Boundary and initial conditions

It is only necessary to change the velocity of top patch from 0 m/s to 10 m/s by typing the instruction

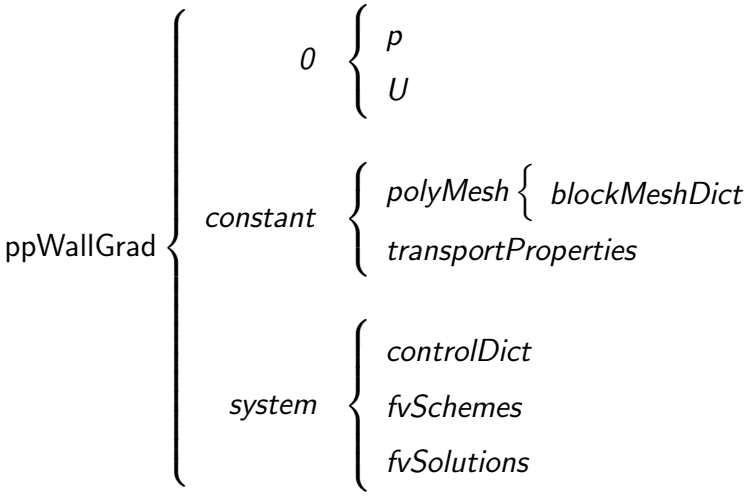
```
uniform (10 0 0)
```

in line 26 of *U* file.

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volVectorField;
13     object      U;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 1 -1 0 0 0 0];
18
19 internalField   uniform (0 0 0);
20
21 boundaryField
22 {
23     top
24     {
25         type            fixedValue;
26         value           uniform (10 0 0);
27     }
28
29     bottom
30     {
31         type            fixedValue;
32         value           uniform (0 0 0);
33     }
```

```
34
35      inlet
36      {
37          type              zeroGradient;
38      }
39
40      outlet
41      {
42          type              zeroGradient;
43      }
44
45      frontAndBack
46      {
47        type                empty;
48      }
49  }
50
51  // ************************************************************************* //
```

The case is ready to be run. As the changes have only been applied to instructions within files, the scheme of directories and subdirectories is maintained.



2.5.3 Post-processing

2.5.3.1 Results of the simulation

Once icoFoam has ben executed and if all the files have been properly set, the results for $|\mathbf{U}|$ and p should be the following.

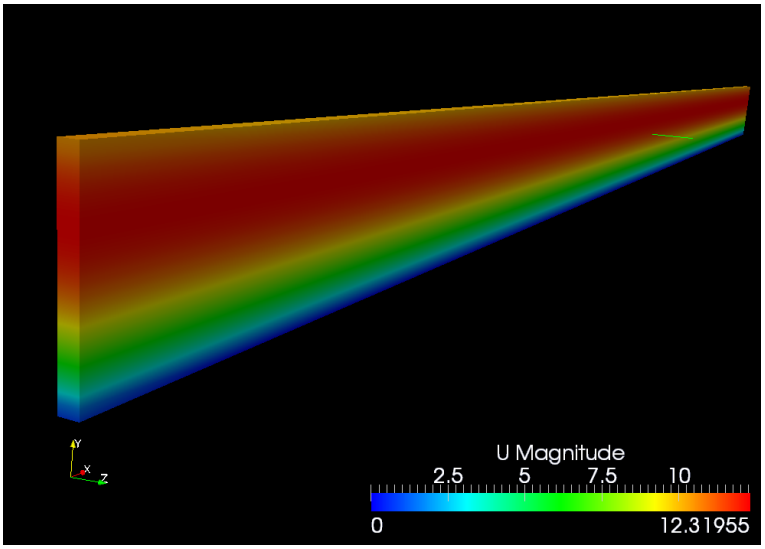


Figure 2.18: Velocity field obtained with icoFoam in the *Couette flow with pressure gradient* case (m/s)

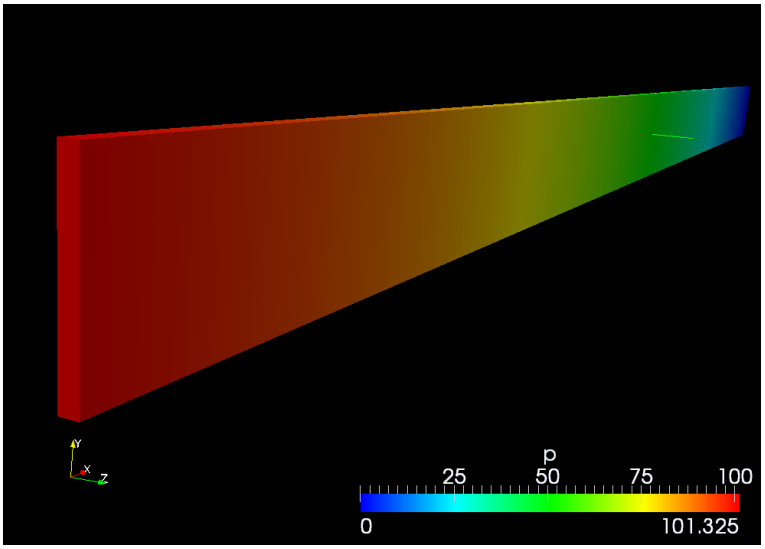


Figure 2.19: Pressure field obtained with icoFoam in the *Couette flow with pressure gradient* case (m^2/s^2)

The user should have noted that the numerical results seem to fit the analytical equations; the pressure shows a linear trend and the velocity profile looks like a non-symmetric parabola arc. As done previously, the user can plot these variables to compare them with the analytical results. They are shown in Figure 2.20 and Figure 2.21.

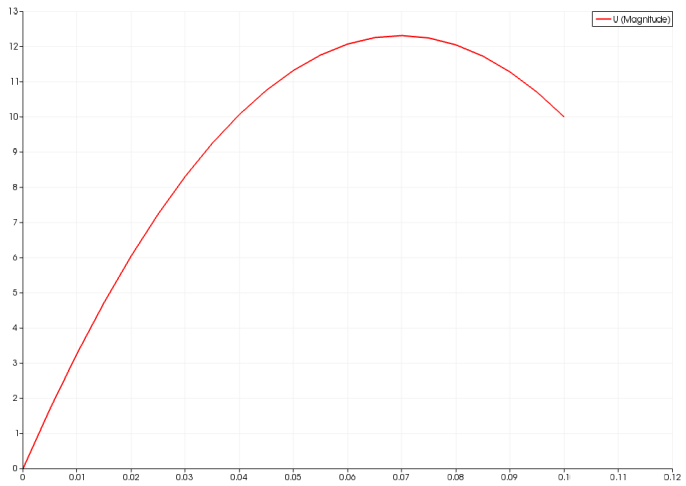


Figure 2.20: Distribution of $|\mathbf{U}|$ (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow with pressure gradient* case

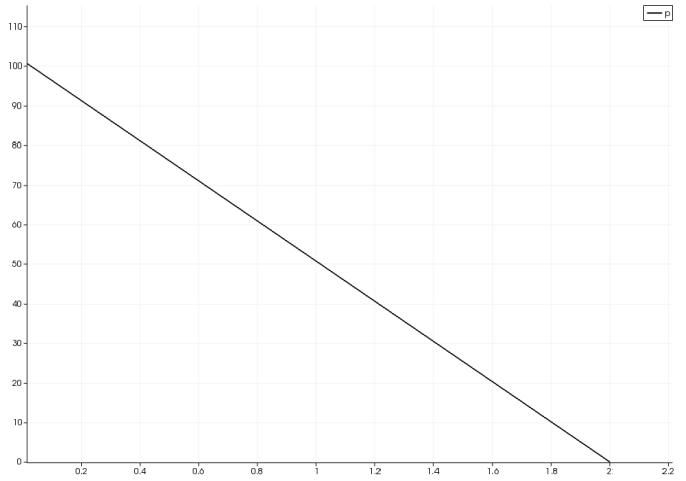


Figure 2.21: Distribution of p (ordinate axis) along the x-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow with pressure gradient* case

The behaviour of the pressure is exactly the same that appears in Figure 2.16 as there are no differences between ppGrad and ppWallGrad in this aspect. The only difference appears in the parabola which represents the velocity profile. The maximum velocity has been incremented and it is not symmetrical respect to the planes; its maximum has been displaced toward the top wall, as it can be deduced from the analytical solution (Equation 2.10).

2.6 Additional utilities

2.6.1 Vector plots

It is possible to plot vectorial fields in ParaView, allowing a better understanding and interpretation of the results. Now, besides $|\mathbf{U}|$, the user will be able to observe the instantaneous velocity vector of the fluid particles by obtaining its velocity field.

It is going to be done with the results of the *plane-Poiseuille flow* case (second case in Chapter 2). First of all, once in ParaView, the user has to generate a vector glyph for velocity at the center of each cell: it is necessary to filter the data to cell centers. To do that, with the `ppGrad.OpenFOAM` module highlighted in the Pipeline Browser (and in $t = 2.5$ s), go to Filters -> Alphabetical -> Cell centers. A new module has appeared, and by highlightning it, go to Filters -> Alphabetical -> Glyph, clicking on Apply both times.

To improve the viewing of the vector, click on `Edit` in the Set Scale Factor in Properties, and set it to 0.005. In Scale Mode (also in Properties), the user can select whether to represent all the vectors with the same lentgh (`off`) or proportional to $|\mathbf{U}|$ (`vector`).

The user is free to change the colour of the vectors by selecting `Solid Color` in Color by wihtin the Display planel (with `Glyph` highlighted in the Pipeline Browser). Moreover, if the user wants to superpose the vectorial representation on the $|\mathbf{U}|$ colour map, with `ppGrad.OpenFOAM` highlighted, select Display -> Style -> Representation -> Points and afterwards, Display -> Backface Style -> Representation -> Surface. The results are shown in Figure 2.22.

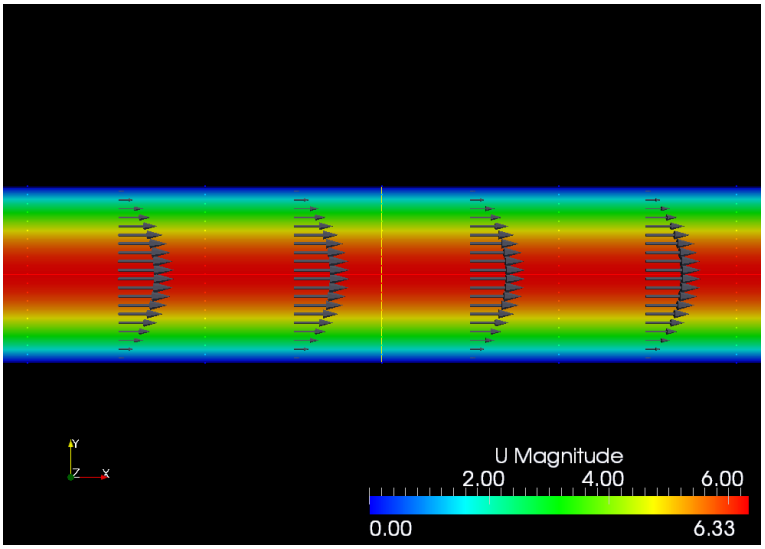


Figure 2.22: Vectors of the flow velocity in the *plane-Poiseuille flow* case (m/s)

2.6.2 Streamlines

As with the vector plots, the streamlines offer a great visual representation of the behaviour of the flow. The streamlines are the family of curves that are instantaneously tangent to the velocity vector of the fluid.

To obtain the streamlines in ParaView, go to Filters -> Alphabetical -> StreamTracer.

Caution:

> Before doing it and if required, erase all the modules that appear in the Pipeline Browser except `ppGrad.OpenFOAM`

In order to improve the viewing of the streamlines, set the Point option within Properties to 0.7 0.05 0.005, Number of points to 5000 and Radius to 0.16. The results can be seen at Figure 2.23.

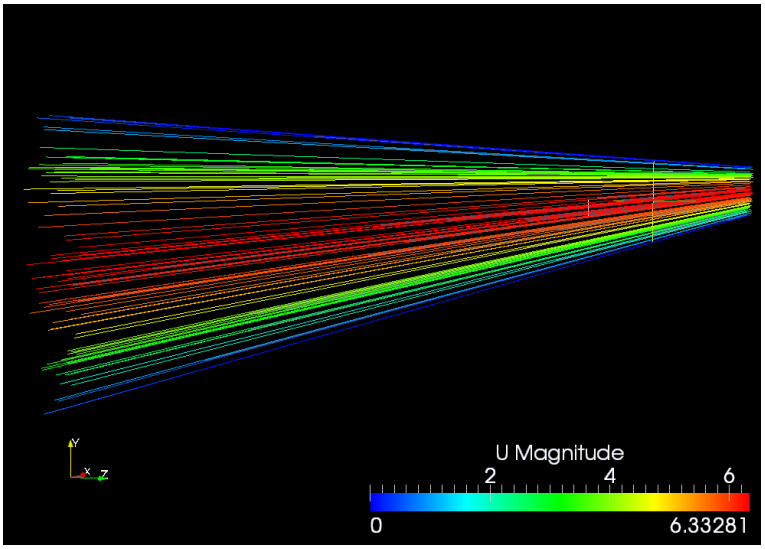


Figure 2.23: Streamlines in the *plane-Poiseuille flow* case (m/s)

Advice:

> The user can choose whether to represent $|\mathbf{U}|$ or other flow properties such as p, vorticity, etc. It can be selected in the first drop-down menu next to the vertical rainbow icon at the top-left corner of ParaView's window.

2.6.3 Computation of the volumetric flow rate

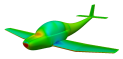
The computation of the volumetric flow rate is going to be obtained by using *OpenFOAM*® utilities. It allows the processing, manipulation and computation of determined parameters to complete or modify the cases. The user can access the list of utilities by typing

```
foam ls
```

and going to applications/utilities. There, it is possible to observe the huge amount of utilties to perform modifications to the case of study.

Advice:

> While typing in the terminal, the words are automatically written from the first letter by pressing *tab*



The instruction for the computation of the volumetric flow rate is contained in postProcessing/patch/patchIntegrate. patchIntegrate calculates the integral of the specified field over the specified patch. Therefore, to compute the volumetric flow rate in the inlet patch of the ppGrad case, type:

```
patchIntegrate phi inlet > inletFlow
```

and a file named *inletFlow* is going to be created within the case, containing the volumetric flow rate at inlet (or any other patch specified according to the necessities of the user). The user can check *inletFlow* to see the results obtained every time step. To understand how to manage *phi* (depending whether the flow is compressible or incompressible) and to figure out its physical meaning, Figure 2.24 can be consulted.

Filename	Description	Dimension
U	Velocity	$\frac{m}{s}$
U_0	Velocity at previous timestep (needed for restarting higher order time-stepping schemes)	$\frac{m}{s}$
phi	Flow through cell faces. Depends on whether the solver is for incompressible or compressible flow.	incompressible flow: $\frac{m^3}{s}$ compressible flow: $\frac{kg}{s}$
p	Pressure	$\frac{m^2}{s^2}$
epsilon	Turbulence dissipation rate	$\frac{m^2}{s^3}$
k	Turbulence kinetic energy	$\frac{m^2}{s^2}$
rho	Gas density	$\frac{kg}{m^3}$
alpha	Dispersed phase volume fraction	(Dimensionless, usually within the interval [0, 1])
Theta	Granular temperature	$\frac{m^2}{s^2}$

Figure 2.24: Table containing the definition of *phi* depending whether the case is compressible or incompressible [1]

An interesting issue to be done is to observe the convergence of the simulation, according to the values of the volumetric flow rate at every time step.

Example: Taking a look into inletFlow, it is possible to observe that gradually the volumetric flow rate tends to stabilize to $Q = -0.00424298\ m^3/s$. *It is negative because the normal vector* $\vec{n}$ *(which is also specified in inletFlow) and* $\vec{v}$ *at inlet are antiparallel*

As done previously, the simple mathematical solution of the *plane-Poiseuille flow* allows the user to check if the results obtained with *OpenFOAM*® coincide with the analytical treatment. Mathematically, the volumetric flow rate is computed as

$$Q = \int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{0}^{0.01} \vec{v} \cdot d\vec{S}$$

and by using Equation 2.8, the volumetric flow rate equals

$$Q = \frac{0.01 \cdot h^3}{12\mu} \left(-\frac{dp}{dx} \right) = -0.0042219 \; m^3/s \tag{2.11}$$

The user should have noted that although this value is very close to the one obtained applying patchIntegrate, it slightly varies (it represents an error of 0.499%). One solution to obtain even more accurate results might be to set a more refined mesh.

2.6.3.1 Refinement of the mesh

Taking advantage of the previous results, in the current section the mesh of the ppGrad case is going to be refined. As $u = u(y)$ (Equation 2.1), the refinement has to be applied in the y-direction. The user can create a new case named ppGradFineMesh containing the same directories, subdirectories and files as ppGrad but changing its *constant/polyMesh/blockMeshDict* and *system/controlDict* files.

The first step is to change the mesh. Up to now, the main block was divided into cells according to the instruction:

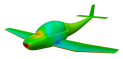
```
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
```

as explained in Section 2.3.2.1. Now, instead of dividing by 20, it will be set that the block be divided into 80, 120 or 160 cells in the y-direction. Then, the volumetric flow rates obtained with each option will be compared.

Caution:

> The relative error achieved while using 20 divisions is of course acceptable for the majority of the engineering problems. However, the present section shows how to obtain even more accurate results if required, as well as showing how to refine a mesh created with blockMesh

The second step is to change the time control settings. As the length of the cells has been changed, the Courant number might be higher than 1. Consequently, deltaT has to be changed to 0.00005 and writeInterval to 2000 (it could be applied a lower value when running with 80 divisions, but these values are set for the most crititcal case, 160 divisions).

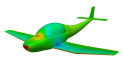


The changes in the files are shown in the following instructions.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
   (0 0 0)
   (20 0 0)
   (20 1 0)
   (0 1 0)
   (0 0 0.1)
   (20 0 0.1)
   (20 1 0.1)
   (0 1 0.1)
);

blocks
(
   hex (0 1 2 3 4 5 6 7) (20 160 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
   top
   {
       type wall;
       faces
       (
            (3 7 6 2)
       );
   }
   bottom
   {
       type wall;
       faces
       (
           (1 5 4 0)
       );
```

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

mergePatchPairs
(
);

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

```
/*--------------------------------* C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application         icoFoam;

startFrom           startTime;

startTime           0;
```



```
23
24  stopAt              endTime;
25
26  endTime             2.5;
27
28  deltaT              0.00005;
29
30  writeControl        timeStep;
31
32  writeInterval       2000;
33
34  purgeWrite          0;
35
36  writeFormat         ascii;
37
38  writePrecision      6;
39
40  writeCompression    off;
41
42  timeFormat          general;
43
44  timePrecision       6;
45
46  runTimeModifiable   true;
47
48  // ************************************************************************* //
```

Caution:

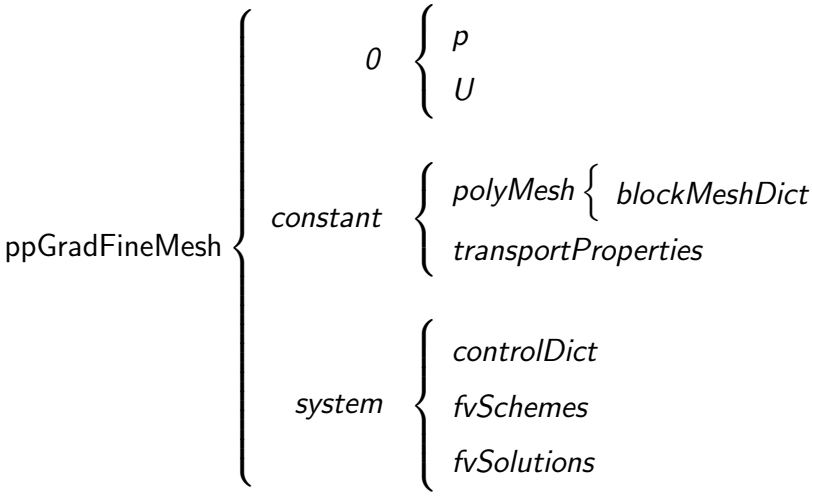
Do not forget to run blockMesh again in order to create the new mesh

The results of the three simulations referred to the volumetric flow rate and the relative error are the following.

$$\text{Refinement} \begin{cases} \textit{80 divisions} \Rightarrow Q = -0.00422319\ m^3/s \Rightarrow e_r = 0.031\% \\ \textit{120 divisions} \Rightarrow Q = -0.00422246\ m^3/s \Rightarrow e_r = 0.013\% \\ \textit{160 divisions} \Rightarrow Q = -0.00422222\ m^3/s \Rightarrow e_r = 0.007\% \end{cases}$$

As it can be seen, the more accurate the mesh, the lower the error.

As the changes have only been applied to instructions within files, the scheme of directories and subdirectories in ppGradFineMesh is maintained.

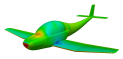


2.6.4 Computation of the wall shear stress

The shear stress is going to be computed by using *OpenFOAM*® utilities, as it was done with the volumetric flow rate. The current utility can be found in (foam ls) applications/utilities/postProcessing/wall/wallShearStress. More specifically, the shear stress at the walls (top and bottom) of the *plane-Poiseuille flow* case is going to be computed.

Before using the utility, the user has to create a new file within ppGrad/constant named *RASProperties*, containing the following information:

```
1  /*---------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "constant";
14     object      RASProperties;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 RASModel        laminar;
19
20 turbulence      off;
21
22 printCoeffs     off;
23
24
```



```
25  // ************************************************************************* //
```

Afterwards, the user has to add a new instruction into *transportProperties*,

`transportModel Newtonian`

above the definition of ν, as it is shown in the following code:

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "constant";
14      object      transportProperties;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  transportModel Newtonian;
19
20  nu              nu [0 2 -1 0 0 0 0] 0.01;
21
22  // ************************************************************************* //
```

Now it is possible to start the computation of the shear stress. Type

`wallShearStress`

within the case, and inside the temporal subdirectories of the case, a new file has appeared, containing the information referred to the shear stress at the walls of the domain. The user can check it in ParaView. There, the user has to select the box of `wallShearStress`, contained in the Volume Fields menu (next to `U` and `p`), located in Properties. The results are shown in Figure 2.12.

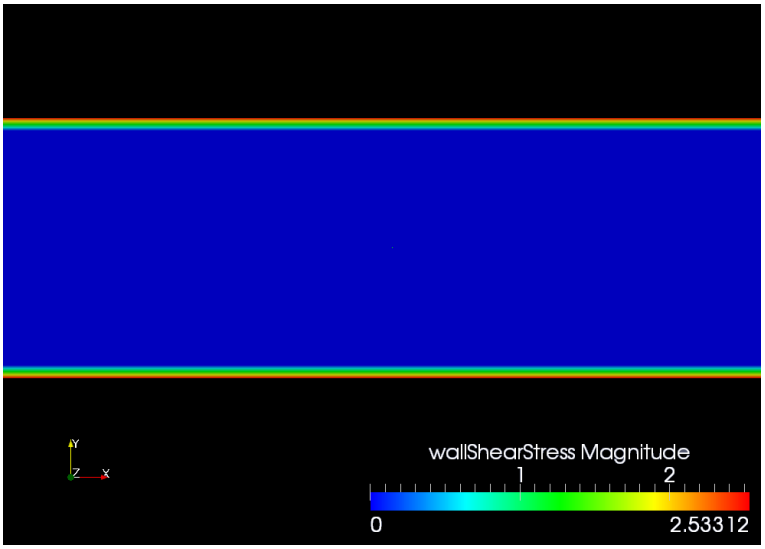


Figure 2.25: Shear stress at the walls of the *plane-Poiseuille flow* case (m^2/s^2)

Mathematically, the shear stress at the bottom wall is (by definition of Newtonian fluid)

$$\tau_w = \tau_{xy\ wall} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \Bigg|_{y=-\frac{h}{2}} \approx \mu \left(\frac{\partial u}{\partial y} \right) \Bigg|_{y=-\frac{h}{2}} \tag{2.12}$$

Combining Equation 2.12 with Equation 2.8, it is obtained that

$$\tau_w = \frac{dp}{dx} \frac{h}{2} = 2533.12\ Pa$$

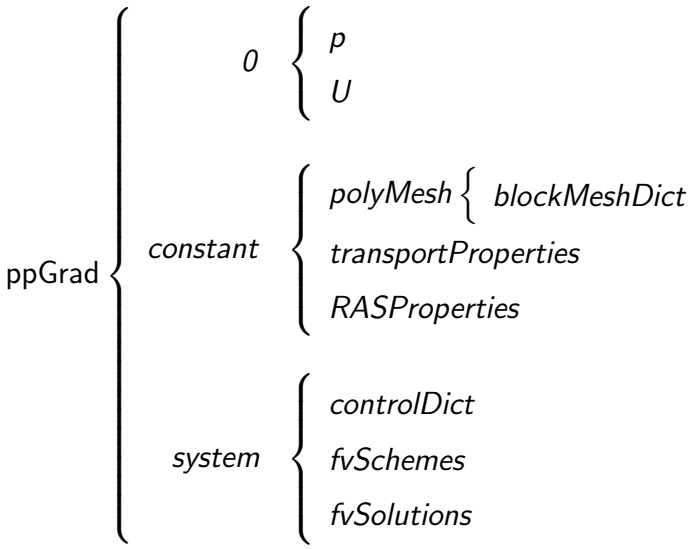
At the top wall, the shear stress is equal but negative, according to the sign convention.

Caution:

> As it happens with pressure, the shear stress provided by $OpenFOAM^{®}$ is divided by the density $\left(\frac{\tau_w}{\rho} \right)$, and therefore it is not expressed in Pa

As it was explained in 2.4.2.1, the value given by $OpenFOAM^{®}$ has to be multiplied by the density. Thus, as in the current case $\rho = 1000$ kg/s, $\tau_w = 2533.12$ Pa, which perfectly matches with the analytical solution.

To compute the shear stress, a new file needs to be added. Thus, the scheme of directories and subdirectories in ppGrad has changed.



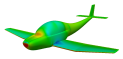
2.6.4.1 Creation of a graded mesh

As with the computation of the volumetric flow rate, the user is invited to create a finer mesh in order to obtain even more accurate results of the shear stress. As it is computed by means of the derivative of the velocity evaluated in the walls, the finer the mesh in the patches, better the results. However, it is not necessary to remesh the whole domain, but only to refine those parts of the mesh next to the walls. To do that, insted of dividing the main block into more cells, the mesh can be graded by changing the cell expansion ratios.

To change the case accordingly, it is necessary to modify the *blockMeshDict* as follows:

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      blockMeshDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 convertToMeters 0.1;
18
19 vertices
20 (
```

```
21      (0 0 0)
22      (20 0 0)
23      (0 1 0)
24      (20 1 0)
25      (0 0 0.1)
26      (20 0 0.1)
27      (0 1 0.1)
28      (20 1 0.1)
29      (0 -1 0)
30      (20 -1 0)
31      (0 -1 0.1)
32      (20 -1 0.1)
33  );
34
35  blocks
36  (
37      hex (0 1 3 2 4 5 7 6) (20 20 1) simpleGrading (1 0.1 1)
38      hex (8 9 1 0 10 11 5 4) (20 20 1) simpleGrading (1 10 1)
39  );
40
41  edges
42  (
43  );
44
45  boundary
46  (
47      movingWall
48      {
49          type wall;
50          faces
51          (
52               (2 6 7 3)
53          );
54      }
55      fixedWalls
56      {
57          type wall;
58          faces
59          (
60              (8 9 11 10)
61          );
62      }
63      inlet
64      {
65          type patch;
66          faces
67          (
68              (10 4 0 8)
69              (4 6 2 0)
70          );
71      }
72      outlet
73      {
74          type patch;
75          faces
76          (
77              (1 3 7 5)
```



```
78              (9 1 5 11)
79          );
80      }
81      frontAndBack
82      {
83          type empty;
84          faces
85          (
86              (11 5 4 10)
87              (5 7 6 4)
88              (8 0 1 9)
89              (0 2 3 1)
90          );
91      }
92  );
93
94  mergePatchPairs
95  (
96  );
97
98  // ************************************************************************* //
```

The user should note that doing it, two blocks are created. The key instructions are lines 37 and 38; they specify the information related to the cell expansion ratios according to the definition given in 2.3.2.1. Also note that for instance, the inlet patch is now containing two faces. By using this *blockMeshDict* to run blockMesh, the mesh takes the form:

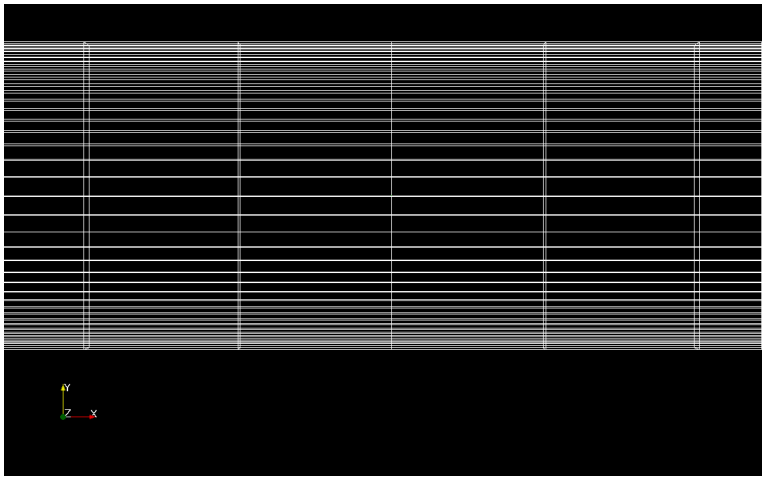


Figure 2.26: Graded mesh used to obtain more accurate values of the wall shear stress in the *plane-Poiseuille flow* case

3. Bidimensional laminar flow around a circular cylinder

3.1 Description of the case

The second tutorial is based on the analysis of external flow. More specifically, the study encompasses the modelling of a bidimensional laminar flow around a circular cylinder. The analytical solution of the same case modeled with potential flow is known and easy to interpret; however the physical mechanism which rules *real* flow around a circular cylinder as well as its mathematical treatment are highly complex. Some examples are the boundary layer detachment and oscillatory effects that without a CFD study would be very difficult to determine either analytically or experimentally. The case includes simulations made with two different Reynolds numbers to observe the behaviour of the flow in such different regimes.

3.2 Hypotheses

- Incompressible flow
- Laminar flow
- Newtonian flow
- Bidimensional flow ($\frac{\partial}{\partial z} = 0$)
- Negligible gravitatory effects

3.3 Physics of the problem

The problem encompasses a circular cylinder with a diameter of $D = 0.1$ m submerged in a stream with an undisturbed speed of $V = 5$ m/s and ambient pressure. The problem statement is shown at Figure 3.1.

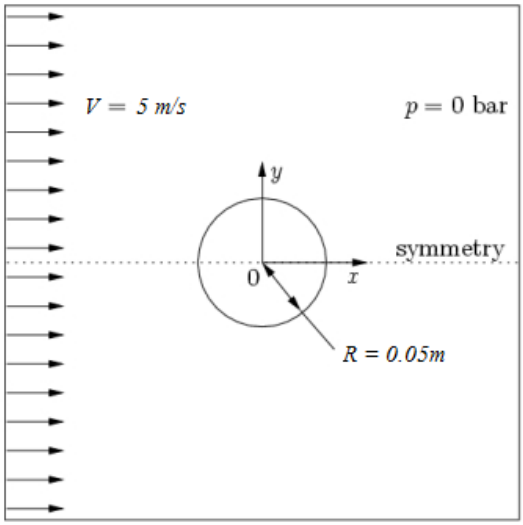


Figure 3.1: Flow around a circular cylinder

Unlike Chapter 2, in the current chapter there is no easy analytical solution to describe the behaviour of the fluid (except for potential flow, which is an ideal case). However, it is necessary to keep in mind the main equations and dimensionless numbers involved in the problem:

The continuity equation,

$$\nabla \cdot \mathbf{U} = 0 \tag{3.1}$$

The momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{U} \tag{3.2}$$

The Reynolds number,

$$Re = \frac{D|\mathbf{U}|}{\nu} \tag{3.3}$$

The Strouhal number (which describes oscillating flow mechanisms),

$$St = \frac{wD}{2\pi|\mathbf{U}|} \tag{3.4}$$

The drag coefficient (dimensionless force parallel to the flow),

$$C_D = \frac{F_D}{\frac{1}{2}\rho|\mathbf{U}|^2 S} \tag{3.5}$$

The behaviour of the flow around a circular cylinder is very complex and highly de-

pendent on the Reynolds number. The boundary layer in laminar regimes detaches at an angle of $\theta = 82^{o}$ as a consequence of the existence of an adverse pressure gradient. Furthermore, due to the symmetry of the cylinder, a curious phenomenon appears named *von Kármán street*, consisting of alternate vortices emitted by the cylinder. The vortices are detached periodically and their frequence is directly related to the Strouhal number.

For external flow, the transition from laminar regime to turbulent regime occurs at $Re \approx 3 \times 10^{5}$. However, the very complex structure of vortex shedding deserves special mention; the relation between the behaviour of the flow and the Reynolds number is shown at Figure 3.2:

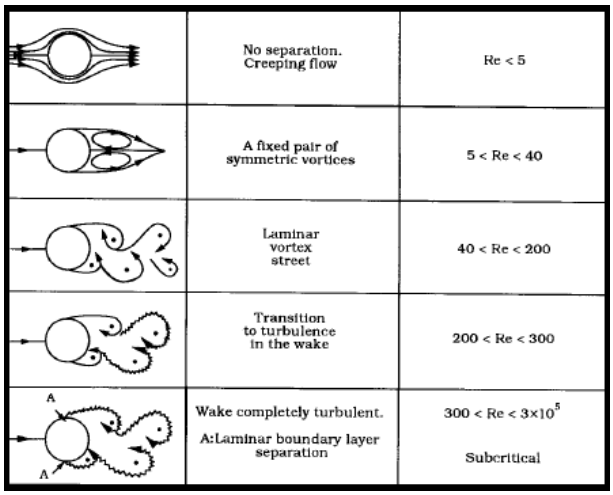


Figure 3.2: Flow structure depending on the Reynolds number, extracted from [2]

As the simulation is going to be run with laminar flow, the Reynolds number is not going to be set higher than 195. This will allow to observe the von Kármán street and its transition while guaranteeing that the whole fluid domain remains laminar. Additionally, another simulation with $Re = 30$ will be run to check if the results of the CFD simulation adapt to the scheme shown at Figure 3.2.

The drag coefficient is a function of the Reynolds number and decreases when the regime turns to turbulent. It can be seen at Figure 3.3.

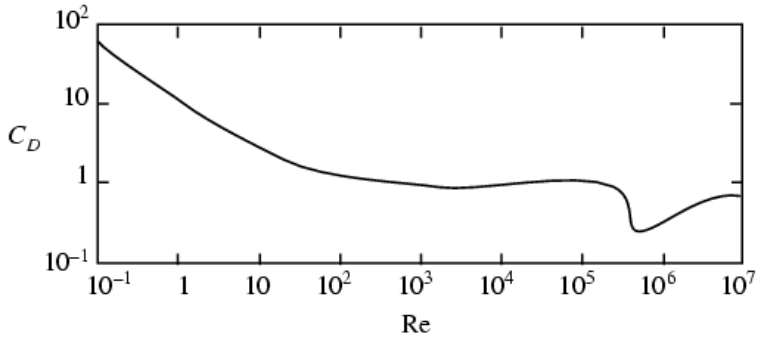


Figure 3.3: Drag coefficient as a function of the Reynolds number in an infinite circular cylinder

3.4 Pre-processing with $Re = 195$

The following codes contain the information to simulate the cylinder with $Re = 195$ using icoFoam. The case directory is named cylinder195 and will be located within FoamCases. Its structure of directories and subdirectories is very similar to the one used in Chapter 2.

3.4.1 Mesh generation

The mesh for the study of the cylinder is not going to be uniform. Some areas of the domain need to contain a higher cell density (mainly the walls of the cylinder and its prolongation). Consequently, it is necessary to divide the domain in different blocks as Figure 3.4 shows.

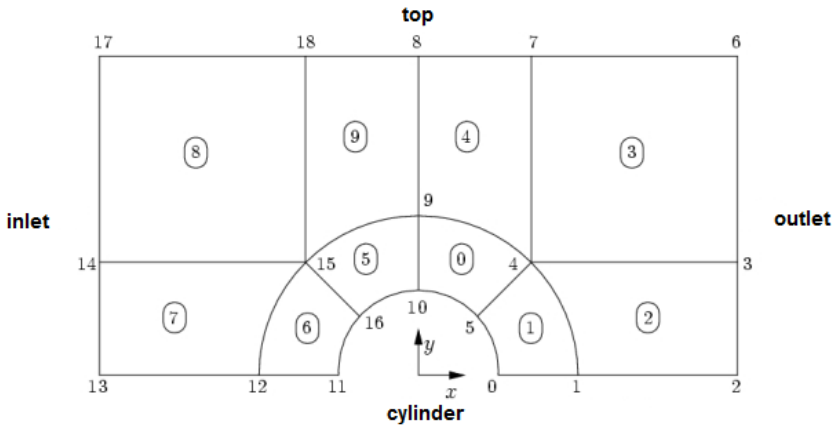
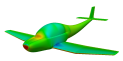


Figure 3.4: Half of the scheme used for the creation of the mesh, extracted from [1]

In Figure 3.4 there is only half of the mesh, which is contained between $z = -0.05$ m and $z = 0.05$ m. The numbers indicate the numbering used for the vertices and the blocks in the *blockMeshDict* file.

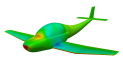
In the *bidimensional cylinder* case, *constant/blockMeshDict* must contain the following instructions:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0.5 0 -0.5)
    (1 0 -0.5)
    (10 0 -0.5)
    (10 0.707107 -0.5)
    (0.707107 0.707107 -0.5)
    (0.353553 0.353553 -0.5)
    (10 2 -0.5)
    (0.707107 2 -0.5)
    (0 2 -0.5)
```



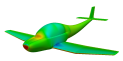
```
30      (0 1 -0.5)
31      (0 0.5 -0.5)
32
33      (-0.5 0 -0.5)
34      (-1 0 -0.5)
35      (-2 0 -0.5)
36      (-2 0.707107 -0.5)
37      (-0.707107 0.707107 -0.5)
38      (-0.353553 0.353553 -0.5)
39      (-2 2 -0.5)
40      (-0.707107 2 -0.5)
41
42      (0.5 0 0.5)
43      (1 0 0.5)
44      (10 0 0.5)
45      (10 0.707107 0.5)
46      (0.707107 0.707107 0.5)
47      (0.353553 0.353553 0.5)
48      (10 2 0.5)
49      (0.707107 2 0.5)
50      (0 2 0.5)
51      (0 1 0.5)
52      (0 0.5 0.5)
53
54      (-0.5 0 0.5)
55      (-1 0 0.5)
56      (-2 0 0.5)
57      (-2 0.707107 0.5)
58      (-0.707107 0.707107 0.5)
59      (-0.353553 0.353553 0.5)
60      (-2 2 0.5)
61      (-0.707107 2 0.5)
62
63
64      (10 -0.707107 -0.5)
65      (0.707107 -0.707107 -0.5)
66      (0.353553 -0.353553 -0.5)
67      (10 -2 -0.5)
68      (0.707107 -2 -0.5)
69      (0 -2 -0.5)
70      (0 -1 -0.5)
71      (0 -0.5 -0.5)
72
73      (-2 -0.707107 -0.5)
74      (-0.707107 -0.707107 -0.5)
75      (-0.353553 -0.353553 -0.5)
76      (-2 -2 -0.5)
77      (-0.707107 -2 -0.5)
78
79      (10 -0.707107 0.5)
80      (0.707107 -0.707107 0.5)
81      (0.353553 -0.353553 0.5)
82      (10 -2 0.5)
83      (0.707107 -2 0.5)
84      (0 -2 0.5)
85      (0 -1 0.5)
86      (0 -0.5 0.5)
```

```
87
88          (-2 -0.707107 0.5)
89          (-0.707107 -0.707107 0.5)
90          (-0.353553 -0.353553 0.5)
91          (-2 -2 0.5)
92          (-0.707107 -2 0.5)
93      );
94
95      blocks
96      (
97          hex (5 4 9 10 24 23 28 29) (80 20 1) simpleGrading (10 1 1)
98          hex (0 1 4 5 19 20 23 24) (80 20 1) simpleGrading (10 1 1)
99          hex (1 2 3 4 20 21 22 23) (200 20 1) simpleGrading (1 1 1)
100         hex (4 3 6 7 23 22 25 26) (200 40 1) simpleGrading (1 1 1)
101         hex (9 4 7 8 28 23 26 27) (20 40 1) simpleGrading (1 1 1)
102         hex (16 10 9 15 35 29 28 34) (20 80 1) simpleGrading (1 10 1)
103         hex (11 16 15 12 30 35 34 31) (20 80 1) simpleGrading (1 10 1)
104         hex (12 15 14 13 31 34 33 32) (20 20 1) simpleGrading (1 1 1)
105         hex (15 18 17 14 34 37 36 33) (40 20 1) simpleGrading (1 1 1)
106         hex (9 8 18 15 28 27 37 34) (40 20 1) simpleGrading (1 1 1)
107
108
109         hex (40 45 44 39 53 58 57 52) (20 80 1) simpleGrading (1 10 1)
110         hex (0 40 39 1 19 53 52 20) (20 80 1) simpleGrading (1 10 1)
111         hex (1 39 38 2 20 52 51 21) (20 200 1) simpleGrading (1 1 1)
112         hex (39 42 41 38 52 55 54 51) (40 200 1) simpleGrading (1 1 1)
113         hex (44 43 42 39 57 56 55 52) (40 20 1) simpleGrading (1 1 1)
114         hex (48  47 44 45 61 60 57 58) (80 20 1) simpleGrading (10 1 1)
115         hex (11 12 47 48 30 31 60 61) (80 20 1) simpleGrading (10 1 1)
116         hex (12 13 46 47 31 32 59 60) (20 20 1) simpleGrading (1 1 1)
117         hex (47 46 49 50 60 59 62 63) (20 40 1) simpleGrading (1 1 1)
118         hex (44 47 50 43 57 60 63 56) (20 40 1) simpleGrading (1 1 1)
119     );
120
121     edges
122     (
123         arc 0 5 (0.469846 0.17101 -0.5)
124         arc 5 10 (0.17101 0.469846 -0.5)
125         arc 1 4 (0.939693 0.34202 -0.5)
126         arc 4 9 (0.34202 0.939693 -0.5)
127         arc 19 24 (0.469846 0.17101 0.5)
128         arc 24 29 (0.17101 0.469846 0.5)
129         arc 20 23 (0.939693 0.34202 0.5)
130         arc 23 28 (0.34202 0.939693 0.5)
131         arc 11 16 (-0.469846 0.17101 -0.5)
132         arc 16 10 (-0.17101 0.469846 -0.5)
133         arc 12 15 (-0.939693 0.34202 -0.5)
134         arc 15 9 (-0.34202 0.939693 -0.5)
135         arc 30 35 (-0.469846 0.17101 0.5)
136         arc 35 29 (-0.17101 0.469846 0.5)
137         arc 31 34 (-0.939693 0.34202 0.5)
138         arc 34 28 (-0.34202 0.939693 0.5)
139
140
141         arc 0 40 (0.469846 -0.17101 -0.5)
142         arc 40 45 (0.17101 -0.469846 -0.5)
143         arc 1 39 (0.939693 -0.34202 -0.5)
```



```
144     arc 39 44 (0.34202 -0.939693 -0.5)
145     arc 19 53 (0.469846 -0.17101 0.5)
146     arc 53 58 (0.17101 -0.469846 0.5)
147     arc 20 52 (0.939693 -0.34202 0.5)
148     arc 52 57 (0.34202 -0.939693 0.5)
149     arc 11 48 (-0.469846 -0.17101 -0.5)
150     arc 48 45 (-0.17101 -0.469846 -0.5)
151     arc 12 47 (-0.939693 -0.34202 -0.5)
152     arc 47 44 (-0.34202 -0.939693 -0.5)
153     arc 30 61 (-0.469846 -0.17101 0.5)
154     arc 61 58 (-0.17101 -0.469846 0.5)
155     arc 31 60 (-0.939693 -0.34202 0.5)
156     arc 60 57 (-0.34202 -0.939693 0.5)
157 );
158
159 boundary
160 (
161
162     top
163     {
164         type symmetryPlane;
165         faces
166         (
167             (7 8 27 26)
168             (6 7 26 25)
169             (8 18 37 27)
170             (18 17 36 37)
171         );
172     }
173
174     bottom
175     {
176         type symmetryPlane;
177         faces
178         (
179            (49 50 63 62)
180            (50 43 56 63)
181            (43 42 55 56)
182            (42 41 54 55)
183         );
184     }
185
186     inlet
187     {
188         type patch;
189         faces
190         (
191             (14 13 32 33)
192             (17 14 33 36)
193
194             (46 13 32 59)
195             (46 49 62 59)
196         );
197     }
198
199     outlet
200     {
```

```
201         type patch;
202         faces
203         (
204             (2 3 22 21)
205             (3 6 25 22)
206 
207             (38 51 21 2)
208             (41 54 51 38)
209         );
210     }
211 
212     cylinder
213     {
214         type wall;
215         faces
216         (
217             (10 5 24 29)
218             (5 0 19 24)
219             (16 10 29 35)
220             (11 16 35 30)
221 
222             (48 11 30 61)
223             (45 48 61 58)
224             (40 45 58 53)
225             (0 40 53 19)
226         );
227     }
228 
229     frontAndBack
230     {
231         type empty;
232         faces
233         (
234            (5 10 9 4)
235            (24 23 28 29)
236            (0 5 4 1)
237            (19 20 23 24)
238            (1 4 3 2)
239            (20 21 22 23)
240            (4 7 6 3)
241            (23 22 25 26)
242            (4 9 8 7)
243            (28 23 26 27)
244            (16 15 9 10)
245            (35 29 28 34)
246            (12 15 16 11)
247            (31 30 35 34)
248            (13 14 15 12)
249            (32 31 34 33)
250            (14 17 18 15)
251            (33 34 37  36)
252            (15 18 8 9)
253            (34 28 27 37)
254 
255            (45 40 39 44)
256            (58 57 52 53)
257            (40 0 1 39)
```



```
258             (53 52 20 19)
259             (39 1 2 38)
260             (52 51 21 20)
261             (39 38 41 42)
262             (52 55 54 51)
263             (44 39 42 43)
264             (57 56 55 52)
265             (47 48 45 44)
266             (60 57 58 61)
267             (12 11 48 47)
268             (31 60 61 30)
269             (13 12 47 46)
270             (32 59 60 31)
271             (49 46 47 50)
272             (62 63 60 59)
273             (50 47 44 43)
274             (63 56 57 60)
275         );
276     }
277 );
278
279 mergePatchPairs
280 (
281 );
282
283 // ************************************************************************* //
```

It can be seen that as the cylinder has curved edges, it is necessary to use the arc instruction to obtain the circular geometry. This instruction must be followed by the labels of the connected vertices and an interpolation point contained in the trajectory of the arc.

In the current *blockMeshDict* there are four different types of patches: *wall*, *empty*, *symmetryPlane* and *patch*. All were used in the *plane-parallel plates* case except *symmetryPlane*, which is used in the top and bottom patches of the current case to indicate that there are no physical walls in the top and bottom borders; the flow must behave as if the domain would extend infinitely in the y-direction.

After running blockMesh and checking the results with checkMesh it is possible to observe the mesh with ParaView.

Caution:

> It is necessary to have the majority of the required files for the simulation within the case directory to create the mesh. Wait until the three main directories (*0*, *constant* and *system*) and their files are ready to run blockMesh. Otherwise, it is also possible to use a solved case as a *dummy* file to run blockMesh

The mesh of cylinder195 is shown at Figures 3.5, 3.6 and 3.7:

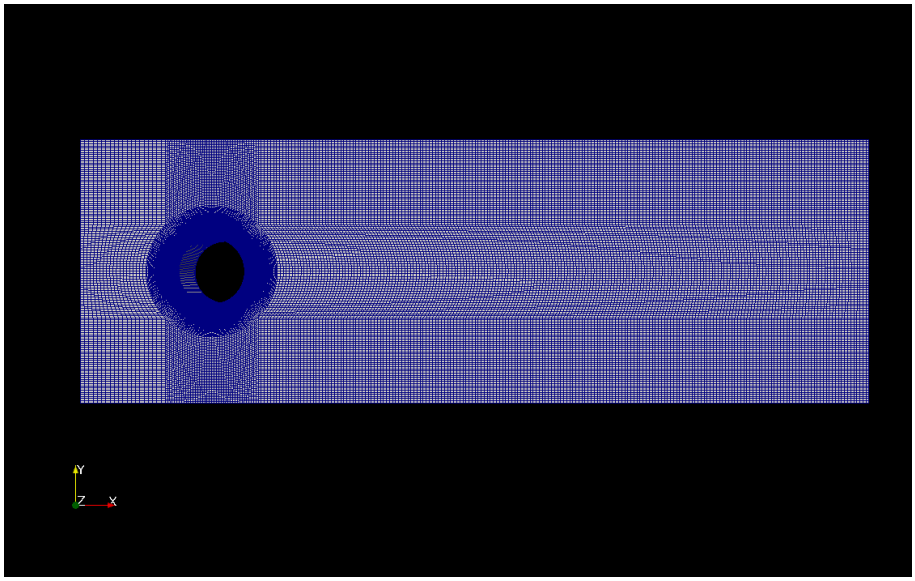


Figure 3.5: Mesh of the *bidimensional cylinder* case

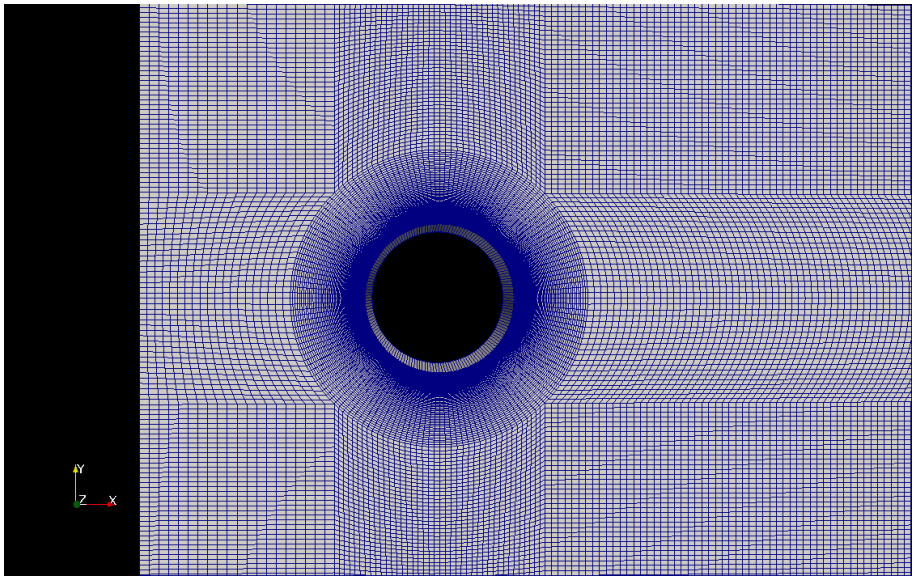


Figure 3.6: Detail of the mesh of the *bidimensional cylinder* case

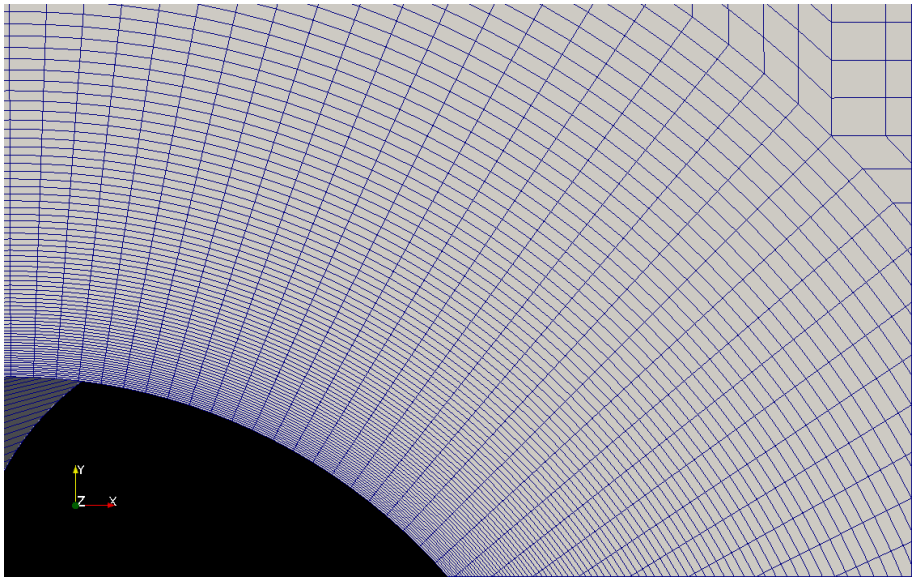


Figure 3.7: Detail of the mesh gradation on the walls of the *bidimensional cylinder* case

Advice:

> On the walls and downstream the cylinder it is necessary to have a high refined mesh. It will allow the user to compute the drag coefficient and to observe the von Kármán street both with high accuracy. However, if the simulation is too much time-consuming or it is done with a Reynolds number such that no vortices are generated, it is recommended to reduce the refinement of the mesh

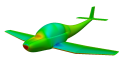
3.4.2 Boundary and initial conditions

The files (located in `0`) containing the information related to the pressure and the velocity fields are the following:

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
```

```
13      object      p;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 2 -2 0 0 0 0];
18
19  internalField   uniform 0;
20
21  boundaryField
22  {
23      top
24      {
25          type          symmetryPlane;
26      }
27
28      bottom
29      {
30          type          symmetryPlane;
31      }
32
33      inlet
34      {
35          type          freestreamPressure;
36      }
37
38      outlet
39      {
40          type          freestreamPressure;
41      }
42
43      frontAndBack
44      {
45          type          empty;
46      }
47
48      cylinder
49      {
50          type          zeroGradient;
51      }
52  }
53
54  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volVectorField;
13      object      U;
```



```
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions            [0 1 -1 0 0 0 0];
18
19  internalField         uniform (5 0 0);
20
21  boundaryField
22  {
23      top
24      {
25          type              symmetryPlane;
26      }
27
28      bottom
29      {
30          type              symmetryPlane;
31      }
32
33      inlet
34      {
35          type              freestream;
36          freestreamValue   uniform (5 0 0);
37      }
38
39      outlet
40      {
41          type              freestream;
42          freestreamValue   uniform (5 0 0);
43      }
44
45      cylinder
46      {
47          type              fixedValue;
48          value             uniform (0 0 0);
49      }
50
51      frontAndBack
52      {
53        type                empty;
54      }
55  }
56
57  // ************************************************************************* //
```

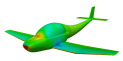
The freestreamPressure condition is acting as zeroGradient but with a more accurate physical behaviour. freestream acts as fixedValue when the flow is ingoing, and as zeroGradient when it is outgoing. This kind of boundary conditions are widely used for external flow simulations.

3.4.3 Physical properties

Within constant one finds the information related to the kinematic viscosity and the RASProperties file to compute the wall shear stress or other utilities requesting the RAS dictionary.

```
1  /*---------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "constant";
14     object      transportProperties;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 transportModel  Newtonian;
19
20 nu              nu [0 2 -1 0 0 0 0] 2.564103e-03;
21
22 // ************************************************************************* //
```

```
1  /*---------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "constant";
14     object      RASProperties;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 RASModel        laminar;
19
20 turbulence      off;
21
22 printCoeffs     off;
23
24 // ************************************************************************* //
```



3.4.4 Control

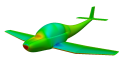
```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application         icoFoam;

startFrom           startTime;

startTime           0;

stopAt              endTime;

endTime             1.75;

deltaT              0.00001;

writeControl        timeStep;

writeInterval       1000;

purgeWrite          0;

writeFormat         ascii;

writePrecision      6;

writeCompression    off;

timeFormat          general;

timePrecision       6;

runTimeModifiable   true;

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

Advice:

The case reaches steady before $t = 1.75$ s. Despite it, endTime has been set to 1.75 to widely appreciate the von Kármán street and its periodical distribution. As the mesh is very refined and the solver is transitory, the simulation may be very slow

3.4.5 Discretization and linear-solver settings

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      fvSchemes;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 ddtSchemes
19 {
20     default          Euler;
21 }
22
23 gradSchemes
24 {
25     default          Gauss linear;
26     grad(p)          Gauss linear;
27 }
28
29 divSchemes
30 {
31     default          none;
32     div(phi,U)       Gauss linear;
33 }
34
35 laplacianSchemes
36 {
37     default          none;
38     laplacian(nu,U) Gauss linear orthogonal;
39     laplacian((1|A(U)),p) Gauss linear orthogonal;
40 }
41
42 interpolationSchemes
43 {
44     default           linear;
45     interpolate(HbyA) linear;
46 }
47
```



```
48  snGradSchemes
49  {
50      default         orthogonal;
51  }
52
53  fluxRequired
54  {
55      default         no;
56      p               ;
57  }
58
59  // ************************************************************************* //
```

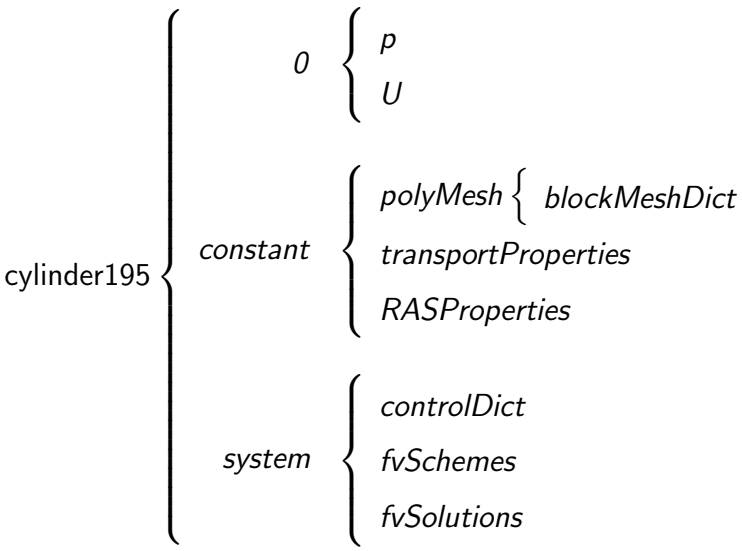
```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "constant";
14      object      transportProperties;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  solvers
19  {
20      p
21      {
22          solver          PCG;
23          preconditioner  DIC;
24          tolerance       1e-06;
25          relTol          0;
26      }
27
28      U
29      {
30          solver          PBiCG;
31          preconditioner  DILU;
32          tolerance       1e-05;
33          relTol          0;
34      }
35  }
36
37  PISO
38  {
39      nCorrectors     2;
40      nNonOrthogonalCorrectors 3;
41      pRefCell        0;
42      pRefValue       0;
43  }
```

```
44
45  // ************************************************************************* //
```

Advice:

> As it can be seen, the instruction nNonOrthogonalCorrectors has been set to 3 instead of 0 (as it was in the *plane-parallel plates* case). It is so because when checking the mesh with checkMesh, it is possible to observe that there are *mesh non-orthogonalities*. Although globally the mesh is OK to be run, it helps in obtaining more physically accurate results

At the end of the pre-processing, the structure of directories and subdirectories within cylinder195 should be as follows:



3.5 Post-processing

3.5.1 Results of the simulation with $Re = 195$

The evolution over time of the velocity is as follows:

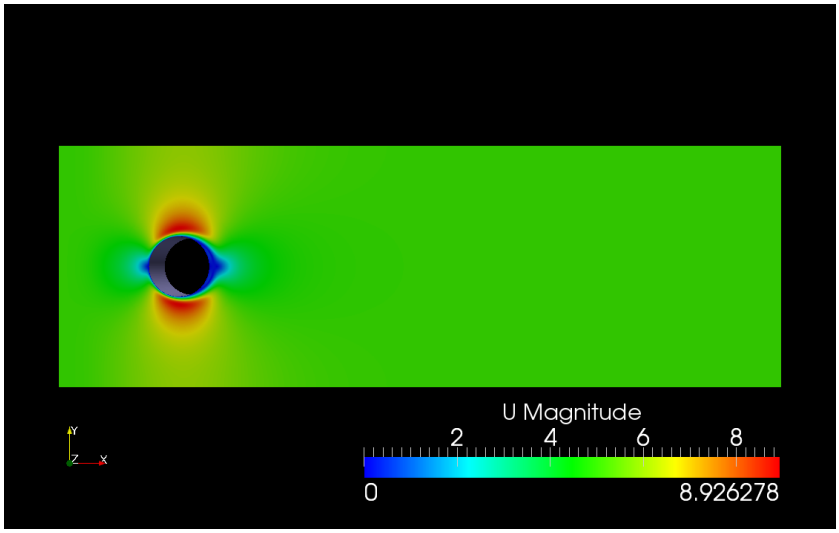


Figure 3.8: Velocity field around the bidimensional cylinder at $t = 0.01$ s (m/s)

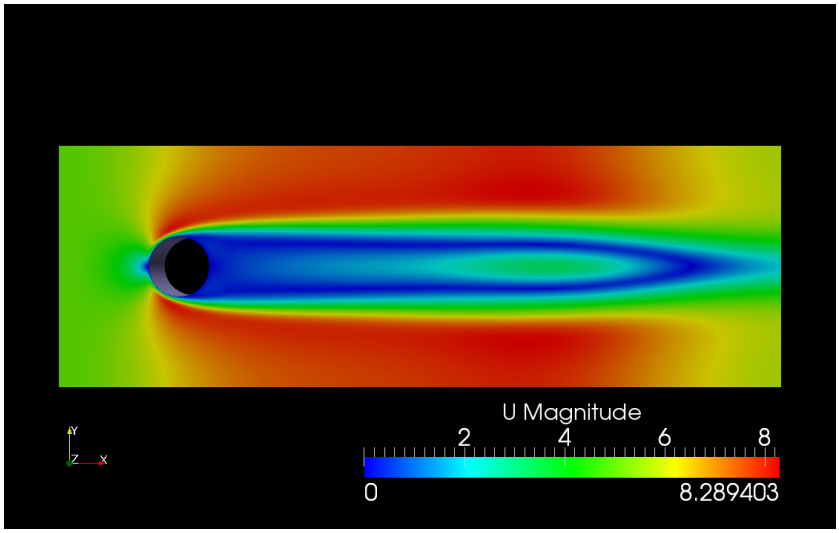


Figure 3.9: Velocity field around the bidimensional cylinder at $t = 0.6$ s (m/s)

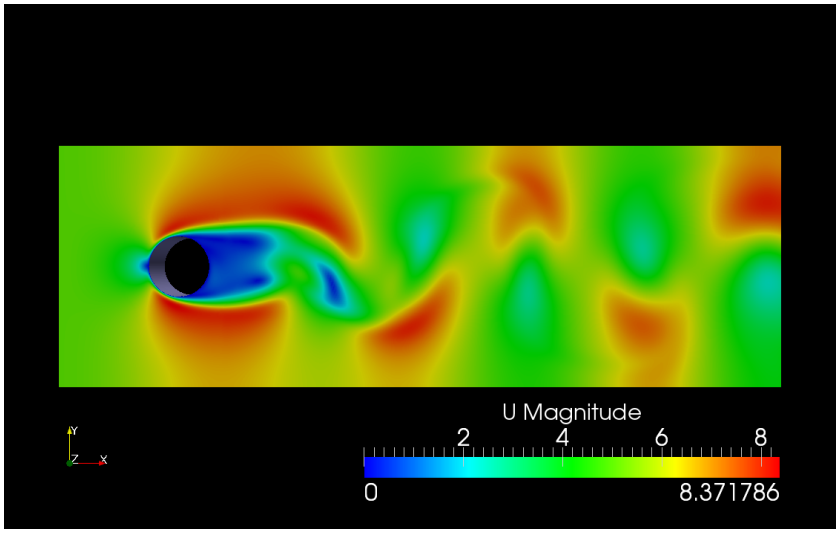


Figure 3.10: Velocity field around the bidimensional cylinder at $t = 1.13$ s (m/s)

The evolution over time of the pressure is as follows:

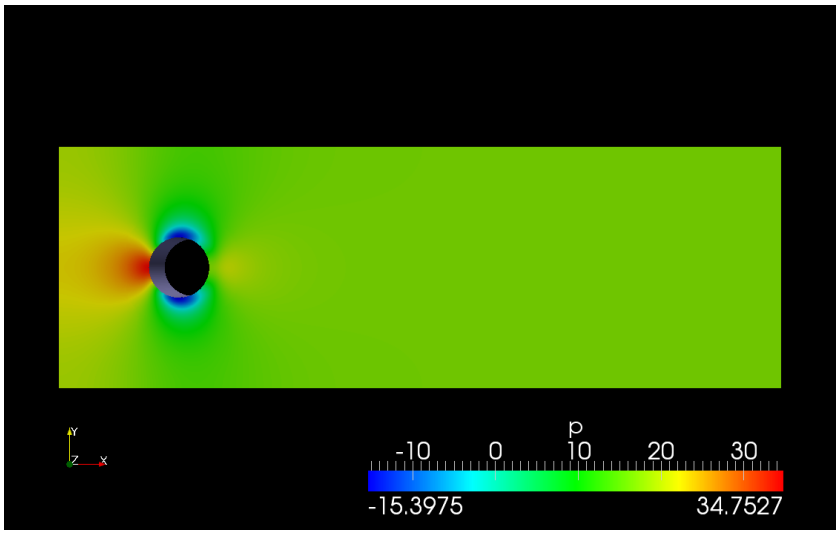


Figure 3.11: Pressure field around the bidimensional cylinder at $t = 0.01$ s (m^2/s^2)

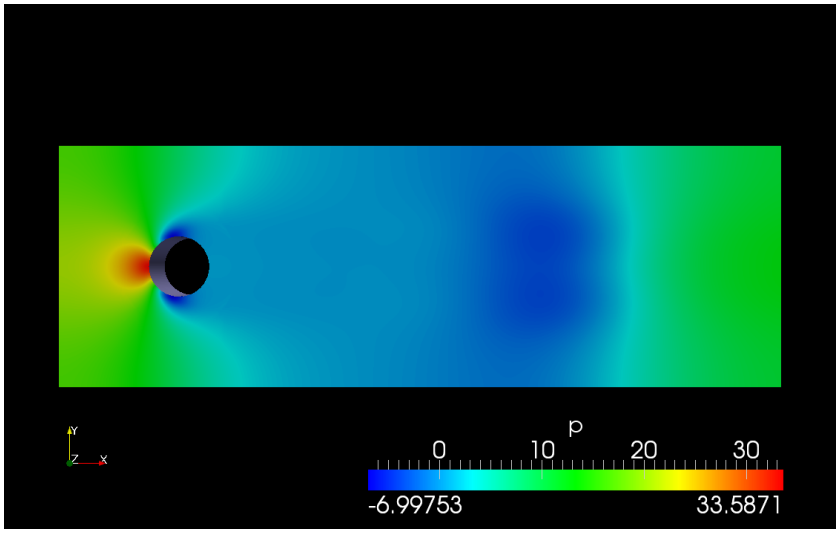


Figure 3.12: Pressure field around the bidimensional cylinder at $t = 0.6$ s (m^2/s^2)

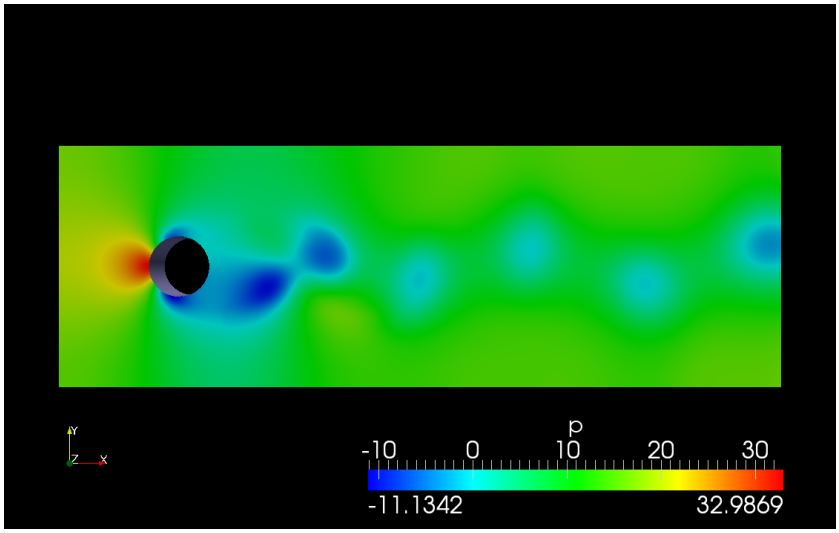


Figure 3.13: Pressure field around the bidimensional cylinder at $t = 1.13$ s (m^2/s^2)

The evolution over time of the streamlines is as follows:

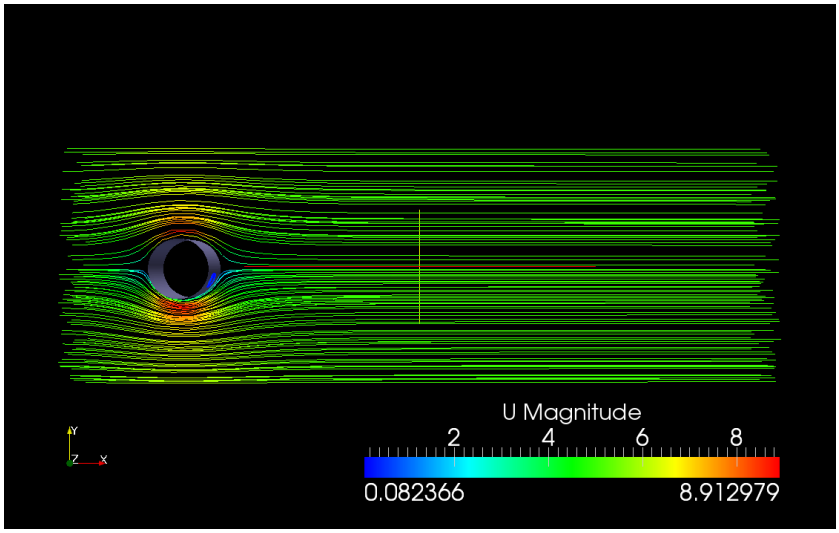


Figure 3.14: Streamlines around the bidimensional cylinder at $t = 0.01$ s (m/s)

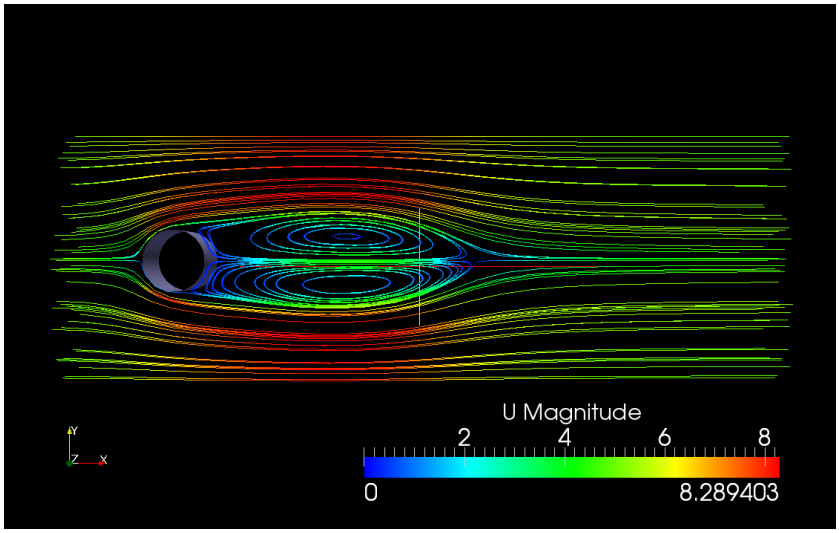


Figure 3.15: Streamlines around the bidimensional cylinder at $t = 0.4$ s (m/s)

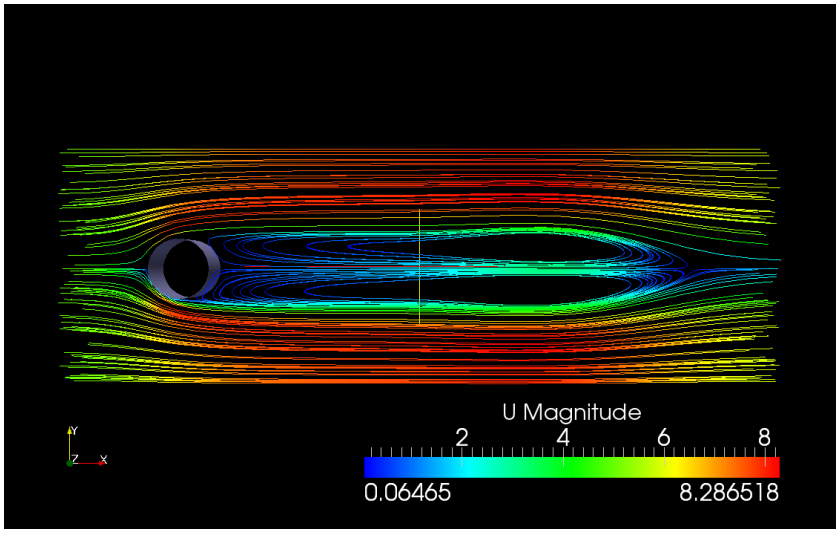


Figure 3.16: Streamlines around the bidimensional cylinder at $t = 0.6$ s (m/s)

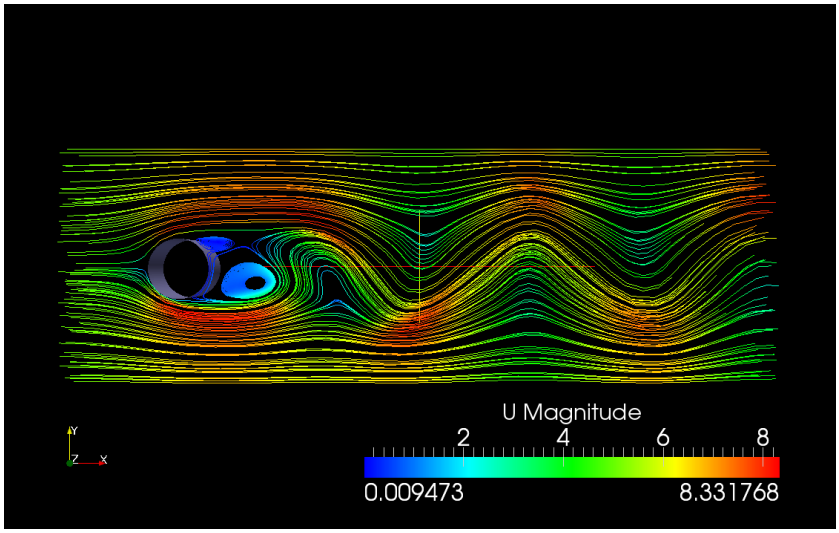


Figure 3.17: Streamlines around the bidimensional cylinder at $t = 1.13$ s (m/s)

A detail of the vector field near the wall of the cylinder is shown to appreciate the boundary layer detachment:

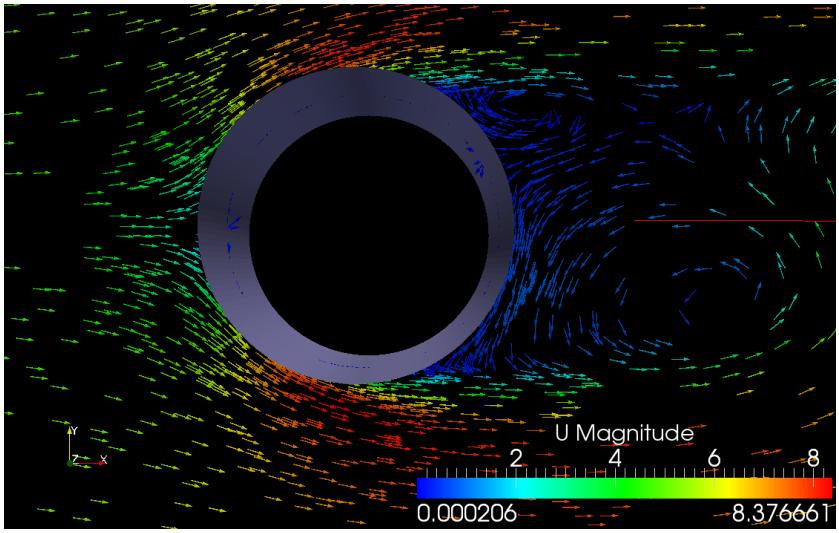


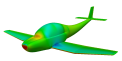
Figure 3.18: Velocity vectors around the bidimensional cylinder at $t = 1.13$ s (m/s)

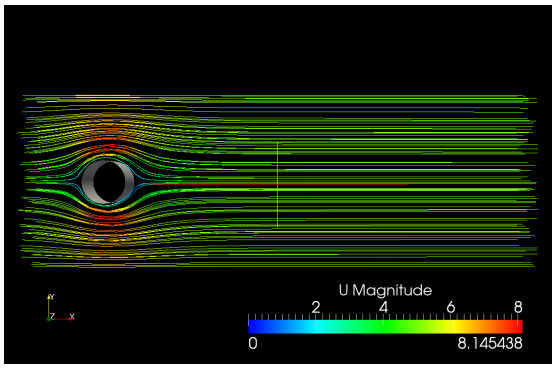
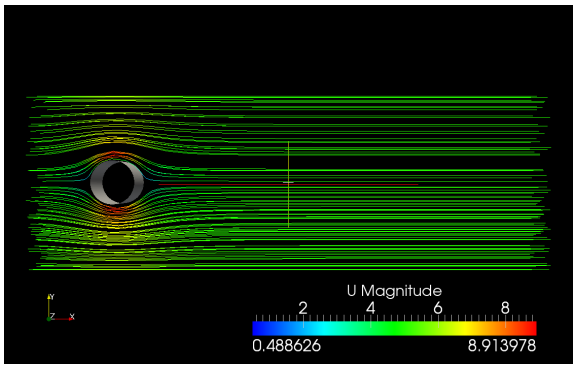
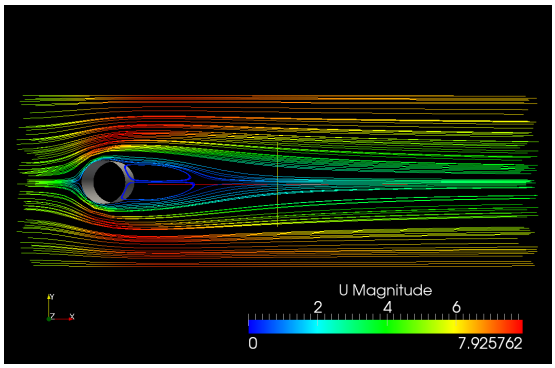
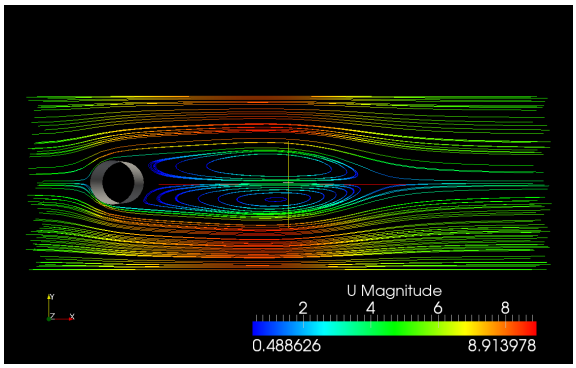
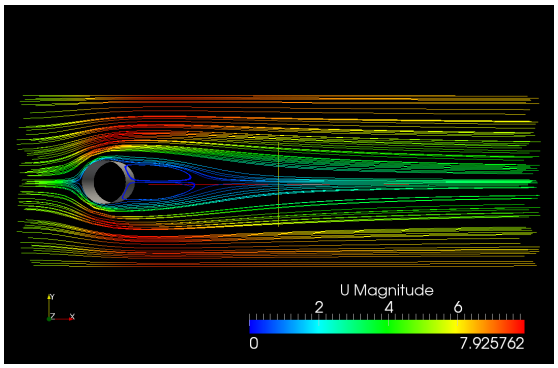
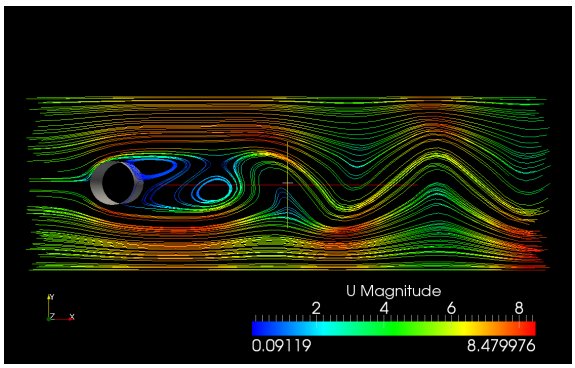
Advice:

> A way to introduce the shape of the cylinder to provide more realism to the results is shown in Section 3.6.5

3.5.2 Comparative between cases with $Re = 30$ and $Re = 195$

Additionally, the simulation has been done for another different Reynolds number ($Re = 30$). The only instruction that needs to be changed in the $OpenFOAM^{®}$ code is for instance the kinetic viscosity in *constant/transportProperties*. According to Equation 5.3, by maintaining the value of the inlet velocity while increasing the kinetic viscosity it is possible to simulate the bidimensional cylinder in such regime (case cylinder30):



$\boldsymbol{Re = 30}$	$\boldsymbol{Re = 195}$
	
$t = 0.01$ s	
	
$t = 0.4$ s	
	
$t = 1$ s	

As it was shown in Section 3.3, for $Re = 30$ the alternate vortices are not detached.

3.6 Additional utilities

3.6.1 Vorticity

As with the computation of the flow rate and the wall shear stress, it is also possible to compute the vorticity in the fluid field by using *OpenFOAM*® utilities. The vorticity is a pseudovector field that describes the local spinning motion of a fluid near some point, as would be seen by an observer located at that point and traveling along with the fluid. Mathematically, the vorticity is the curl of the velocity field:

$$\vec{w} = \nabla \times \mathbf{U}$$

The vorticity of a two-dimensional flow is always perpendicular to the plane of the flow. It plays a relevant role in the current chapter due to the existence of vortices generated by the cylinder forming the von Kármán street.

To execute it, type within the case directory

```
vorticity
```

Within the directories of each time step it has appeared a new file. Then, to observe the vorticity field, open ParaView and select it by clicking the `vorticity` box contained within Volume Fields. For the cylinder195 case, the results are:

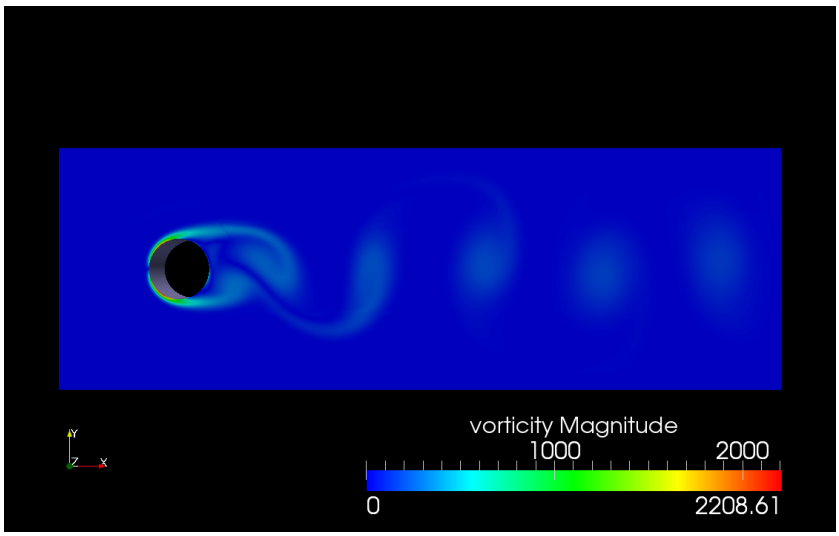
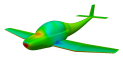


Figure 3.19: Vorticity field around the bidimensional cylinder at $t = 1.75$ s



It offers a clear-cut representation of the von Kármán vortices at $Re = 195$.

3.6.2 Computation of the aerodynamic coefficients

One of the main interests when studying external flows is the computation and analysis of the aerodynamic forces that the fluid exerts to solid objects. The drag is the force parallel to the flow velocity and the lift is the force perpendicular to the flow velocity. The dimensionless drag force is presented in Equation 5.6 and for an infinite circular cylinder its dependence on the Reynolds number is shown at Figure 3.3.

With *OpenFOAM*®, to compute the aerodynamic forces that a fluid exerts on solid walls, add the following code after the last instruction of controlDict:

```
functions
(
forces
{
type forces;
functionObjectLibs ("libforces.so");
patches (cylinder); // Patch where the force exerted by the fluid is calculated
pName p;
UName U;
rhoName rhoInf;
rhoInf 1000; // Reference density of the fluid
CofR (0 0 0); // Origin for moment calculations
outputControl timeStep; // Time criterion used to print the results
outputInterval 100; // How often (according to outputControl) the results are
    printed
}
forceCoeffs
{
type forceCoeffs;
functionObjectLibs ("libforces.so");
patches (cylinder); // Patch where the force exerted by the fluid is calculated
pName p;
UName U;
rhoName rhoInf;
rhoInf 1000; // Reference density of the fluid
CofR (0 0 0); // Origin for moment calculations
liftDir (0 1 0);
dragDir (1 0 0);
pitchAxis (0 0 1);
magUInf 5; // Free stream velocity
lRef 0.1; // Reference length (diameter of the cylinder)
Aref 0.01; // Reference area (cross sectional area of the cylinder)
outputControl timeStep; // Time criterion used to print the results
outputInterval 100; // How often (according to outputControl) the results are
    printed
}
```

```
35 );
```

Advice:

> For incompressible cases, the value of rhoInf is irrelevant for the computation of the dimensionless coefficients

Now, when the case is rerun, a new directory named *postProcessing* appears next to *0*, *constant* and *system*. This directory contains two subdirectories with information concerning the evolution of the aerodynamic forces and moments and their dimensionless coefficients.

3.6.3 Plotting the results with Gnuplot

Once the aerodynamic forces have been computed with the instruction shown in Section 3.6.2, it is useful to plot the results. Besides showing the behaviour of the forces with time, it allows an understanding of the convergence (or divergence) of the case. For the *bidimensional cylinder* case, it is possible to claim that if the drag coefficient converges then the case converges too.

First of all the user has to have Gnuplot installed. It is a portable command-line driven graphing utility for Linux, MS Windows, OSX and many other platforms. It is widely used to plot data obtained with *OpenFOAM*®. First, access the file containing the required data:

```
cd FoamCases/cylinder195/postProcessing/forceCoeffs/0
```

Secondly, execute Gnuplot by typing:

```
gnuplot
```

Finally plot the values of the drag coefficient (third column) in front of the time (first column) by typing:

```
plot './forceCoeffs.dat' u 1:3 w l
```

The plot is:

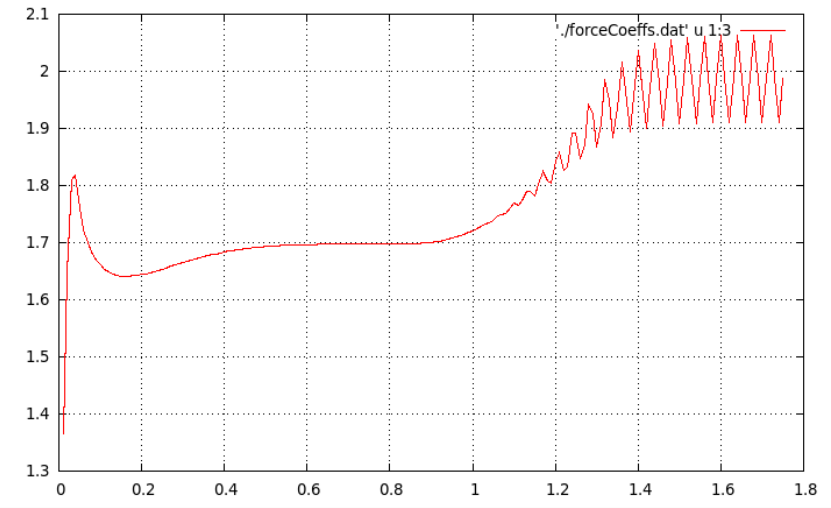


Figure 3.20: Drag coefficient (ordinate axis) of the bidimensional cylinder at $Re = 195$ in front of time (abscissa axis)

It is also possible to plot the values of the lift coefficient (fourth column) in front of the time (first column) by typing:

```
plot './forceCoeffs.dat' u 1:4 w l
```

The plot is:

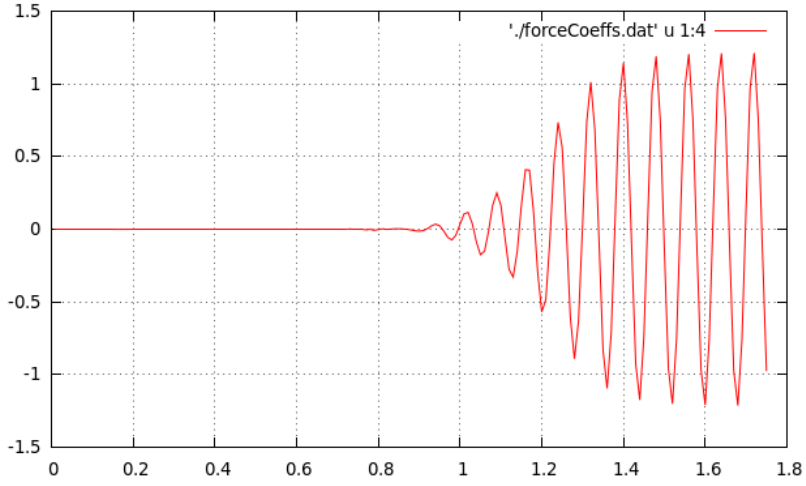


Figure 3.21: Lift coefficient (ordinate axis) of the bidimensional cylinder at $Re = 195$ in front of time (abscissa axis)

3.6.4 Computation of the stream function

As it was explained in Section 2.6.2, the streamlines offer a clear understanding of the behaviour of the flow; they represent the trajectories of particles in steady fluids. Related to it, there exists a scalar function (stream function) such that the flow velocity components can be expressed as the derivatives of this function, also being used to plot the streamlines. Mathematically it is related to the velocity as:

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x}$$

Since streamlines are tangent to the velocity vector of the flow, the value of the stream function must be constant along a streamline.

To obtain the stream function of the velocity, type:

```
streamFunction
```

To view the results with ParaView, it is necessary to select the `streamFunction` box located within Point Fields. Here are the results:

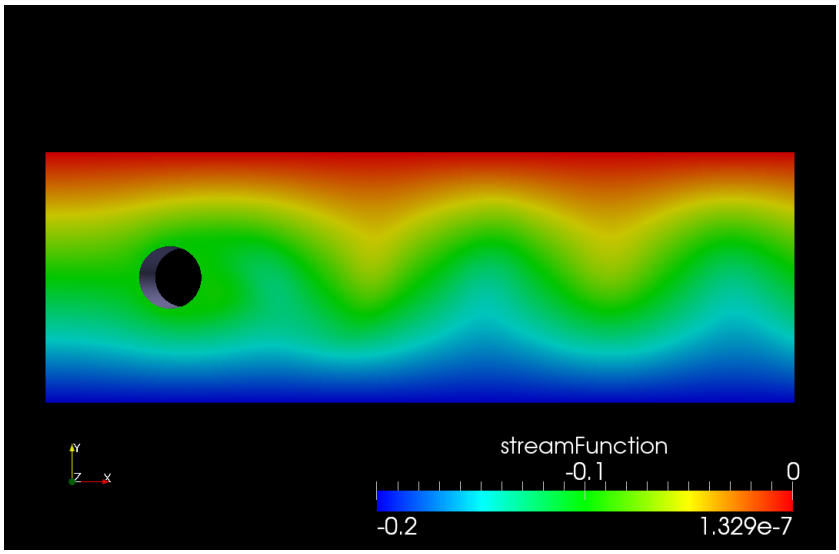
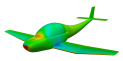


Figure 3.22: Stream function of the velocity of the *bidimensional cylinder* case for $Re = 195$ at $t = 1.75$ s

As it can be seen, it follows the same trend as the streamlines but with a continuous appearance. The colour indicates the value of the stream function in a particular point (note that this utility generates a point field), taking the $\psi = 0$ streamline as



the one at the top patch of the domain.

It can be proved that the volumetric flow rate between two streamlines is equal to the difference between their stream functions. This helps in the validation of the streamFunction utility. At the inlet:

$$Q = V \cdot S = 5 \cdot 0.4 \cdot 0.1 = 0.2\ m^3/s$$

$$Q = \psi_n - \psi_0 = 0.2\ m^3/s$$

3.6.5 Convertion to VTK

It is possible to convert data from *OpenFOAM*® to VTK format. For instance, access cylinder195 and type:

```
foamToVTK
```

A new directory appears within the case containing the VTK data. Since it is a worldwide used format, it is also possible to open it with ParaView.

Example: Once in ParaView, click on the "open" icon, access the VTK directory of the case and click on "cylinder". Within the Pipeline Browser, a new module has appeared. It is the cylinder patch whose shape can be used to give more realism to the results of the simulation. This same procedure can be carried out for each one of the defined patches of the case.

4. Laminar flow through a circular pipe

4.0.6 Description of the case

As done in Chapter 2, this tutorial studies confined flow. Nevertheless, in the current chapter one of the most representative cases of internal fluid is presented: flow through a circular pipe. It is a very typical problem in the fluid mechanics field because of its wide presence in a great number of experiments, analysis and our daily life. As it is the basic fluid conveyance, the circular pipe and its influence on the circulating flow behaviour has been widely studied in the literature.

4.0.7 Hypotheses

- Incompressible flow
- Laminar flow
- Newtonian flow
- Cylindrical simmetry ($\frac{\partial}{\partial \theta} = 0$)
- Negligible gravitatory effects
- Steady flow ($\frac{\partial}{\partial t} = 0$)
- Very large pipe so that when flow becomes fully developed, $v_z \neq 0$ but $v_\theta = 0$, $v_r = 0$
- Smooth pipe so roughness does not influence

4.0.8 Physics of the problem

The problem encompasses a $L = 0.2$ m length circular pipe with a diameter of $D = 8$ mm. An inlet volumetric flow rate of $Q = 25.6\ cm^3/\text{s}$ and an outlet pressure

of $p_{outlet} = 9000$ Pa are imposed. For the mathematical introductory analysis and due to the fact that the case is axi-symmetric, cylindrical coordinates (r, θ, z) are used. z is parallel to the axis of the pipe, r points to the wall of the pipe and is perpendicular to the line tangent to the contour in the intersection point, and θ completes the coordinate system. The problem statement is shown at Figure 4.1.

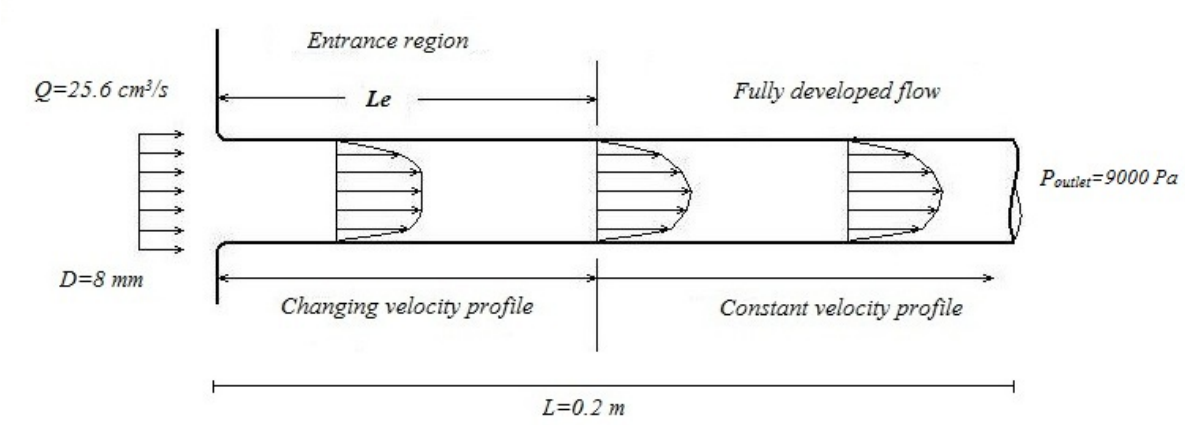


Figure 4.1: Flow through a circular pipe with a constant inlet velocity

As it can be seen, when the inlet flow enters the pipe, the viscous boundary layers grow up downstream, slowing the axial flow on the wall while accelerating the central core to mantain continuity (Equation 4.2).

In a finite distance from the inlet (entrance length, L_e), the boundary layers join and the fluid becomes entirely viscous (fully developed flow). When it happens, the velocity profile becomes constant, the wall shear stress becomes constant and the pressure changes linearly with z. For laminar flow, the entrance length is commonly accepted as being:

$$\frac{L_e}{D} \approx 0.06\ Re \quad (laminar) \tag{4.1}$$

According to the hypotheses, if the pipe is very large and for $z > L_e$, the only non-zero component of the velocity is v_z. In this domain, there is an easy mathematical solution to describe the behaviour of the flow. Thus, the current CFD analysis will be useful to corroborate this anaylitical model and to figure out the behaviour of the flow for $z < L_e$, which is more complex to describe mathematically.

For $z > L_e$, the continuity equation in cylindrical coordinates is

$$\frac{1}{r}\frac{\partial}{\partial r}(rv_r) + \frac{1}{r}\frac{\partial v_\theta}{\partial \theta} + \frac{\partial v_z}{\partial z} = 0 \Rightarrow \frac{\partial v_z}{\partial z} = 0 \Rightarrow v_z = v_z(r) \tag{4.2}$$

The momentum equation for v_z,

$$\frac{\partial v_z}{\partial t} + v_r\frac{\partial v_z}{\partial r} + v_\theta\frac{1}{r}\frac{\partial v_z}{\partial \theta} + v_z\frac{\partial v_z}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \frac{\mu}{\rho}\left[\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial v_z}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 v_z}{\partial \theta^2} + \frac{\partial^2 v_z}{\partial z^2}\right] \tag{4.3}$$

is reduced to

$$\frac{\mu}{r}\frac{d}{dr}\left(r\frac{dv_z}{dr}\right) = \frac{dp}{dz}$$

and as $p = p(z)$ according to the momentum equation for v_r, the right hand side of the equation is constant and < 0.

Integrating twice, applying the boundary condition $v_z = 0$ in $r = \frac{D}{2}$ and knowing that the first constant must be 0 because $ln\ 0$ is a singularity in $r = 0$, the analytical solution takes the form

$$v_z = \left(-\frac{dp}{dz}\right)\frac{1}{4\mu}\left(\frac{D^2}{4} - r^2\right) \tag{4.4}$$

This is the general solution of the flow through a circular pipe for $z > L_e$, a parabolic velocity profile (Hagen-Poiseuille flow). Additionally, an interesting value to be computed is the maximum velocity corresponding to the vertex of the parabola, which is

$$v_{z_{max}} = \left(-\frac{dp}{dz}\right)\frac{D^2}{16\mu} \tag{4.5}$$

For internal flow, the transition from laminar to turbulent happens at a Reynolds number of about $Re \approx 2300$. Therefore, as the simulation is made for laminar flow, its physical properties will be set accordingly: an oil is used with $\rho = 900\ kg/m^3$ and $\mu = 0.0288$ Pa·s. Then the Reynolds number is

$$Re = \frac{\rho DV}{\mu} = 65.48$$

4.0.9 Pre-processing

The following codes contain the information to simulate the circular pipe with $Re = 65.48$ using icoFoam. The case directory is named circularPipe and will be located

within FoamCases. Its structure of directories and subdirectories is very similar to the one used in Chapter 2 and Chapter 3.

4.0.9.1 Mesh generation

The mesh of the circularPipe case includes a new feature: a block with fewer than 8 vertices will be used. There is an axi-symmetry in the case so it is not necessary to mesh the whole pipe but only a wedge of it. Since $\frac{\partial}{\partial \theta} = 0$ for the whole domain, the results of the simulation will be the same while the number of elements of the mesh will be lower. Figure 4.2 represents a schematic view of the geometry that it is going to be created:

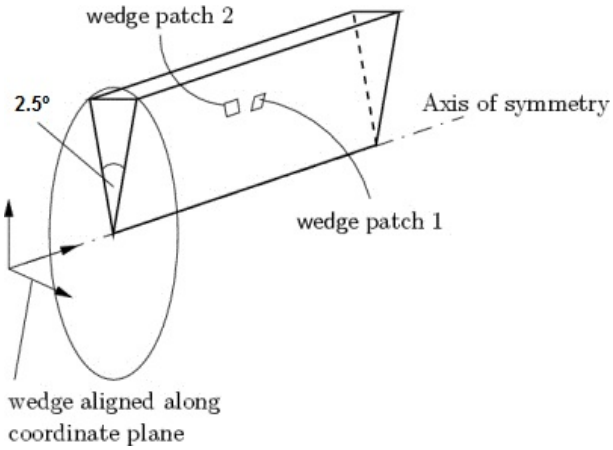
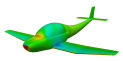


Figure 4.2: Scheme of the domain of the circularPipe case, extracted from [1]

The *blockMeshDict* contains:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;

```

```
19  vertices
20  (
21      (0 0 0)
22      (0 3.9990481 -0.0872595)
23      (0 3.9990481 0.0872595)
24      (200 0 0)
25      (200 3.9990481 -0.0872595)
26      (200 3.9990481 0.0872595)
27  );
28
29  blocks
30  (
31      hex (0 1 2 0 3 4 5 3) (50 1 100) simpleGrading (0.1 1 10)
32  );
33
34  edges
35  (
36     arc 1 2 (0 4 0)
37     arc 4 5 (200 4 0)
38  );
39
40  boundary
41  (
42      axis
43      {
44         type empty;
45         faces
46         (
47              (0 3 3 0)
48         );
49      }
50
51      inlet
52      {
53          type patch;
54          faces
55          (
56              (0 0 2 1)
57          );
58      }
59      wall
60      {
61          type wall;
62          faces
63          (
64              (2 5 4 1)
65          );
66      }
67      outlet
68      {
69          type patch;
70          faces
71          (
72              (3 4 5 3)
73          );
74      }
75
```



```
76      front
77      {
78          type wedge;
79          faces
80          (
81              (0 3 5 2)
82          );
83       }
84
85       back
86      {
87          type wedge;
88          faces
89          (
90              (0 1 4 3)
91          );
92       }
93  );
94
95  mergePatchPairs
96  (
97  );
98
99  // ************************************************************************* //
```

When running blockMesh, the created mesh is the following:

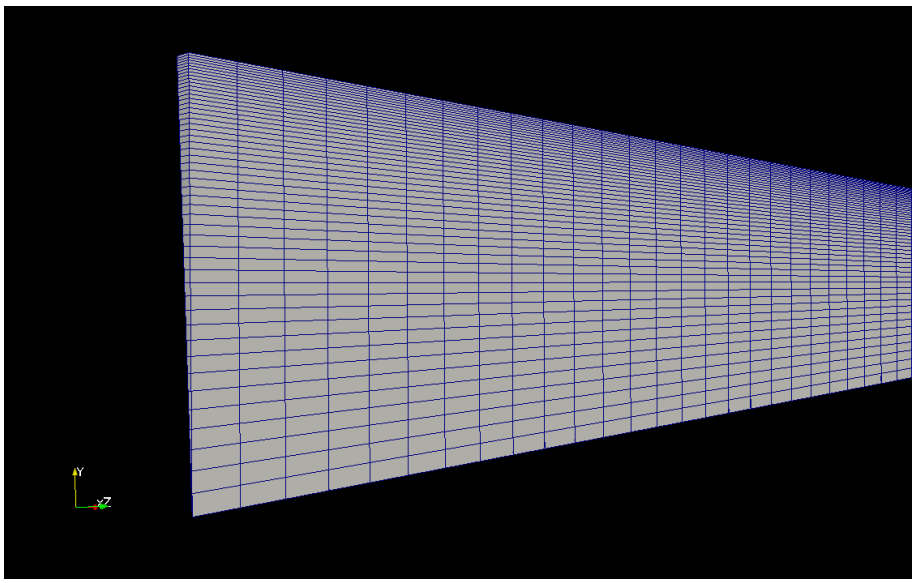


Figure 4.3: Initial mesh of the circularPipe case

First of all, it is important to comprehend the collapsing of the vertices. It can be seen at line 31 that vertices 0 and 3 are repeated in an adequate position within the

parentheses to indicate that the two vertices that would be occupying this position have been collapsed.

Secondly, in the patches definition, the same vertices 0 and 3 are joined creating an empty patch forming the axis of the pipe. The patch type wedge is used in two faces (front and back at lines 76 and 85) to indicate that an axi-symmetry exists and there are no physical walls (wedge patch 1 and wedge patch 2 in Figure 4.2).

Caution:

> The user should have noted that the vertices are written such that the wedge represents a 2.5º section of the pipe. To avoid errors, any edge of the wedge may coincide with the axis of the global coordinate system

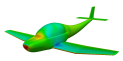
There is a grading towards the wall to compute a more exact value of wall shear stress and towards the inlet where the flow is not fully developed and therefore the velocity profile changes.

4.0.9.2 Boundary and initial conditions

Advice:

> When solving circulating internal flow, the most common boundary conditions are to fix an inlet velocity and an outlet pressure (or viceversa)

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.1.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      p;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     axis
24     {
```



```
25          type            empty;
26      }
27
28      inlet
29      {
30          type            zeroGradient;
31      }
32
33      wall
34      {
35          type            zeroGradient;
36      }
37
38      outlet
39      {
40          type            fixedValue;
41          value           uniform 10;
42      }
43
44      front
45      {
46          type            wedge;
47      }
48
49      back
50      {
51          type            wedge;
52      }
53  }
54
55  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.1.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volVectorField;
13      object      U;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (0.50929 0 0);
20
21  boundaryField
22  {
23      axis
24      {
```

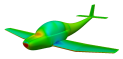
```
25          type            empty;
26      }
27
28      inlet
29      {
30          type            fixedValue;
31          value           uniform (0.50929 0 0);
32
33      }
34
35      wall
36      {
37          type            fixedValue;
38          value           uniform (0 0 0);
39      }
40
41      outlet
42      {
43          type                inletOutlet;
44          inletValue uniform  (0.50929 0 0);
45          value uniform       (0.50929 0 0);
46      }
47
48      front
49      {
50          type            wedge;
51      }
52
53      back
54      {
55          type            wedge;
56      }
57  }
58
59  // ************************************************************************* //
```

The inlet velocity has been computed as:

$$V = \frac{Q}{\pi \cdot (D/2)^2} = \frac{25.6 \times 10^{-6}}{\pi \cdot (4 \times 10^{-3})^2} = 0.50929\ m/s$$

In the **U** dictionary one finds a new type of boundary condition: inletOutlet. It switches **U** and p between fixedValue and zeroGradient: the first when the flow is ingoing and the second when it is outgoing.

Example: By applying it in the circularPipe case, the outlet velocity will always be adapted to the conditions according to zeroGradient except if there is an inflow at the outlet patch. If this happens, to this inflow velocity it is assigned the value specified under the inletOutlet instruction and so the flow will always be outgoing.



4.0.9.3 Physical properties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 0.000032;

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

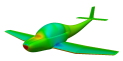
```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel        laminar;

turbulence      off;

printCoeffs     off;

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

4.0.9.4 Control

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.2;

deltaT          0.00005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

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

4.0.9.5 Discretization and linear-solver settings

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
```



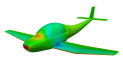
```
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      fvSchemes;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  ddtSchemes
19  {
20      default         Euler;
21  }
22
23  gradSchemes
24  {
25      default         Gauss linear;
26      grad(p)         Gauss linear;
27  }
28
29  divSchemes
30  {
31      default         none;
32      div(phi,U)      Gauss linear;
33  }
34
35  laplacianSchemes
36  {
37      default         none;
38      laplacian(nu,U) Gauss linear corrected;
39      laplacian((1|A(U)),p) Gauss linear corrected;
40  }
41
42  interpolationSchemes
43  {
44      default         linear;
45      interpolate(HbyA) linear;
46  }
47
48  snGradSchemes
49  {
50      default         corrected;
51  }
52
53  fluxRequired
54  {
55      default         no;
56      p               ;
57  }
58
59  // ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-07;
        relTol          0;
    }

    U
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
}

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

Advice:

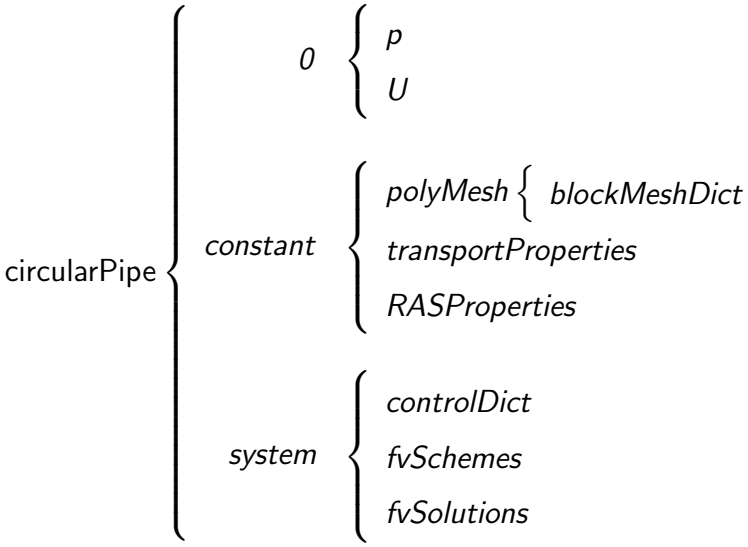
In the *fvSolution* file of the circularPipe case, the absolute tolerances are set lower than other cases. The tolerances are indicative of the error that the user *accepts* in the iterative resolution procedure. The lower the tolerances, the more accurate the results (but lower the simulation speed)

Advice:



Below the solver definition, it has to be specified PISO or SIMPLE and its characteristics. These algorithms are iterative procedures for solving equations for velocity and pressure, PISO being used for transient problems and SIMPLE for steady-state. When using SIMPLE, nCorrectors has to be set to 1, whereas when using PISO, it should be higher than 1 but typically not more than 4

At the end of the pre-processing of the circularPipe case, the scheme of directories and sub-directories is the following:



4.0.10 Post-processing

4.0.10.1 Results of the simulation

The velocity field at the inlet of the pipe (non-developed flow):

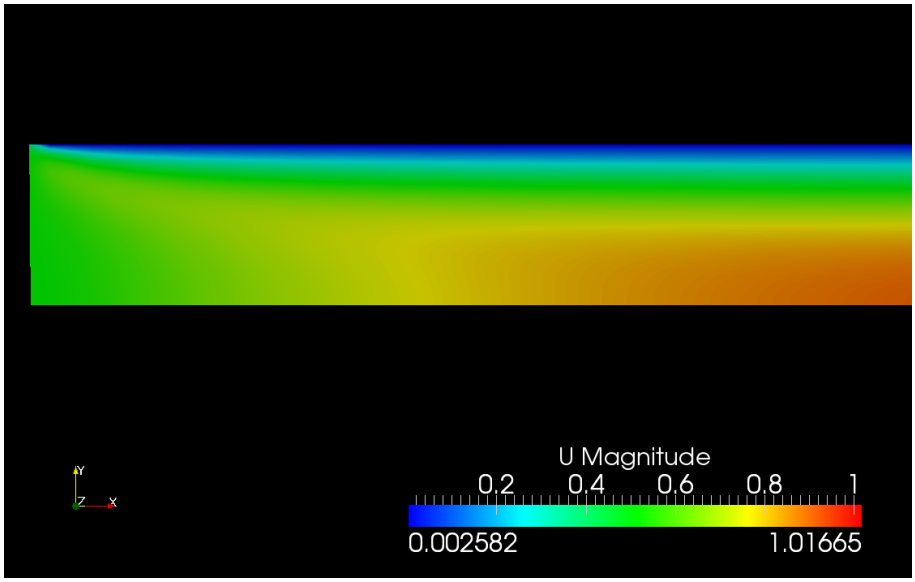


Figure 4.4: Velocity field at the inlet of the pipe in the circularPipe case (m/s)

The velocity field at the outlet of the pipe (developed flow):

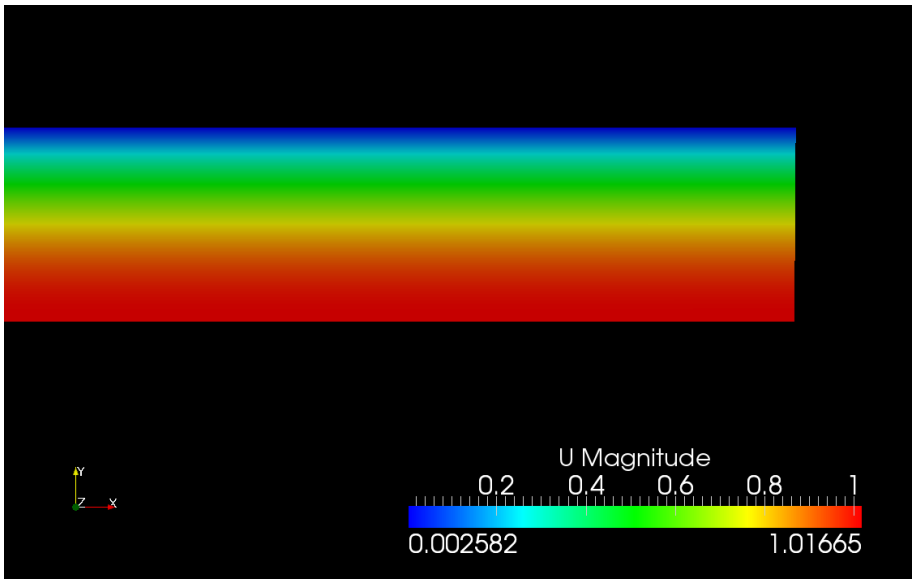


Figure 4.5: Velocity field at the outlet of the pipe in the circularPipe case (m/s)

The pressure field along the pipe:

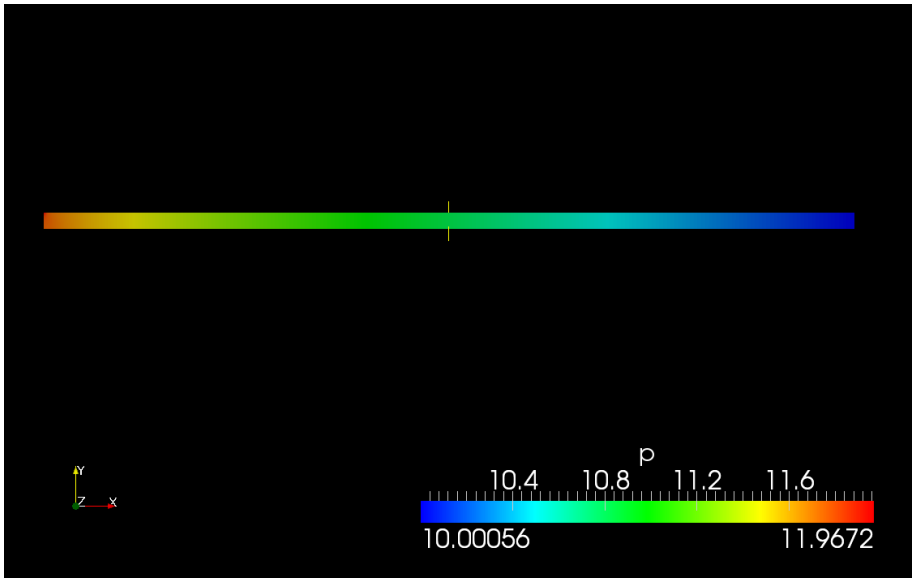


Figure 4.6: Pressure field in the circularPipe case (m^2/s^2)

The streamlines at the inlet of the pipe (non-developed flow):

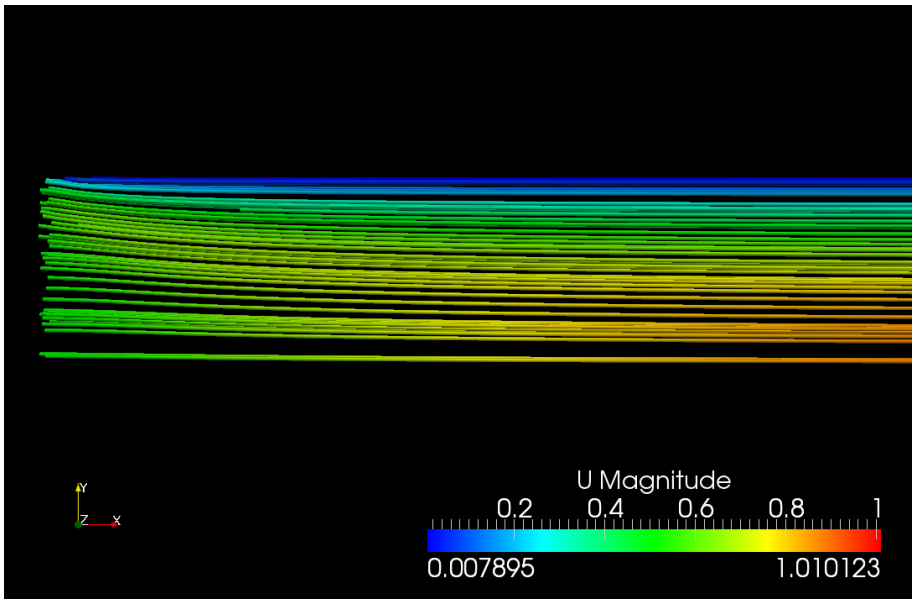


Figure 4.7: Streamlines of the flow at the inlet of the pipe in the circularPipe case (m/s)

The streamlines at the outlet of the pipe (developed flow):

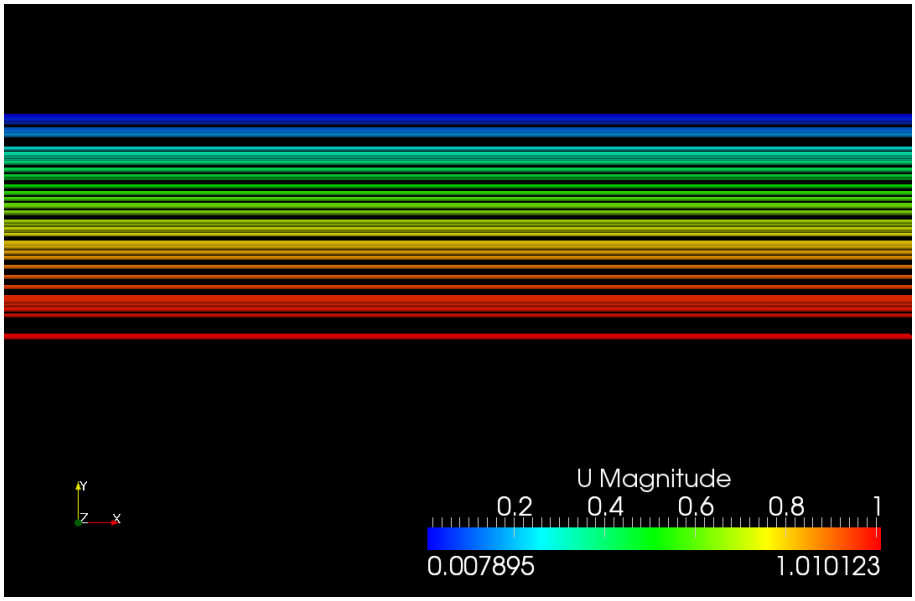


Figure 4.8: Streamlines of the flow at the outlet of the pipe in the circularPipe case (m/s)

To observe if the analytical equation of the velocity for $z > L_e$ (developed flow) shown in Section 4.0.8 matches with the results of the simulation, $|\mathbf{U}|$ at outlet as a function of r has been plot:

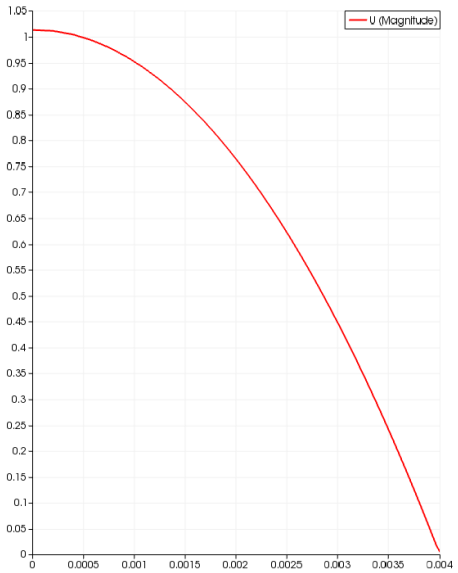
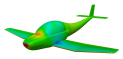


Figure 4.9: Plot of $|\mathbf{U}|$ (ordinate axis) as a function of r (abscissa axis) at the outlet of the pipe in the circularPipe case



As it can be seen, the velocity profile is a branch of a parabola. According to Equation 4.5, the maximum velocity at a section with developed flow is $v_{z_{max}} = 1.0190$ m/s. According to the results of the simulation, the maximum velocity is $|\mathbf{U}| = 1.0167$ m/s. It represents a relative error of $e_r = 0.226\%$.

Caution:

> Unlike in the *plane-Poiseuille flow* case, $\frac{dp}{dz}$ cannot be computed as $\frac{p_{outlet} - p_{inlet}}{L}$ because in the region of non-developed flow, the pressure gradient is not linear. Figure 4.10 can be used to compute it

According to the initial explanations, the pressure of the domain is linear for $z > L_e$ but not for $z < L_e$. This trend can be observed in the results of the simulation:

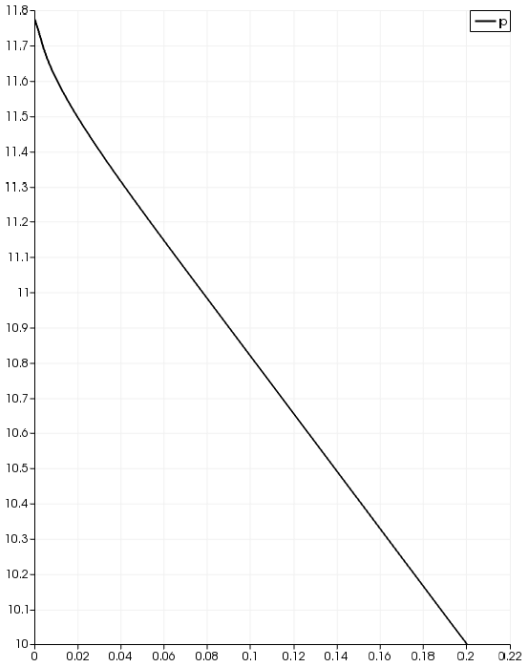


Figure 4.10: Plot of p (ordinate axis) as a function of z (abscissa axis) in the circularPipe case

Finally, it is shown the plot of the wall shear stress as a function of z:

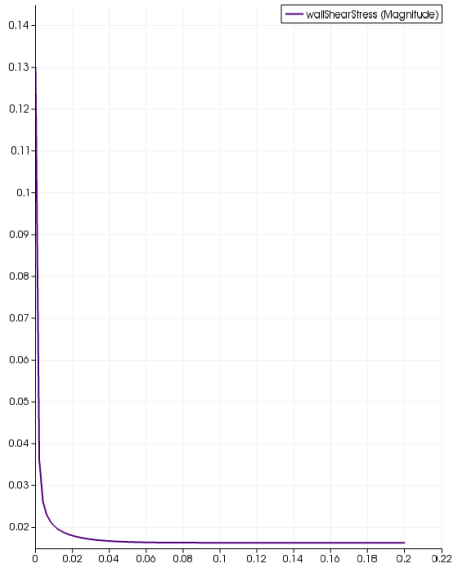


Figure 4.11: Plot of τ_w (ordinate axis) as a function of z (abscissa axis) in the circularPipe case

As it can be seen, the wall shear stress (τ_w) turns to constant for $z > L_e$ (obviously, inasmuch as the velocity profile turns to constant too). This plot allows the user to observe an approximate value of L_e according to the numerical simulation and compare it to Equation 4.1.

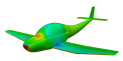
Caution:

> The wall shear stress has to be plotted and read when the patch wall is selected

4.0.11 Additional utilities

4.0.11.1 Computation of average field values at patches

In the current case, there is an imposed inlet velocity and an imposed outlet pressure. However, one variable that would be interesting to compute is the inlet pressure that exists in the pipe according to such boundary conditions. Knowing this factor may help the user in the computation of some useful values such as the pressure drop. There is a tool in *OpenFOAM®* to compute average values of a concrete field in any patch of the domain (previously defined in *blockMeshDict*). It is the patchAverage utility, which must be followed by the name of the field and the patch where it is calculated. For instance, to compute the average pressure at the inlet



while redirecting it to a file, type:

```
patchAverage p inlet > inletPressure
```

There, it is possible to observe that the solution stabilizes at $t = 0.159$ s and that the average value of the pressure at inlet is $p = 11.8239 \cdot 900 = 10641.51$ Pa. Therefore, the pressure drop within the pipe due to the viscous forces is 1641.51 Pa.

The same utility can be used to compute average vector fields such as $|\mathbf{U}|$. For instance, to corroborate that the continuity equation is fulfilled, type:

```
patchAverage U outlet > outletVelocity
```

By doing this, a file is generated with the average velocity vector at outlet. It can be seen that the average velocity is maintained between the inlet and the outlet.

4.0.11.2 Read field values with ParaView

Althoug when running icoFoam all the field data are stored within the subdirectories of the case, it is possible to read concrete field values while using ParaView. It is useful to obtain rapid data from the cells of the domain during the analysis of the results.

To do it, the user has to launch ParaView, select the last time of the simulation and check that the field or fields whose data are to be read are activated (in this example, p and $\mathbf{U}$). Then click on the rectangular icon (red circle at the top right corner of Figure 4.12) and select Spreadsheet View. It appears the information of the whole domain. To select only a specific cell or set of cells, click on the icon shown in the second red circle. Then, after clicking the icon surrounded by the purple circle, select the region of the geometry whose data are to be read.

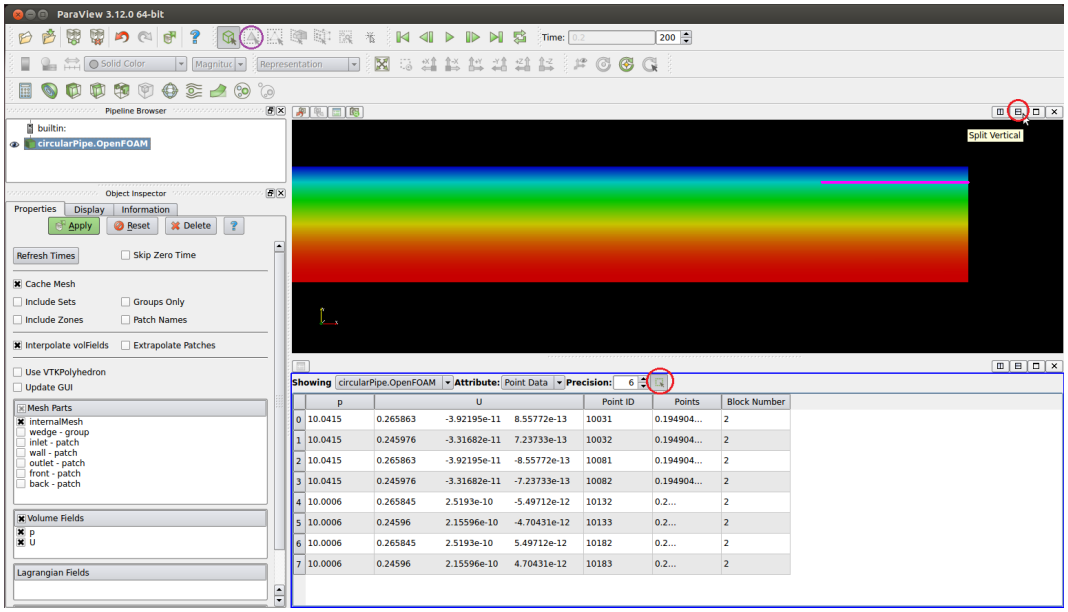


Figure 4.12: Menu to read field data in ParaView

4.0.11.3 Plot the residuals of the simulation

A fundamental point in the numerical simulations is the convergence of the solution. The resolutive procedure is an iterative process and by definition values are changing from one iteration to the next. Every numerical solution contains errors, but it is important to understand how big it is compared to the magnitude of the variable. If the solution does not converge, normally unphysical results are obtained.

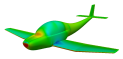
One way of measuring the convergence is with the residuals. It represents the absolute error in the solution of a particular variable. The system iterates until the residuals achieve the values of the tolerances defined in *controlDict*.

In the current section, it is shown how to plot these residuals. It may help the user in understanding if the solution has converged, and if so, how fast has it been.

Caution:

> A converging solution does not necessarily guarantee that the results of the simulation are correct. Factors such as a coarse mesh or a bad problem definition may cause the solver to produce inaccurate results

The graphical results of the residuals are going to be obtained by plotting the values within *log.icoFoam*, the file containing the information redirected while icoFoam was running. If it is not yet created, type:



```
icoFoam > log.icoFoam
```

The residuals are going to be plotted with Gnuplot by executing a Script. Copy within the case in a gedit sheet named executePlotResiduals the following instructions:

```
set logscale y
set title "Residuals"
set ylabel 'Residual'
set xlabel 'Iteration'
plot "< cat log.icoFoam | grep 'Solving for Ux' | cut -d' ' -f9" title 'Ux' with
    lines,\
"< cat log.icoFoam | grep 'Solving for Uy' | cut -d' ' -f9" title 'Uy' with lines,\
"< cat log.icoFoam | grep 'Solving for Uz' | cut -d' ' -f9" title 'Uz' with lines,\
"< cat log.icoFoam | grep 'Solving for p' | cut -d' ' -f9 | tr -d ',' " title 'p'
    with lines
pause 1000
```

Make the Script executable by typing:

```
sudo chmod a+x executePlotResiduals
```

Execute it to plot the results with Gnuplot by typing within the case:

```
gnuplot executePlotResiduals
```

The plots of the residuals are shown in Figures 4.13 and 4.14 (they are presented separately for a better understanding):

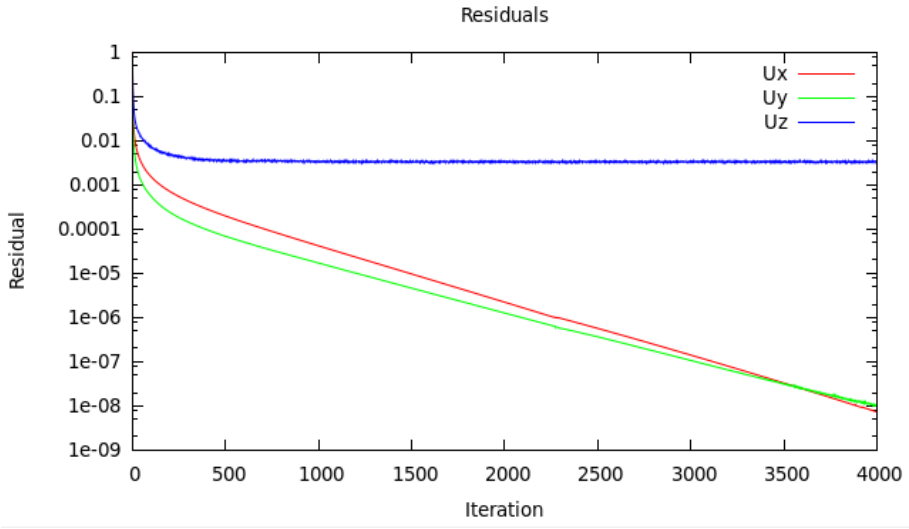


Figure 4.13: Residuals of the velocity in the circularPipe case

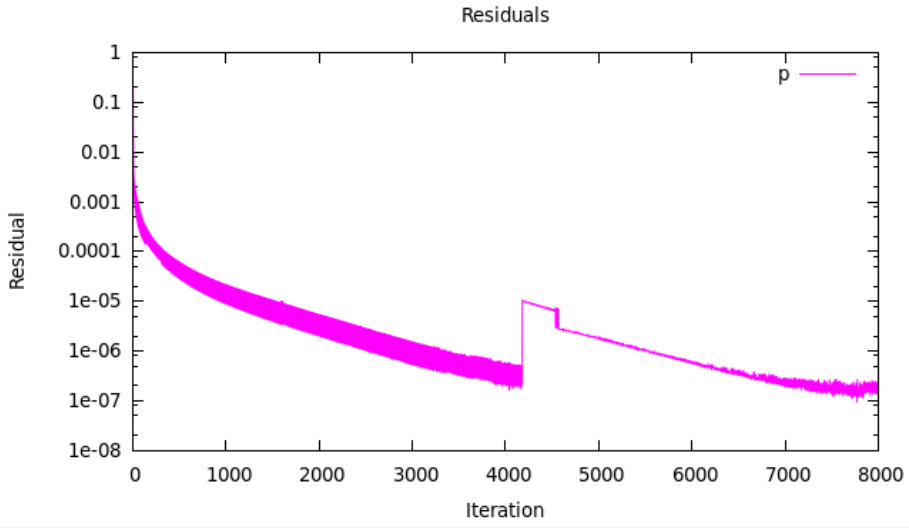
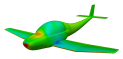


Figure 4.14: Residuals of the pressure in the circularPipe case



5. Aerodynamics of a 2D airfoil NACA 23012

5.0.12 Description of the case

The current chapter studies flow around a 2D airfoil. Although it encompasses the study of external flow, there are significant differences with Chapter 3, such as the fact that an airfoil is a streamlined body, and thus the behaviour of the flow around it changes substantially. Besides the physics of the problem, Chapter 5 includes new features as the introduction of turbulence models and the creation of a mesh without using blockMesh.

5.0.13 Hypotheses

- Incompressible flow
- Turbulent flow
- Newtonian flow
- Bidimensional flow ($\frac{\partial}{\partial z} = 0$)
- Negligible gravitatory effects
- Sea level conditions
- RAS turbulence modelling with wall functions

5.0.14 Physics of the problem

The problem deals with an airfoil NACA 23012 flying at a speed of $V = 45$ m/s at sea level. As the medium is air ($\nu = 1.5 \times 10^{-5}\ m^2/\text{s}$), the Reynolds number is

$$Re = \frac{Vc}{\nu} = \frac{45 \cdot 1}{1.5 \times 10^{-5}} = 3 \times 10^6$$

where c is the chord of the airfoil and is equal to 1 m. The chord is the imaginary straight line joining the leading and trailing edges. The problem statement is shown at Figure 5.1.

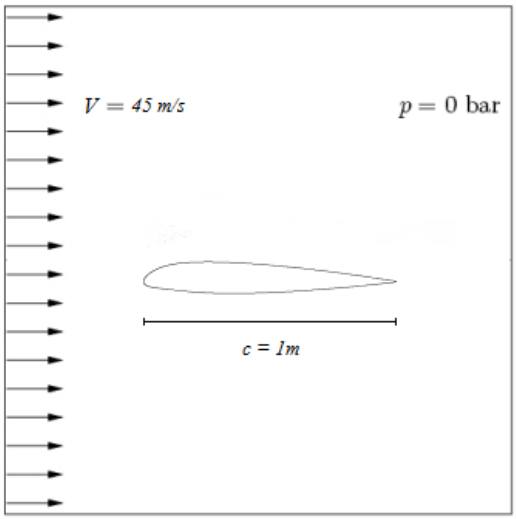


Figure 5.1: Airfoil NACA 23012 flying at 45 m/s and ambient pressure

The airfoil NACA 23012 has been chosen because of its high aerodynamic performance at low flying velocities. It presents a high maximum lift coefficient ($C_{l_{max}}$) and a high stall angle (α_{stall}). Its geometric characteristics are:

- Thickness: 12%
- Maximum thickness position: 30%
- Maximum chamber: 1.83%
- Maximum chamber position: 13%

The shape of the NACA 23012 is shown at Figure 5.2.

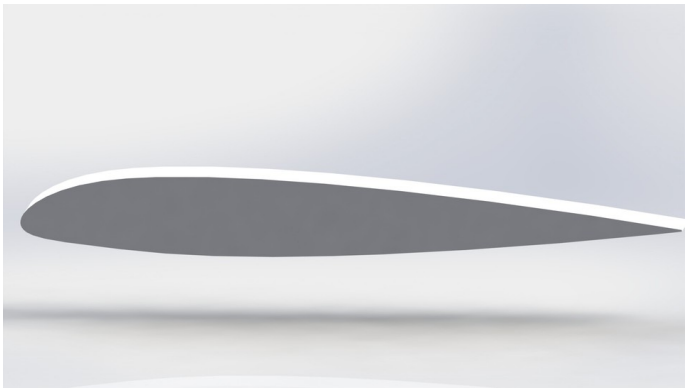


Figure 5.2: NACA 23012

In the current chapter there is no easy analytical solution to describe the behaviour of the fluid. However, it is necessary to keep in mind the main equations and dimensionless numbers involved in the problem:

The continuity equation,

$$\nabla \cdot \mathbf{U} = 0 \tag{5.1}$$

The momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{U} \tag{5.2}$$

The Reynolds number,

$$Re = \frac{c|\mathbf{U}|}{\nu} \tag{5.3}$$

The lift coefficient (dimensionless force perpendicular to the flow),

$$C_l = \frac{L}{\frac{1}{2}\rho|\mathbf{U}|^2 S} \tag{5.4}$$

The drag coefficient (dimensionless force parallel to the flow),

$$C_d = \frac{D}{\frac{1}{2}\rho|\mathbf{U}|^2 S} \tag{5.5}$$

The moment coefficient (dimensionless moment of the airfoil calculated at the aerodynamic center),

$$C_m = \frac{M}{\frac{1}{2}\rho|\mathbf{U}|^2 Sc} \tag{5.6}$$

For a low velocity flow and a surface with a given rugosity, C_l and C_d depend on the angle of attack and the Reynolds number:

$$C_l = f(Re, \alpha) \qquad \text{or} \qquad C_d = f(Re, \alpha)$$

The angle of attack (α) is the angle between the chord of the airfoil and the direction of the fluid velocity.

At low values of α, there is a pressure gradient on the upper surface of the airfoil but it is not strong enough to detach the boundary layer. The flow around the airfoil is smooth, the drag is low and the lift excellent. The relation between the lift coefficient and the angle of attack is approximately linear. When α increases, the pressure gradient grows. Normally, at an angle of attack between 15º and 20º, the flow is completely detached on the upper surface. When it happens, the airfoil stalls.

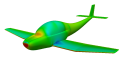
5.0.15 Pre-processing with $\alpha = 15^o$

The following codes contain the information to simulate the case with $Re = 3 \times 10^6$ using simpleFoam (steady-state solver for incompressible turbulent flow) and the Spalart-Allmaras turbulence model. As the current case includes new features and files, a general directory is going to be created with the name AirfoilCase and containing two files to create the mesh (without using blockMesh), as well as a subdirectoy containing the cases solved for different angles of attack (2DAirfoil). Within 2DAirfoil the names of the cases are going to be alpha0 ($\alpha = 0^o$), alpha5 ($\alpha = 5^o$), etc. In particular, the current pre-processing section contains the instructions to simulate the airfoil for $\alpha = 15^o$ (alpha15 case).

5.0.15.1 Mesh generation

As it happened with Chapter 3, the mesh is not going to be uniform. It is necessary to divide the mesh by regions, especially to provide a high refinement to the walls of the airfoil and downstream. However, in the current case, the geometry is more complex than a simple circular shape. Therefore, the mesh is going to be created with GNU Octave and Gmsh Mesh Generator. Then, it will be converted to *OpenFOAM*®.

The first step is to install these two softwares: GNU Octave is a high-level interpreted language, primarily intended for numerical computations. Gmsh Mesh Generator



contains 4 modules, for geometry description, meshing, solving and post-processing. They are both free software.

Secondly, it is necessary to create two gedit sheets. The first is going to be named *foilgmsh.m* and contains the code shown in the Appendix, Chapter A.

As it can be seen, this code was developed by an external author, who provided it selflessly. Before the instructions, it is explained how to use it and what parameters are necessary to define. The code can be entirely found in:

$$code - saturne.org/forum/old\_forums\_files/614676387/foilgmsh.m$$

Afterwards, a Script is going to be created to execute the previous code and introduce the output to Gmsh Mesh Generator. The results are going to be two new files, containing the created geometries and mesh. This mesh will be later converted to the $OpenFOAM^{®}$ format. The Script must be named *executeAirfoilMesher* and contains:

```
1  #!/bin/bash
2
3  archi="naca23012.dat" #File containing the coordinates of the airfoil (.dat
       extension)
4  alfa=15
5  yplus=100
6  eter="a"
7  Re=3000000
8  M=0.128
9  T0=300
10 N="[100, 40, 30, 30]"
11 bump=1
12 cd /home/<linuxUserName>/<Desktop>/AirfoilCase #If different, write how to access
       foilgmsh.m
13 octave --silent --eval "foilgmsh('$archi', $alfa, $yplus, '$eter', $Re, $M, $T0,
       '$N', $bump)" #Executes the foilgmsh function
14 gmsh -3 naca23012.dat.geo #Meshes with Gmsh Mesh Generator
```

So, once the programs have been installed and the gedit sheets containing the previous codes are ready:

1.- Make the Script executable as it was shown in Section 4.0.11.3

2.- Copy *foilgmsh.m* and *executeAirfoilMesher* within AirfoilCase

3.- Copy a file with the coordinates of the airfoil (in this case the NACA 23012) with the extension *.dat* named *naca23012.dat* within AirfoilCase. It can be easily found in Internet

4.- Access AirfoilCase and type in the terminal:

```
./executeAirfoilMesher
```

5.- Select one of the created files (*naca23012.dat.msh*) and copy it inside an empty directory named alpha15

6.- Copy a *system* folder from another case as it is necessary to convert the mesh to $OpenFOAM^{®}$ (it does not matter what time-control setting it contains; it will be later correctly defined)

7.- Access 2DAirfoil/alpha15

8.- Type:

```
gmshToFoam naca23012.dat.msh
```

9.- The *constant* directory appears

10.- Acces *constant/polyMesh/boundary*, change the name of the first patch (symmetry) to frontAndBack and define it as empty instead of patch (both, type and physicalType). Keep the second name (airfoil) but change its patch type to wall. Finally, change the third patch name (walls) to topAndBottom and its patch type to symmetryPlane instead of patch

11.- The mesh is ready

The mesh of the alpha15 case is:

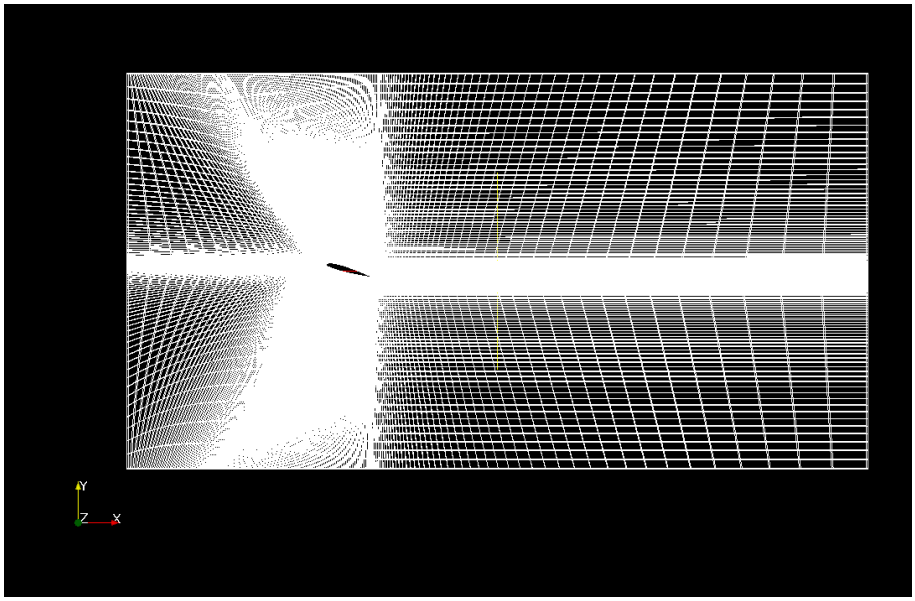


Figure 5.3: Global view of the mesh of the alpha15 case

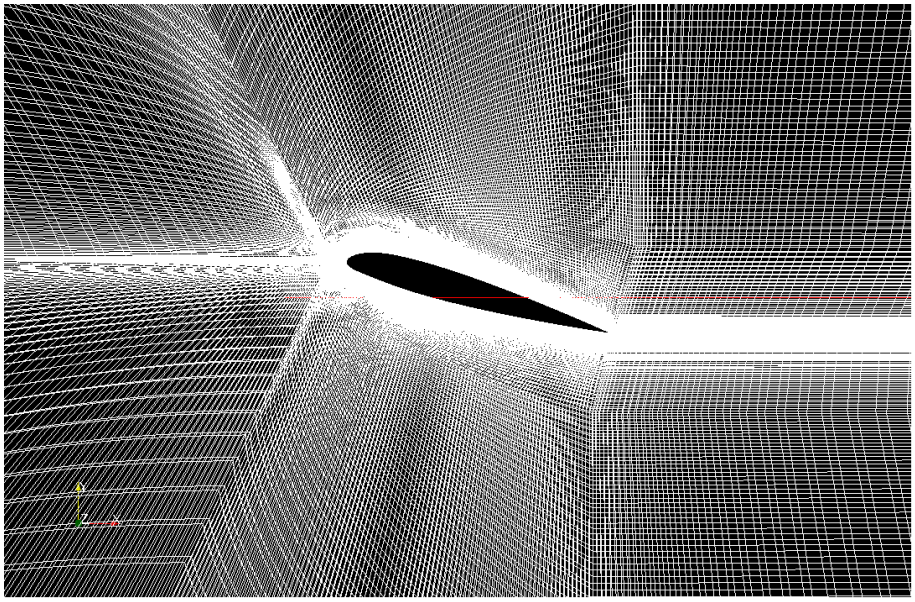


Figure 5.4: Airfoil shape in the mesh of the alpha15 case

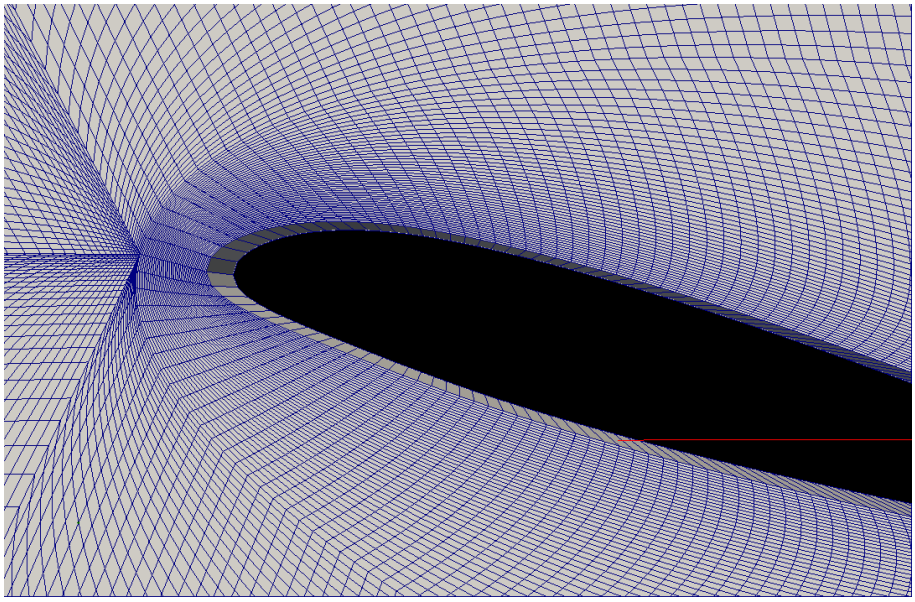


Figure 5.5: Detail of the mesh grading at the walls of the alpha15 case

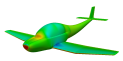
Advice:

> The degree of refinement of the mesh and its relation to the treatment of the turbulent boundary layer is discussed in Section 5.0.17.2. It determines the value written in the fifth line of *executeAirfoilMesher*

5.0.15.2 Boundary and initial conditions

Besides the *p* and *U* files, as the case is set turbulent, two new files are necessary to be created. Their names are *nut* and *nuTilda*. These dictionaries contain the boundary conditions of the parameters used to implement the Spalart-Allmaras turbulence model. Before introducing the user to their calculation, the boundary conditions for p and $\mathbf{U}$ are:

```
1  /*---------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      p;
```



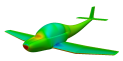
```
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 2 -2 0 0 0 0];
18
19  internalField   uniform 0;
20
21  boundaryField
22  {
23      inlet
24      {
25          type            freestreamPressure;
26      }
27
28      outlet
29      {
30          type            freestreamPressure;
31      }
32
33      airfoil
34      {
35          type            zeroGradient;
36      }
37
38      frontAndBack
39      {
40          type            empty;
41      }
42
43      topAndBottom
44      {
45          type            symmetryPlane;
46      }
47  }
48
49  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volVectorField;
13      object      U;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (45 0 0);
```

```
20
21  boundaryField
22  {
23      inlet
24      {
25          type            freestream;
26          freestreamValue uniform (45 0 0);
27      }
28
29      outlet
30      {
31          type            freestream;
32          freestreamValue uniform (45 0 0);
33      }
34
35      airfoil
36      {
37          type            fixedValue;
38          value           uniform (0 0 0);
39      }
40
41      frontAndBack
42      {
43          type            empty;
44      }
45
46      topAndBottom
47      {
48          type            symmetryPlane;
49      }
50  }
51
52  // ************************************************************************* //
```

The dictionaries for *nut* and *nuTilda* (they must be saved within *0* with these names) are:

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volScalarField;
13      object      nut;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 2 -1 0 0 0 0];
18
```



```
19  internalField   uniform 0.0386;
20
21  boundaryField
22  {
23      inlet
24      {
25          type            freestream;
26          freestreamValue uniform 0.0386;
27      }
28
29      outlet
30      {
31          type            freestream;
32          freestreamValue uniform 0.0386;
33      }
34
35      airfoil
36      {
37          type            nutUSpaldingWallFunction;
38          value           uniform 0;
39      }
40
41      frontAndBack
42      {
43          type            empty;
44      }
45
46      topAndBottom
47      {
48          type            symmetryPlane;
49      }
50  }
51
52  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volScalarField;
13      object      nuTilda;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 2 -1 0 0 0 0];
18
19  internalField   uniform 0.1929;
20
21  boundaryField
```

```
22  {
23      inlet
24      {
25          type            freestream;
26          freestreamValue uniform 0.1929;
27      }
28
29      outlet
30      {
31          type            freestream;
32          freestreamValue uniform 0.1929;
33      }
34
35      airfoil
36      {
37          type            fixedValue;
38          value           uniform 0;
39      }
40
41      frontAndBack
42      {
43          type            empty;
44      }
45
46      topAndBottom
47      {
48          type            symmetryPlane;
49      }
50  }
51
52  // ************************************************************************* //
```

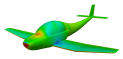
ν_t (*nut*) and $\tilde{\nu}_t$ (*nuTilda*) are parameters required to simulate the case with a turbulence model (as it was said, for this case it is used the Spalart-Allmaras one-equation model). Mathematically, they are determined as:

$$\tilde{\nu_t} = \sqrt{\frac{3}{2}}(|\mathbf{U}|Il) \tag{5.7}$$

where $|\mathbf{U}|$ is the mean flow velocity, I is the turbulence intensity and l is the turbulent length scale. They can be computed as:

$$I \approx 5\%$$

$$l \approx 0.07 \cdot c = 0.07$$



With these assumptions, $\tilde{\nu_t} = 0.1929$. Then, a convenient option is to set $\tilde{\nu_t} = 5\nu_t$ in the freestream. The model then provides fully turbulent results and any regions like boundary layers that contain shear become fully turbulent.

5.0.15.3 Physical properties

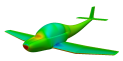
```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

rho             rho [ 1 -3 0 0 0 0 0 ] 1.225;

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

CrossPowerLawCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    m               m [ 0 0 1 0 0 0 0 ] 1;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}

BirdCarreauCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    k               k [ 0 0 1 0 0 0 0 ] 0;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
```

```
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "constant";
14      object      RASProperties;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  RASModel        SpalartAllmaras;
19
20  turbulence      on;
21
22  printCoeffs     on;
23
24  // ************************************************************************* //
```

5.0.15.4 Control

```
1   /*---------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      controlDict;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  application         simpleFoam;
19
20  startFrom           startTime;
21
22  startTime           0;
23
24  stopAt              endTime;
25
26  endTime             10000;
27
28  deltaT              1;
29
30  writeControl        timeStep;
31
32  writeInterval       100;
33
34  purgeWrite          0;
```

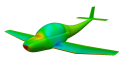


```
35
36  writeFormat          ascii;
37
38  writePrecision       7;
39
40  writeCompression     off;
41
42  timeFormat           general;
43
44  timePrecision        6;
45
46  runTimeModifiable    true;
47
48  functions
49  (
50  forces
51  {
52  type forces;
53  functionObjectLibs ("libforces.so");
54  patches (airfoil);
55  pName p;
56  UName U;
57  rhoName rhoInf;
58  rhoInf 1.225;
59  CofR (0.2704 -0.0012 4.0128);
60  outputControl timeStep;
61  outputInterval 100;
62  }
63  forceCoeffs
64  {
65  type forceCoeffs;
66  functionObjectLibs ("libforces.so");
67  patches (airfoil);
68  pName p;
69  UName U;
70  rhoName rhoInf;
71  rhoInf 1.225;
72  CofR (0.2704 -0.0012 4.0128);
73  liftDir (0 1 0);
74  dragDir (1 0 0);
75  pitchAxis (0 0 1);
76  magUInf 45; //
77  lRef 1; //
78  Aref 0.1; //
79  outputControl timeStep;
80  outputInterval 100;
81  }
82  );
83
84  // ************************************************************************* //
```

5.0.15.5 Discretization and linear-solver settings

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
```

```
3   | \\      /  F ield          | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration      | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      fvSchemes;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  ddtSchemes
19  {
20      default         steadyState;
21  }
22
23  gradSchemes
24  {
25      default         Gauss linear;
26      grad(p)         Gauss linear;
27      grad(U)         Gauss linear;
28  }
29
30  divSchemes
31  {
32      default           none;
33      div(phi,U)        Gauss linearUpwind grad(U);
34      div(phi,nuTilda)  Gauss linearUpwind grad(nuTilda);
35      div((nuEff*dev(T(grad(U))))) Gauss linear;
36  }
37  laplacianSchemes
38  {
39      default         none;
40      laplacian(nuEff,U) Gauss linear corrected;
41      laplacian((1|A(U)),p) Gauss linear corrected;
42      laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
43      laplacian(1,p) Gauss linear corrected;
44  }
45  interpolationSchemes
46  {
47      default        linear;
48      interpolate(U) linear;
49  }
50
51  snGradSchemes
52  {
53      default         corrected;
54  }
55
56  fluxRequired
57  {
58      default         no;
59      p               ;
```

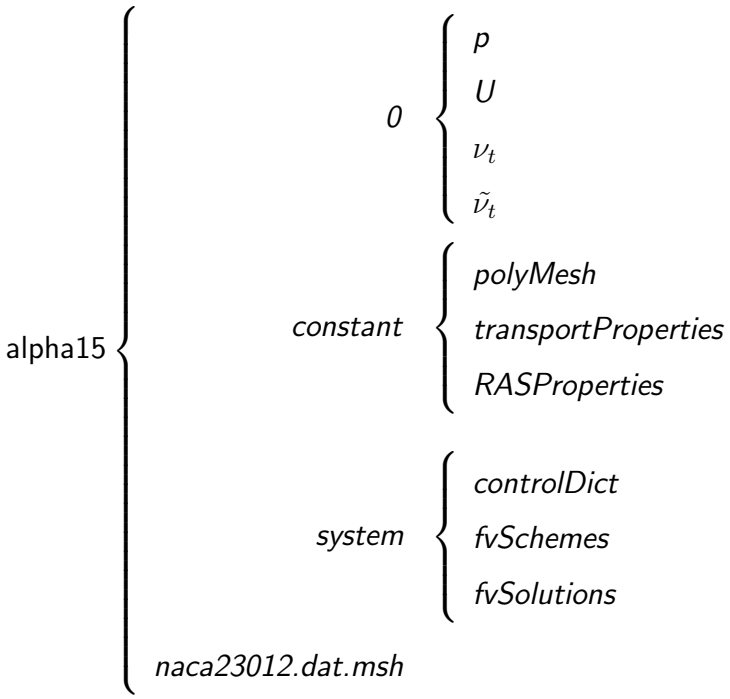


```
60  }
61
62  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      fvSolution;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  solvers
19  {
20      p
21      {
22          solver           GAMG;
23          tolerance        1e-06;
24          relTol           0.1;
25          smoother         GaussSeidel;
26          nPreSweeps       0;
27          nPostSweeps      2;
28          cacheAgglomeration true;
29          nCellsInCoarsestLevel 10;
30          agglomerator     faceAreaPair;
31          mergeLevels      1;
32      }
33
34      U
35      {
36          solver           smoothSolver;
37          smoother         GaussSeidel;
38          nSweeps          2;
39          tolerance        1e-08;
40          relTol           0.1;
41      }
42
43      nuTilda
44      {
45          solver           smoothSolver;
46          smoother         GaussSeidel;
47          nSweeps          2;
48          tolerance        1e-08;
49          relTol           0.1;
50      }
51  }
52
```

```
53  SIMPLE
54  {
55          nCorrectors                 1;
56          nNonOrthogonalCorrectors    2;
57          pRefCell                    0;
58          pRefValue                   0;
59          residualControl
60      {
61          p                           1e-5;
62          U                           1e-5;
63          nuTilda                     1e-5;
64      }
65  }
66       relaxationFactors
67      {
68
69       fields
70      {
71          p                           0.3;
72      }
73       equations
74      {
75          U                           0.7;
76          nuTilda                     0.7;
77      }
78  }
79
80  // ************************************************************************* //
```

At the end of the pre-processing, the structure of directories, subdirectories and files within alpha15 should be as follows:



5.0.16 Post-processing

5.0.16.1 Results of the simulation for $\alpha = 15^{o}$

The velocity field:

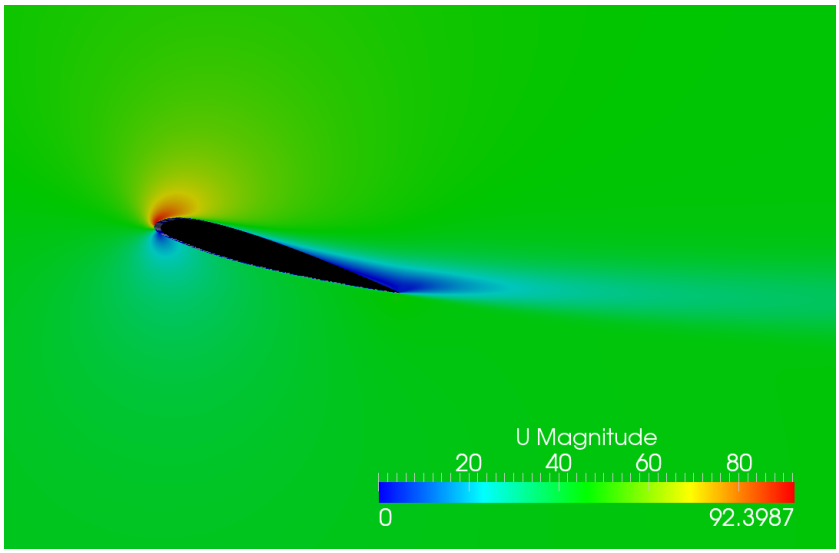


Figure 5.6: Velocity field around the NACA 23012 at $\alpha = 15^{o}$ (m/s)

The pressure field:

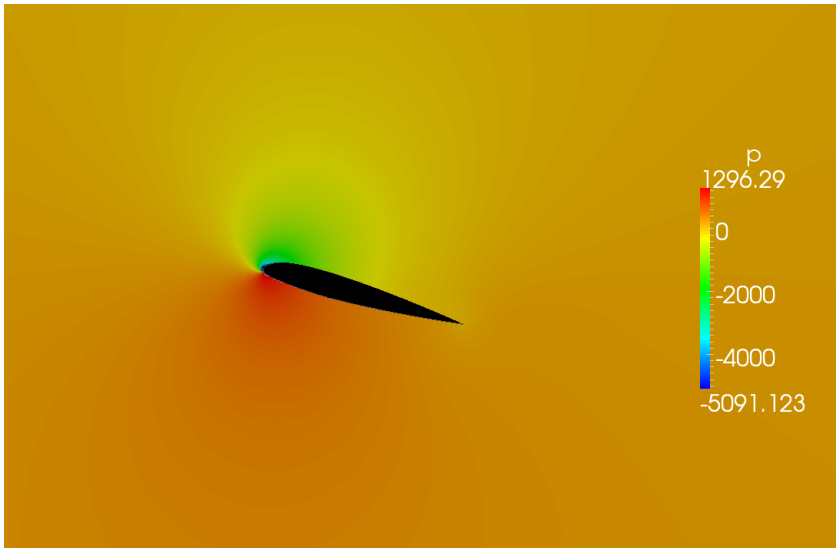


Figure 5.7: Pressure field around the NACA 23012 at $\alpha = 15^{o}$ (m^2/s^2)

The streamlines around the airfoil:

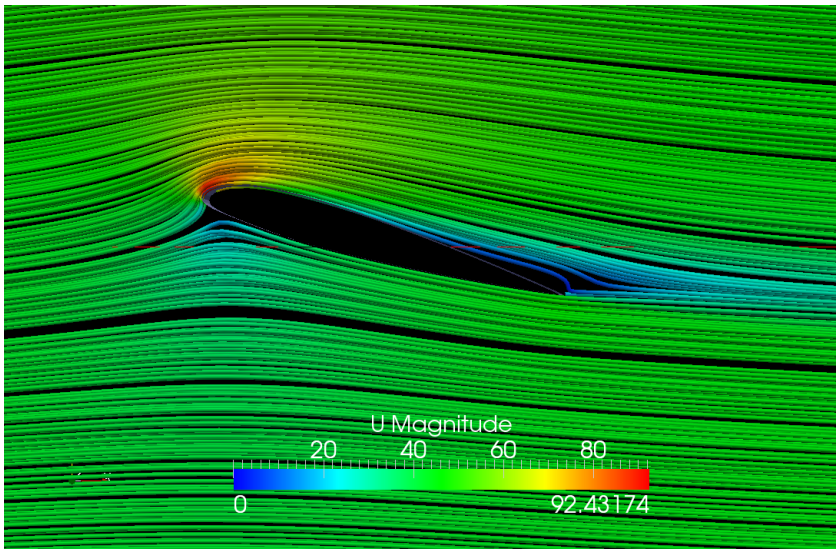


Figure 5.8: Streamlines around the NACA 23012 at $\alpha = 15^{o}$ (m/s)

It is possible to observe that at $\alpha = 15^{o}$ the flow has started to detach due to the strong pressure gradient on the upper surface. However, 15^{o} is not the maximum angle of attack (α_{stall}) because the fluid is not fully detached and therefore stall has not yet happened.

The vector field at the trailing edge (to appreciate the boundary layer detachment):

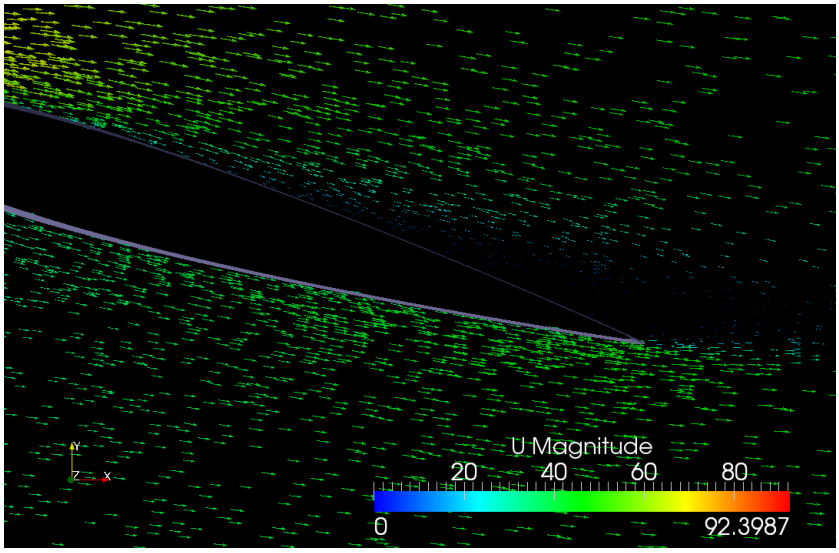


Figure 5.9: Velocity vectors around the NACA 23012 at $\alpha = 15^{o}$ (m/s)

The distribution of $\tilde{\nu}_t$ in the domain:

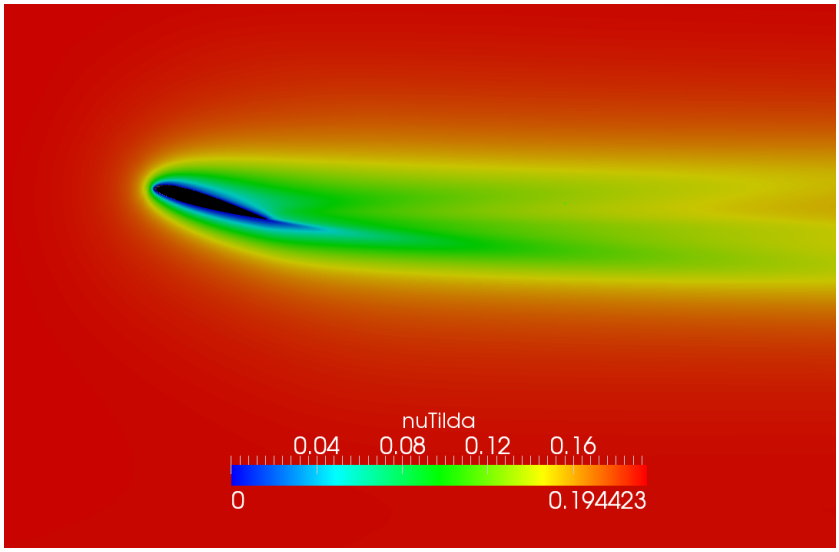


Figure 5.10: Distribution of $\tilde{\nu}_t$ in the domain of the alpha15 case

As it was explained in Chapter 3, the plot of the aerodynamic forces in front of the time gives information about the convergence of the case. In Figure 5.11 it is represented the lift coefficient:

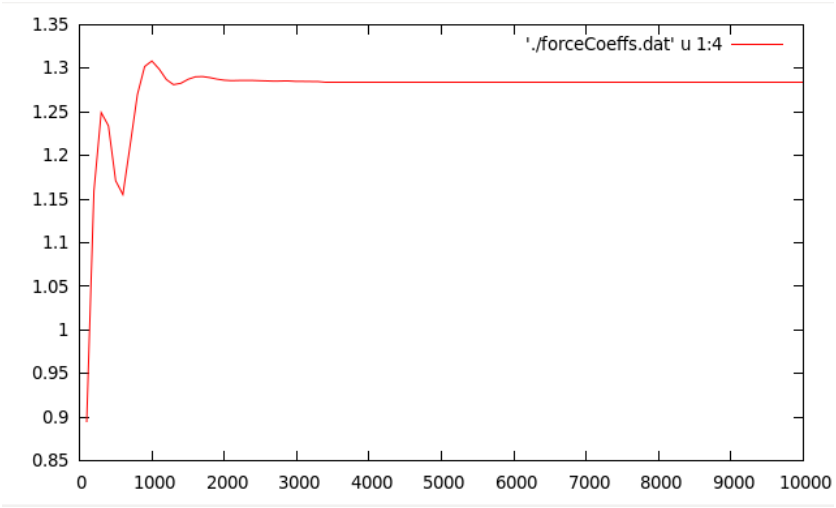


Figure 5.11: Evolution of the lift coefficient (ordinate axis) with the time (abscissa axis) in the alpha15 case

As it can be seen, the case has converged rapidly.

5.0.16.2 Results of the simulation for a range of α. Plot of the main aerodynamic curves

To simulate the NACA 23012 for different configurations (according to α), it is necessary to change this parameter before meshing. It has to be done in the parameter definition within *executeAirfoilMesher*. At the fourth line of the Script, the user can write the angle of attack (in degrees) at which the simulation will be carried out.

Caution:

> For $\alpha > 15^{o}$, as the flow detaches, there are unsteady phenomena. Thus, simpleFoam would not be the appropriate solver for these cases

Caution:

> Although the angle of attack changes from one case to the other, the velocity boundary conditions have to be set equal for all cases. This is because the flow velocity is not modified, but the geometric characteristics of the airfoil shape in the mesh

Some representative results of the behaviour of the NACA 23012 for different configurations are:

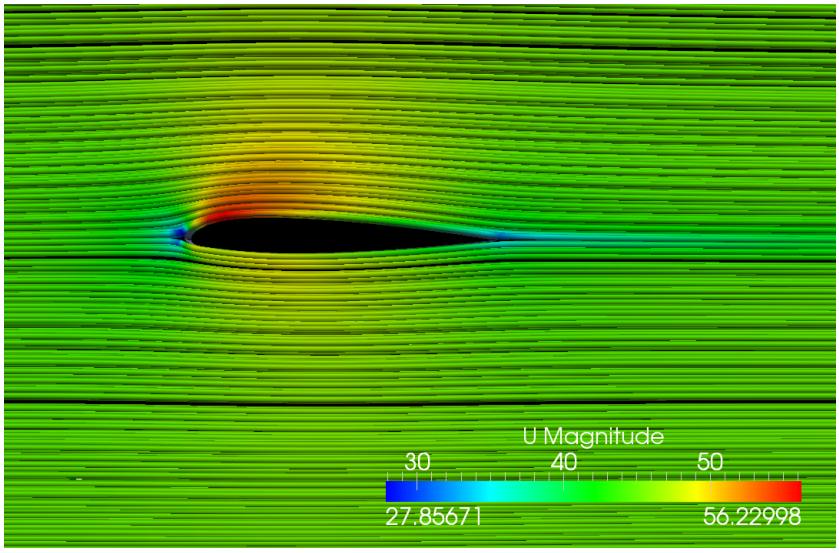


Figure 5.12: Behaviour of the flow around the NACA 23012 at $\alpha = 0^{o}$ (m/s)

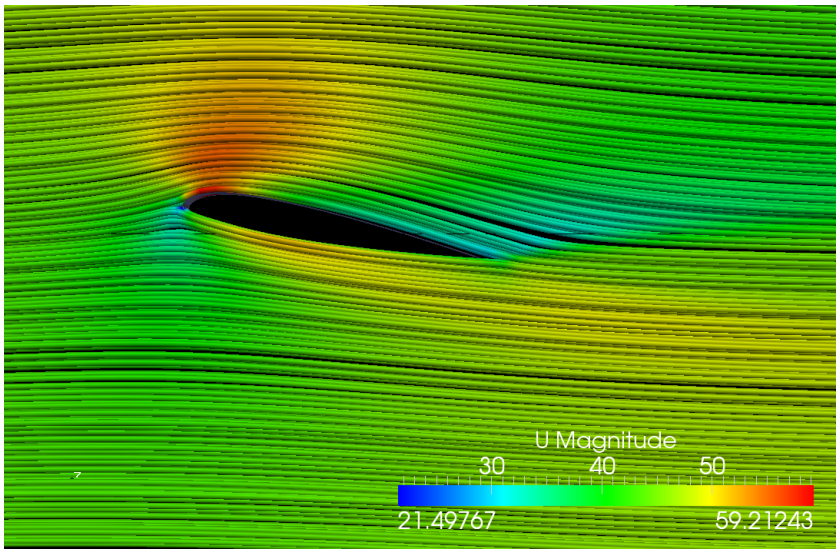


Figure 5.13: Behaviour of the flow around the NACA 23012 at $\alpha = 10^{o}$ (m/s)

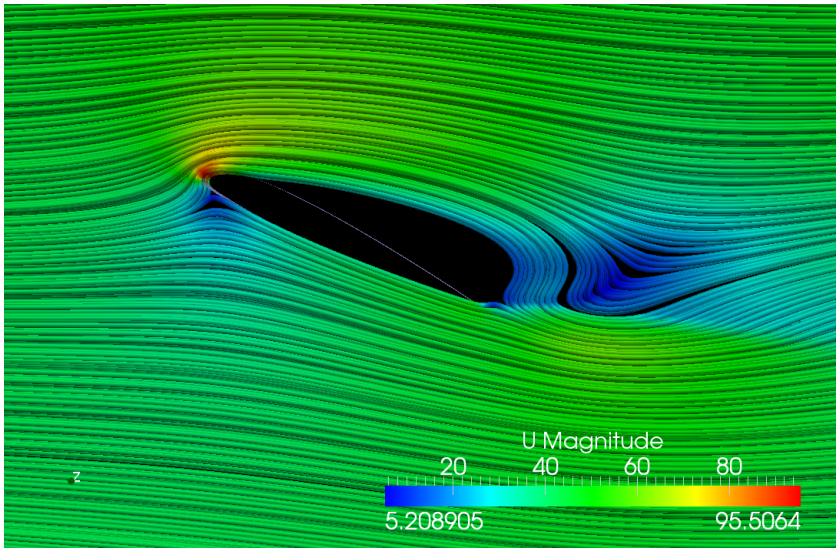


Figure 5.14: Behaviour of the flow around the NACA 23012 at $\alpha = 20^{o}$ (m/s)

By using these simulations, the main aerodynamic curves have been plot (in a range between -10^{o} and 15^{o} with intervals of 5^{o}). It can be done by copying the C_l and C_d obtained with the simulations and plot their relation using Gnuplot and the instructions shown in Section 3.6.3. The curves are:

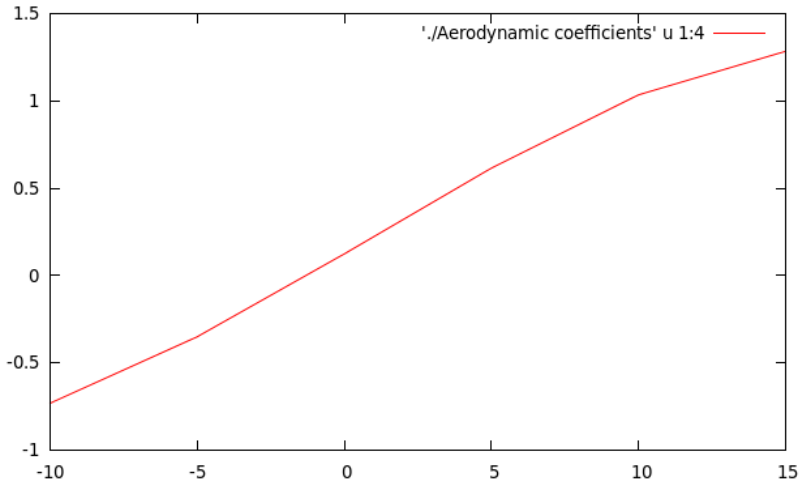


Figure 5.15: Relation between C_l (ordinate axis) and α (abscissa axis) for the NACA 23012

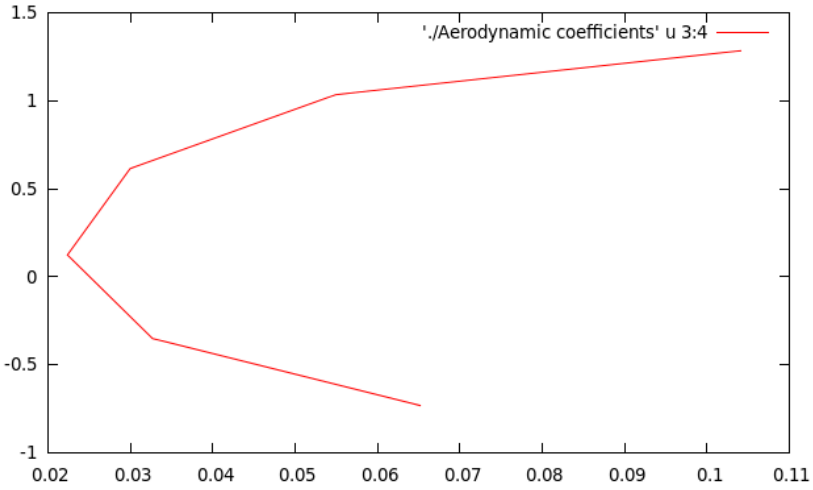


Figure 5.16: Relation between C_l (ordinate axis) and C_d (abscissa axis), polar curve, for the NACA 23012

It can be seen that for near zero angles of attack, the relation between C_l and α is linear. However, it is shown that this trend starts to stabilize when high values of $|\alpha|$ are achieved due to the boundary layer detachment.

5.0.17 Additional utilities

5.0.17.1 Mesh conversion

The user can generate meshes using other packages and convert them into the format that *OpenFOAM*® uses. This has been done in Section 5.0.15.1 to convert a mesh exported from Gmsh Mesh Generator by typing:

```
gmshToFoam
```

Furthermore, it is possible to convert meshes exported from other packages using instructions such as:

```
ansysToFoam
```

```
fluentMeshToFoam
```

5.0.17.2 Computation of y^+

y^+ is the dimensionless wall distance for a wall-bounded flow. It can be expressed as:

$$y^+ = \frac{u_\star y}{\nu} \tag{5.8}$$

where $u_\star$ is the friction velocity $\left(u_\star \equiv \sqrt{\frac{\tau_w}{\rho}}\right)$, y is the distance to the nearest wall and ν is the kinematic viscosity of the fluid. y^+ is commonly used in boundary layer theory and in defining the law of the wall.

y^+ plays a relevant role in the treatment of the boundary layer. The subdivision of the near-wall region in a turbulent boundary layer can be summarized as follows (*Fluent, 2005*):

- $y^+ < 5$: viscous sublayer region (velocity profile is assumed to be laminar and viscous stress dominates the wall shear)
- $5 < y^+ < 30$: buffer region (both viscous and turbulent shear dominate)
- $30 < y^+ < 300$: fully turbulent portion or log-law region (turbulent shear predominates)

In the alpha15 case, a wall function has been used to model the behaviour of the boundary layer. To use a wall function approach for a particular turbulence model with confidence, it is necessary to ensure that the y^+ values are within a certain range. As in the current case a RAS model has been implemented, it is necessary to guarantee that the first node of the mesh falls within the log-law region. It is to ensure that $30 < y^+ < 300$.

Caution:

> In addition to the concern about having a mesh with y^+ values that are too large, it is necessary to be aware that if the y^+ is too low then the first calculation point will be placed in the viscous sublayer region and the wall function will also be outside its validity

Caution:

> If an attached flow is modeled, then generally a wall function approach can be used. If flow separation is expected and the accurate prediction

of the separation point will have an impact on the results then it would be advisable to resolve the boundary layer with a finer mesh

OpenFOAM® presents a utility to calculate and report y^+ for all the wall patches. Note that as it depends on the local Reynolds number, it can only be obtained in the post-processing. Consequently only approximate values can be estimated when meshing.

As in the current case a RAS turbulence model has been used, the instruction (written in the terminal) to obtain y^+ is:

```
yPlusRAS
```

To view the results with ParaView it is necessary to select the `yPlus` box located within Volume Fields, choose the time at which the results will be consulted and select the wall patch (airfoil). The results:

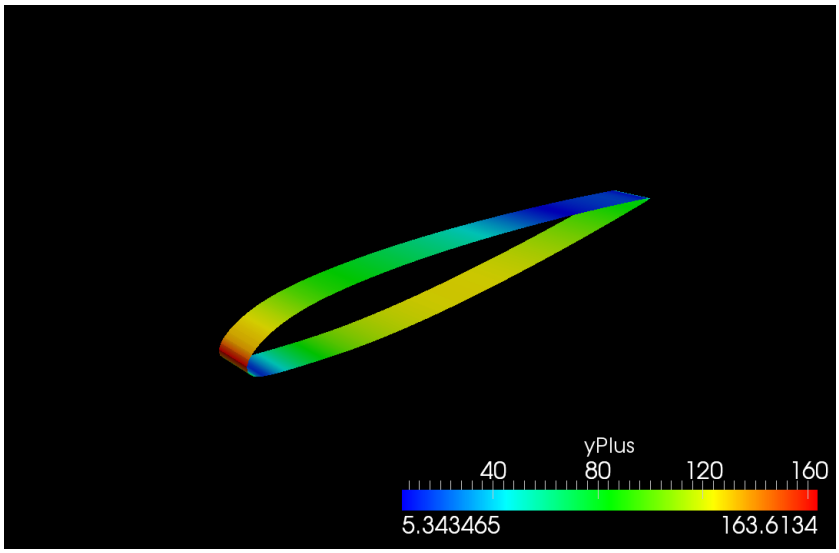


Figure 5.17: Distribution of y^+ at the wall of the airfoil in the alpha15 case at $t = 10000$ s

Note that there are some regions where y^+ is not within the required range. As it can be seen, it is basically located in the region of boundary layer detachment. In these regions the wall function approach would be out of its validity.

5.0.17.3 Isobars around a body

In this section it is explained how to create a contour plot across a plane. It will be used to plot the isobars around the NACA 23012 at $\alpha = 15^{o}$.

First of all the user has to launch ParaView, click on `internalMesh` and select the pressure field. Then, with the `alpha15.OpenFOAM` module highlighted, go to Filters -> Alphabetical -> Slice. The Slice filter allows the user to specify a cutting plane making the case truly two-dimensional. For the current case click on $\underline{Z}$ Normal so the plane will cut the domain in the adequate direction. Afterwards, select the `Slice1` module and go to Filters -> Alphabetical -> Contour.

To appreciate the isobars, select the `Slice1` and `Contour1` modules while keeping `alpha15.OpenFOAM` deselected. In the Properties menu of the `Contour` module, select the pressure field in the Contour By option and in the Isosurface panel click on New Range. Make sure that the `From` and `To` values coincide with the maximum and minimum of the legend and set the `Steps` option to 25. Finally, go to the Display menu and in the Color panel choose `Solid Color` in Color by instead of the pressure. Choose a color of your choice to represent the isobars. The results are then:

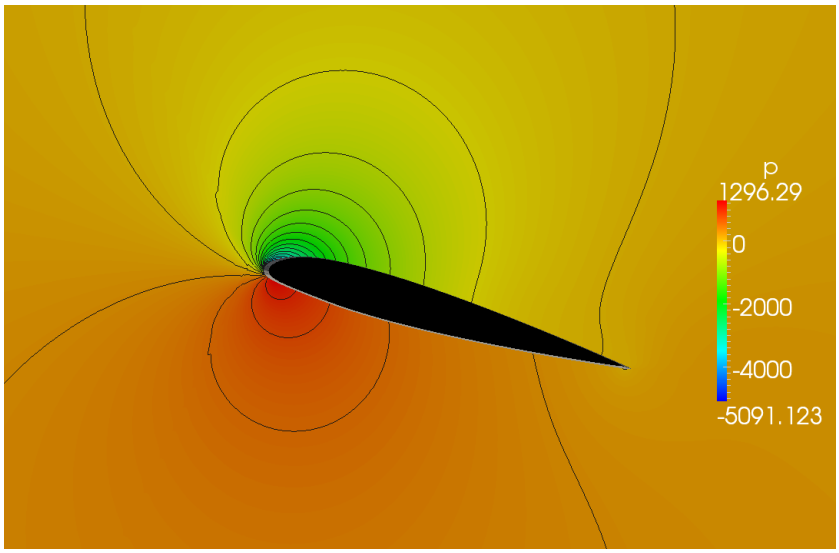


Figure 5.18: Isobars around the NACA 23012 at the alpha15 case at $t = 10000$ s (m^2/s^2)

5.0.17.4 Tube-like streamlines

In Section 2.6.2 it was shown how to plot the streamlines. Additionally, in the current section it is explained how to represent tube-like streamlines. It may add more

realism to the simulation or simply a different way of representing the streamlines. The differences between both configurations can be observed in the following figures:

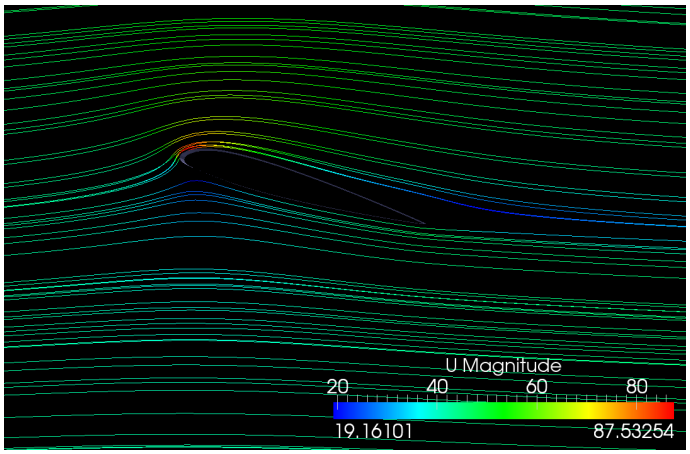


Figure 5.19: Streamlines around the NACA 23012 at the alpha15 case at $t = 10000$ s (m/s)

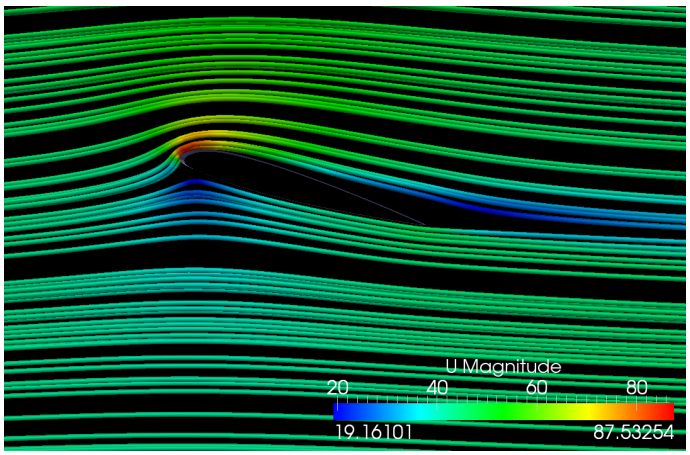


Figure 5.20: Tube-like streamlines around the NACA 23012 at the alpha15 case at $t = 10000$ s (m/s)

To obtain a configuration like Figure 5.20 it is first necessary to plot the streamlines as it was shown in Section 2.6.2. In the current case, Number of Points has been set to 3000 and Radius to 2. Once it is generated, it is necessary to select the module of the streamlines and go to Filters -> Alphabetical -> Tube. Finally, the thickness of the tube-like streamlines can be adjusted by changing the Radius option. In Figure 5.20 this parameter has been set to 0.01.

6. Fluid dynamics of a Very Light Aircraft

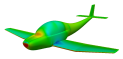
6.0.18 Description of the case

The last chapter of the guide describes a completely different case to be solved. While up to now only two-dimensional fluid mechanics problems have been solved, the current chapter encompasses the simulation of a real 3D body in a free stream. More specifically, this body is a very light aircraft flying at sea level conditions.

Unlike previous chapters, the mesh of the current case is going to be created using an external surface generated with a 3D CAD design software. Nevertheless, the user can follow the tutorial using a geometry created by himself. As the main steps are common for generic geometries, it will only be necessary to take into consideration particular instructions which may depend on the specifications of each case. It implies that standard bodies in a free stream (a sphere, an automobile, etc.) will be suitable to be simulated according to the instructions and methods shown in Chapter 6.

6.0.19 Hypotheses

- Incompressible flow
- Turbulent flow
- Newtonian flow
- Negligible gravitatory effects
- Sea level conditions
- RAS turbulence modelling with wall functions
- Non-static components of the aircraft are not included (as for instance the propeller), as well as the landing gear



- Aircraft flying at $\alpha = 0$

6.0.20 Physics of the problem

The problem encompasses a very light aircraft flying at a speed of $V = 45$ m/s at sea level. As the medium is air ($\nu = 1.5 \times 10^{-5}\ m^2/\text{s}$), the Reynolds number is

$$Re = \frac{Vc}{\nu} = \frac{45 \cdot 1.276}{1.5 \times 10^{-5}} = 38.28 \times 10^5$$

where c is the mean aerodynamic chord of the wings and is equal to 1.276 m. The mean aerodynamic chord is the chord of a rectangular wing which has the same area, aerodynamic force and position of the pressure center for a given angle of attack as the original. The very light aircraft that it is going to be used for the simulation does not belong to any real aeronautics company; it was designed by the author of the guide during a course taught at the university (and thus it might contain flaws).

The aircraft is shown at Figure 6.1.

Figure 6.1: Aircraft used in the simulation of the *Very Light Aircraft* case

The aircraft measures and geometrical properties are:

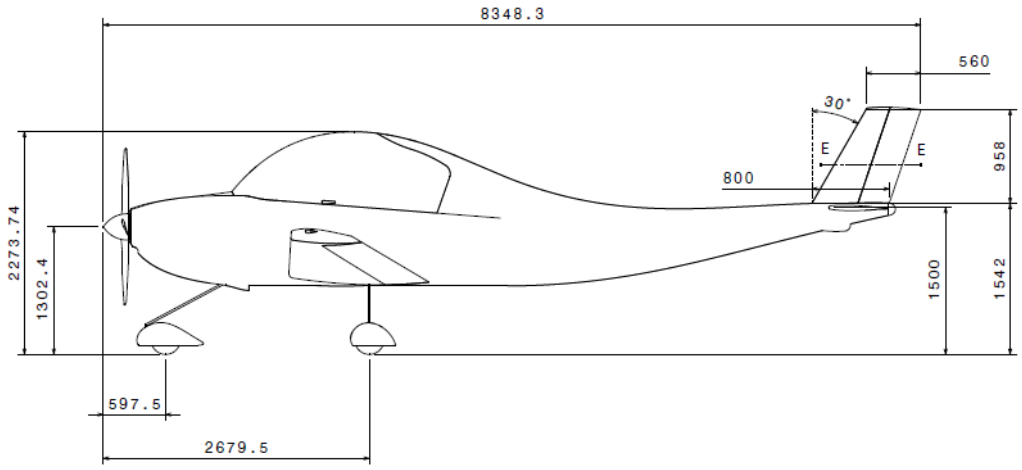


Figure 6.2: Measures of the aircraft used in the simulation of the *Very Light Aircraft* case

The problem statement is shown at Figure 6.3.

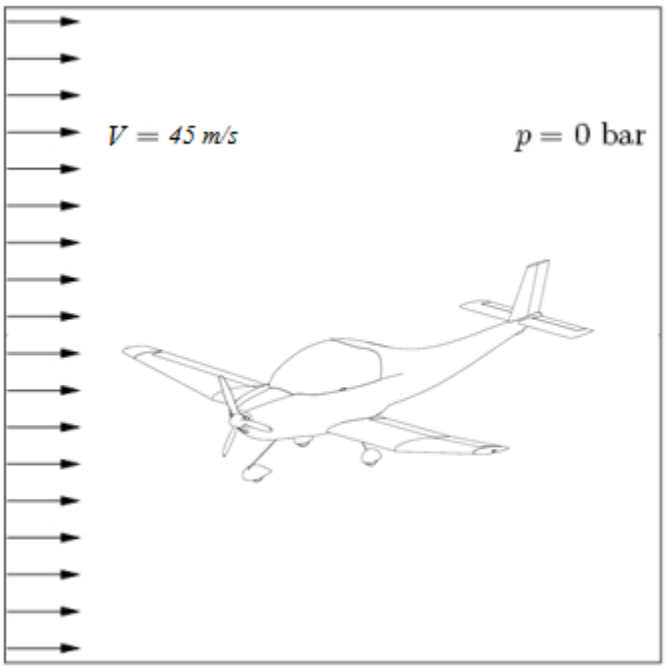
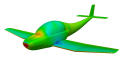


Figure 6.3: Very light aircraft flying at 45 m/s and ambient pressure

In the current chapter there is no easy analytical solution to describe the behaviour of the fluid. However, it is necessary to keep in mind the main equations involved in the problem:



The continuity equation,

$$\nabla \cdot \mathbf{U} = 0 \tag{6.1}$$

The momentum equation,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{U} \tag{6.2}$$

As it can be seen at Figure 6.2, the aircraft presents a length of 8.3 m. Unlike previous cases, it cannot be directly meshed and treated because of its large size (it would be necessary to create an enormous number of cells). This is the reason why the geometry will be rescaled 100 times. It can be understood as if the analysis would be done to a prototype introduced into a wind tunnel.

Consequently, it is necessary to do a dimensional analysis to find out the dimensionless numbers involved in the problem. It ensures that the flow conditions are consistent to guarantee that the results of the 100 times rescaled aircraft are the same as if the real aircraft would be flying in the sky.

The dimensionless variables of Equations 6.1 and 6.2 are:

$$\tilde{\mathbf{x}} = \frac{\mathbf{x}}{c}$$

$$\tilde{\mathbf{U}} = \frac{\mathbf{U}}{V}$$

$$\tilde{p} = \frac{p}{\rho V^2}$$

$$\tilde{t} = \frac{t}{c/V}$$

By introducing them, the dimensionless equations of mass and momentum are:

$$\tilde{\nabla} \cdot \tilde{\mathbf{U}} = 0 \tag{6.3}$$

$$\frac{\partial \tilde{\mathbf{U}}}{\partial \tilde{t}} + \tilde{\mathbf{U}} \cdot \tilde{\nabla}\tilde{\mathbf{U}} = -\tilde{\nabla}\tilde{p} + \frac{\mu}{\rho c V}\tilde{\nabla}^2\tilde{\mathbf{U}}$$

which can be rewritten as:

$$\frac{\partial \tilde{\mathbf{U}}}{\partial \tilde{t}} + \tilde{\mathbf{U}} \cdot \tilde{\nabla}\tilde{\mathbf{U}} = -\tilde{\nabla}\tilde{p} + \frac{1}{Re}\tilde{\nabla}^2\tilde{\mathbf{U}} \tag{6.4}$$

It can be seen that the only dimensionless number involved in the problem is the

Reynolds number. Therefore, maintaining Re (comparing the real aircraft flying in the sky with the prototype analysed in the wind tunnel), the behaviour of the flow around them may be comparable and the case preserves its coherence. As a consequence, if the mean aerodynamic chord has been reduced 100 times, the kinematic viscosity will be also reduced this quantity:

$$Re = \frac{Vc}{\nu} = \frac{45 \cdot 0.01276}{1.5 \times 10^{-7}} = 38.28 \times 10^5$$

6.0.21 Pre-processing

The following codes contain the information to simulate the case with $Re = 38.28 \times 10^5$ using simpleFoam and the SST k-w turbulence model. The main differences with previous cases come from the fact that the *constant* and *system* directories will include new features. Therein it will be contained the dictionaries and files required to treat and mesh the geometry.

It is highly recommendable to follow the current tutorial bearing in mind the structure of directories that the case is going to contain (shown at the end of Section 6.0.21.6), as the case setting changes significantly. The scheme of directories is similar as the one used in the official $OpenFOAM^{®}$ motorBike case (located within *incompressible/simpleFoam*). It may help the user to figure up the case resolution, before and whilst running it.

The case directory is named aircraft and will be located within FoamCases.

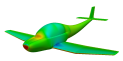
6.0.21.1 Mesh generation

Handling surface meshes

First of all, create empty *constant* and *system* directories within aircraft. In *constant*, there will be two new directories, *polyMesh* and *triSurface*, this second containing the case geometry saved with the .stl extension. In the current tutorial, the file with the geometry is going to be named aircraft.stl.

STL (STereoLithography) is a file format native to the stereolithography CAD software created by 3D Systems. This file format is supported by many other software packages; it is widely used for rapid prototyping and computer-aided manufacturing. STL files describe only the surface geometry of a three-dimensional object without any representation of color, texture or other common CAD model attributes.

Next, as it was done in Chapter 5, a *dummy controlDict* file is needed to be included.



Caution:

For coherence with further explanations, set deltaT and writeInterval within *controlDict* equal to 1

Now the user has to include a new file: *surfaceFeatureExtractDict*. It is contained within *system* and is used to extract feature edges from tri-surfaces. The file includes:

```
1  /*--------------------------------* C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.0                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      surfaceFeatureExtractDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 aircraft.stl
18 {
19     // How to obtain raw features (extractFromFile || extractFromSurface)
20     extractionMethod    extractFromSurface;
21
22     extractFromSurfaceCoeffs
23     {
24         // Mark edges whose adjacent surface normals are at an angle less
25         // than includedAngle as features
26         // - 0  : selects no edges
27         // - 180: selects all edges
28         includedAngle   120;
29     }
30
31     // Write options
32
33         // Write features to obj format for postprocessing
34         writeObj                yes;
35 }
36 // ************************************************************************* //
```

Advice:

At line 17 the name has to match the one used for the file located within *constant/triSurface*

Next, the user has to execute the dictionary to extract the features. Type within the case:

```
surfaceFeatureExtract
```

Note that a new directory and a new file have been generated within *constant* and *constant/triSurface* respectively.

At this moment the user can launch ParaView to view the aircraft shape, opening the .stl file contained in *constant/triSurface*.

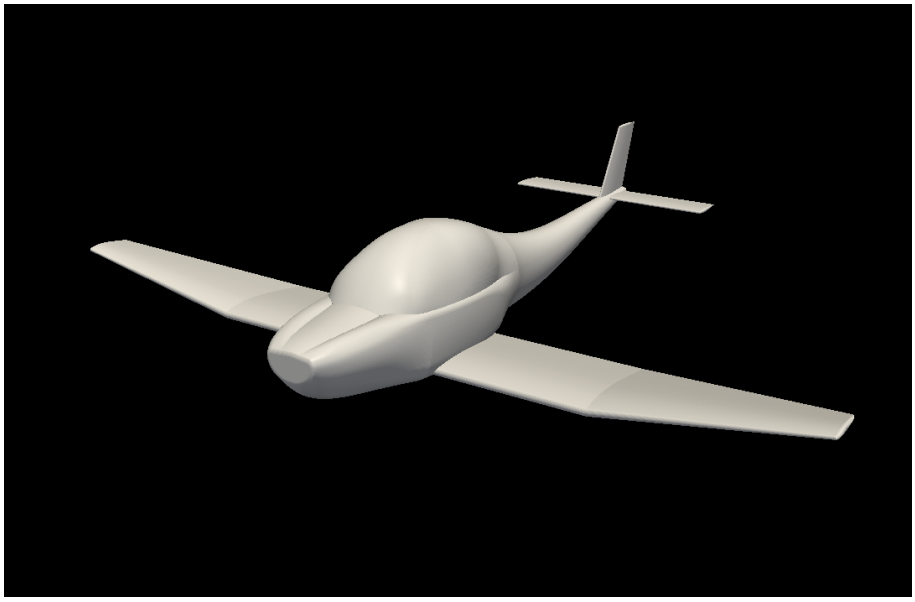


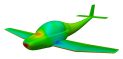
Figure 6.4: Aircraft' STL surface used in the simulation of the *Very Light Aircraft* case

Generating a background mesh

Before meshing the aircraft itself, it is first necessary to create a background mesh. It can be simply done using blockMesh and *blockMeshDict*. The aim is to create a big rectangular prism encompassing the whole aircraft. This prism will become the fluid domain of the case. The inside of the aircraft will be removed using further instructions transforming the aircraft into a real obstacle for the flow.

Caution:

> As it is necessary to analyze the flow downstream more than the flow upstream, the distance from the inlet patch of the prism to the aircraft's cabin will be shorter than the distance between the outlet patch and the aircraft's tail



Advice:

> Try to generate the cells of the background mesh with an aspect ratio as close to 1 as possible

So, to create an adequate mesh for the aircraft, copy the following *blockMeshDict* code within *constant/polyMesh* and type:

`blockMesh`

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.0                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      blockMeshDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 convertToMeters 0.01;
18
19 vertices
20 (
21     (-30 -8 -10)
22     ( 10 -8 -10)
23     ( 10  8 -10)
24     (-30  8 -10)
25     (-30 -8  10)
26     ( 10 -8  10)
27     ( 10  8  10)
28     (-30  8  10)
29
30 );
31
32 blocks
33 (
34     hex (0 1 2 3 4 5 6 7) (32 13 16) simpleGrading (1 1 1)
35 );
36
37 edges
38 (
39 );
40
41 boundary
42 (
43     inlet
44     {
```

```
45          type patch;
46      faces
47      (
48          (1 2 6 5)
49      );
50        }
51
52        outlet
53        {
54          type patch;
55      faces
56      (
57          (0 4 7 3)
58      );
59        }
60
61        bottom
62        {
63          type slip;
64      faces
65      (
66          (0 3 2 1)
67      );
68        }
69
70        top
71        {
72          type slip;
73      faces
74      (
75                  (4 5 6 7)
76      );
77        }
78
79        frontAndBack
80        {
81          type slip;
82      faces
83      (
84                  (0 1 5 4)
85                  (2 3 7 6)
86      );
87        }
88
89  );
90
91  mergePatchPairs
92  (
93  );
94
95  // ************************************************************************* //
```

Figure 6.5 shows the result of combining the mesh generated with the initial STL surface:

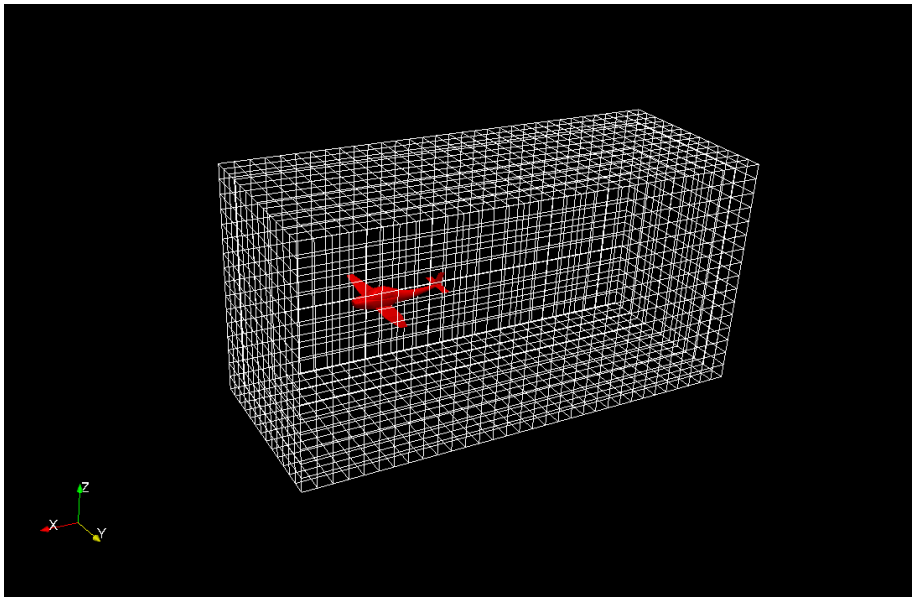


Figure 6.5: Aircraft' STL surface contained within the mesh generated with blockMesh in the *Very Light Aircraft* case

Advice:

> To obtain the results shown in Figure 6.5 it is necessary to include *dummy* files of *fvScheme* and *fvSolution*

Using snappyHexMesh

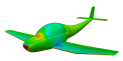
To mesh, the snappyHexMesh application is going to be used, being controlled by the *snappyHexMeshDict* file located in *system*. The main parts of the dictionary are:

- 3 switches to control the individual meshing processes
- A geometry subdictionary for the surface geometry used in the meshing
- 3 subdictionaries, one for each meshing process
- A subdictionary for the control of the quality criteria

The *snappyHexMeshDict* file that it is going to be used for the meshing of the aircraft is:

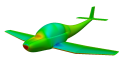
```
1 /*---------------------------------*- C++ -*----------------------------------*\
2 | =========                 |                                                 |
3 | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
```

```
4   |  \\    /   O peration     | Version:  2.2.0                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      snappyHexMeshDict;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  // Which of the steps to run
18  castellatedMesh true;
19  snap            false;
20  addLayers       false;
21
22  // Geometry. Definition of all surfaces. All surfaces are of class
23  // searchableSurface.
24  // Surfaces are used
25  // - to specify refinement for any mesh cell intersecting it
26  // - to specify refinement for any mesh cell inside/outside/near
27  // - to 'snap' the mesh boundary to the surface
28  geometry
29  {
30      aircraft.stl //STL filename where all the regions are added
31      {
32          type triSurfaceMesh;
33
34          regions
35    {
36      patch0              //Named region in the STL file
37      {
38        name aircraftPatch;   //User-defined patch name. If not provided will be
              <name>_<region>
39      }
40    }
41      }
42
43      refinementBox //Geometry to refine. Entities: Box, Cylinder, Sphere, Plane
44      {
45          type searchableBox;
46          min (-0.3 -0.06 -0.04);
47          max (0 0.06 0.04);
48      }
49  };
50
51  // Settings for the castellatedMesh generation.
52  castellatedMeshControls
53  {
54
55      // Refinement parameters
56      // ~~~~~~~~~~~~~~~~~~~~~
57
58      // If local number of cells is >= maxLocalCells on any processor
59      // switches from from refinement followed by balancing
```



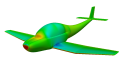
```
60      // (current method) to (weighted) balancing before refinement.
61      maxLocalCells 1500000;
62
63      // Overall cell limit (approximately). Refinement will stop immediately
64      // upon reaching this number so a refinement level might not complete.
65      // Note that this is the number of cells before removing the part which
66      // is not 'visible' from the keepPoint. The final number of cells might
67      // actually be a lot less.
68      maxGlobalCells 2000000;
69
70      // The surface refinement loop might spend lots of iterations refining just a
71      // few cells. This setting will cause refinement to stop if <= minimumRefine
72      // are selected for refinement. Note: it will at least do one iteration
73      // (unless the number of cells to refine is 0)
74      minRefinementCells 0;
75
76      // Allow a certain level of imbalance during refining
77      // (since balancing is quite expensive)
78      // Expressed as fraction of perfect balance (= overall number of cells /
79      // nProcs). 0=balance always.
80      maxLoadUnbalance 0.1;
81
82      // Number of buffer layers between different levels.
83      // 1 means normal 2:1 refinement restriction, larger means slower
84      // refinement.
85      nCellsBetweenLevels 3;
86
87
88      // Explicit feature edge refinement
89      // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
90
91      // Specifies a level for any cell intersected by explicitly provided
92      // edges.
93      // This is a featureEdgeMesh, read from constant/triSurface for now.
94      // Specify 'levels' in the same way as the 'distance' mode in the
95      // refinementRegions (see below). The old specification
96      //      level   2;
97      // is equivalent to
98      //      levels  ((0 2));
99
100     features //Disabled because the refinement will be surface-based (as interal
            aircraft cells are removed)
101
102     (
103         //{
104         //    file "aircraft.eMesh";
105         //    levels ((4 4));
106         //}
107     );
108
109     // Surface based refinement
110     // ~~~~~~~~~~~~~~~~~~~~~~~~
111
112     // Specifies two levels for every surface. The first is the minimum level,
113     // every cell intersecting a surface gets refined up to the minimum level.
114     // The second level is the maximum level. Cells that 'see' multiple
115     // intersections where the intersections make an
```

```
116     // angle > resolveFeatureAngle get refined up to the maximum level.
117
118     refinementSurfaces
119     {
120
121       aircraft.stl    //STL filename where all the regions are added
122         {
123     level (6 6);
124     regions
125     {
126       /*zone0 //Named region in the STL file
127       {
128                 // Surface-wise min and max refinement level
129         level (2 2);
130                     // Optional specification of patch type (default is wall). No
131                     // constraint types (cyclic, symmetry) etc. are allowed.
132                 patchInfo
133                 {
134                     type patch;
135                     inGroups (meshedPatches);
136                     }
137       }*/
138     }
139   }
140     }
141
142     // Feature angle:
143     // - used if min and max refinement level of a surface differ
144     // - used if feature snapping (see snapControls below) is used
145     resolveFeatureAngle 30;
146
147
148     // Region-wise refinement
149     // ~~~~~~~~~~~~~~~~~~~~~~
150
151     // Specifies refinement level for cells in relation to a surface. One of
152     // three modes
153     // - distance. 'levels' specifies per distance to the surface the
154     //   wanted refinement level. The distances need to be specified in
155     //   increasing order.
156     // - inside. 'levels' is only one entry and only the level is used. All
157     //   cells inside the surface get refined up to the level. The surface
158     //   needs to be closed for this to be possible.
159     // - outside. Same but cells outside.
160
161     refinementRegions
162     {
163         refinementBox
164         {
165             mode inside;
166             levels ((1E15 3)); //(1E15) not relevant.
167         }
168
169     //aircraft.stl
170         //{
171         //    mode distance;
172         //    levels (0.008 5);
```



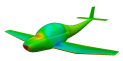
```
173             //}
174         }
175
176         // Mesh selection
177         // ~~~~~~~~~~~~~~~
178
179         // After refinement patches get added for all refinementSurfaces and
180         // all cells intersecting the surfaces get put into these patches. The
181         // section reachable from the locationInMesh is kept.
182         // NOTE: This point should never be on a face, always inside a cell, even
183         // after refinement.
184         locationInMesh (-0.051 -0.04 -0.008);
185
186         // Whether any faceZones (as specified in the refinementSurfaces)
187         // are only on the boundary of corresponding cellZones or also allow
188         // free-standing zone faces. Not used if there are no faceZones.
189         allowFreeStandingZoneFaces true;
190     }
191
192     // Settings for the snapping.
193     snapControls
194     {
195         // Number of patch smoothing iterations before finding correspondence
196         // to surface
197         nSmoothPatch 3;
198
199         // Maximum relative distance for points to be attracted by surface.
200         // True distance is this factor times local maximum edge length.
201         // Note: changed(corrected) w.r.t 17x! (17x used 2* tolerance)
202         tolerance 1.0;
203
204         // Number of mesh displacement relaxation iterations.
205         nSolveIter 30;
206
207         // Maximum number of snapping relaxation iterations. Should stop
208         // before upon reaching a correct mesh.
209         nRelaxIter 5;
210
211         // Feature snapping
212
213             // Number of feature edge snapping iterations.
214             // Leave out altogether to disable.
215             nFeatureSnapIter 10;
216
217             // Detect (geometric only) features by sampling the surface
218             // (default=false).
219             implicitFeatureSnap false;
220
221             // Use castellatedMeshControls::features (default = true)
222             explicitFeatureSnap true;
223
224             // Detect features between multiple surfaces
225             // (only for explicitFeatureSnap, default = false)
226             multiRegionFeatureSnap false;
227     }
228
229     // Settings for the layer addition.
```

```
230 addLayersControls
231 {
232     // Are the thickness parameters below relative to the undistorted
233     // size of the refined cell outside layer (true) or absolute sizes (false).
234     relativeSizes true;
235
236     // Layer thickness specification. This can be specified in one of four ways
237     // - expansionRatio and finalLayerThickness (cell nearest internal mesh)
238     // - expansionRatio and firstLayerThickness (cell on surface)
239     // - overall thickness and firstLayerThickness
240     // - overall thickness and finalLayerThickness
241
242         // Expansion factor for layer mesh
243         expansionRatio 2;
244
245         // Wanted thickness of the layer furthest away from the wall.
246         // If relativeSizes this is relative to undistorted size of cell
247         // outside layer.
248         finalLayerThickness 0.4;
249
250         // Wanted thickness of the layer next to the wall.
251         // If relativeSizes this is relative to undistorted size of cell
252         // outside layer.
253         //firstLayerThickness 0.3;
254
255         // Wanted overall thickness of layers.
256         // If relativeSizes this is relative to undistorted size of cell
257         // outside layer.
258         //thickness 0.5
259
260
261     // Minimum overall thickness of total layers. If for any reason layer
262     // cannot be above minThickness do not add layer.
263     // If relativeSizes this is relative to undistorted size of cell
264     // outside layer..
265     minThickness 0.2;
266
267
268     // Per final patch (so not geometry!) the layer information
269     // Note: This behaviour changed after 21x. Any non-mentioned patches
270     //       now slide unless:
271     //          - nSurfaceLayers is explicitly mentioned to be 0.
272     //          - angle to nearest surface < slipFeatureAngle (see below)
273     layers
274     {
275         aircraftPatch
276         {
277             nSurfaceLayers 2;
278
279         }
280         maxY
281         {
282             nSurfaceLayers 2;
283             // Per patch layer data
284             expansionRatio      2;
285             finalLayerThickness 0.4;
286             minThickness        0.2;
```



```
287         }
288
289         // Disable any mesh shrinking and layer addition on any point of
290         // a patch by setting nSurfaceLayers to 0
291         frozenPatches
292         {
293             nSurfaceLayers 0;
294         }
295     }
296
297     // If points get not extruded do nGrow layers of connected faces that are
298     // also not grown. This helps convergence of the layer addition process
299     // close to features.
300     // Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
301     nGrow 0;
302
303     // Advanced settings
304
305     // When not to extrude surface. 0 is flat surface, 90 is when two faces
306     // are perpendicular
307     featureAngle 60;
308
309     // At non-patched sides allow mesh to slip if extrusion direction makes
310     // angle larger than slipFeatureAngle.
311     slipFeatureAngle 30;
312
313     // Maximum number of snapping relaxation iterations. Should stop
314     // before upon reaching a correct mesh.
315     nRelaxIter 5;
316
317     // Number of smoothing iterations of surface normals
318     nSmoothSurfaceNormals 1;
319
320     // Number of smoothing iterations of interior mesh movement direction
321     nSmoothNormals 3;
322
323     // Smooth layer thickness over surface patches
324     nSmoothThickness 10;
325
326     // Stop layer growth on highly warped cells
327     maxFaceThicknessRatio 0.5;
328
329     // Reduce layer growth where ratio thickness to medial
330     // distance is large
331     maxThicknessToMedialRatio 0.3;
332
333     // Angle used to pick up medial axis points
334     // Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130 in 17x.
335     minMedianAxisAngle 90;
336
337     // Create buffer region for new layer terminations
338     nBufferCellsNoExtrude 0;
339
340     // Overall max number of layer addition iterations. The mesher will exit
341     // if it reaches this number of iterations; possibly with an illegal
342     // mesh.
343     nLayerIter 50;
```

```
344
345         // Max number of iterations after which relaxed meshQuality controls
346         // get used. Up to nRelaxIter it uses the settings in meshQualityControls,
347         // after nRelaxIter it uses the values in meshQualityControls::relaxed.
348         nRelaxedIter 20;
349
350         // Additional reporting: if there are just a few faces where there
351         // are mesh errors (after adding the layers) print their face centres.
352         // This helps in tracking down problematic mesh areas.
353         //additionalReporting true;
354     }
355
356     // Generic mesh quality settings. At any undoable phase these determine
357     // where to undo.
358     meshQualityControls
359     {
360         // Maximum non-orthogonality allowed. Set to 180 to disable.
361         maxNonOrtho 45;
362
363         // Max skewness allowed. Set to <0 to disable.
364         maxBoundarySkewness 20;
365         maxInternalSkewness 4;
366
367         // Max concaveness allowed. Is angle (in degrees) below which concavity
368         // is allowed. 0 is straight face, <0 would be convex face.
369         // Set to 180 to disable.
370         maxConcave 80;
371
372         // Minimum pyramid volume. Is absolute volume of cell pyramid.
373         // Set to a sensible fraction of the smallest cell volume expected.
374         // Set to very negative number (e.g. -1E30) to disable.
375         minVol 1e-13;
376
377         // Minimum quality of the tet formed by the face-centre
378         // and variable base point minimum decomposition triangles and
379         // the cell centre. This has to be a positive number for tracking
380         // to work. Set to very negative number (e.g. -1E30) to
381         // disable.
382         //     <0 = inside out tet,
383         //      0 = flat tet
384         //      1 = regular tet
385         minTetQuality 1e-9;
386
387         // Minimum face area. Set to <0 to disable.
388         minArea -1;
389
390         // Minimum face twist. Set to <-1 to disable. dot product of face normal
391         // and face centre triangles normal
392         minTwist 0.05;
393
394         // minimum normalised cell determinant
395         // 1 = hex, <= 0 = folded or flattened illegal cell
396         minDeterminant 0.001;
397
398         // minFaceWeight (0 -> 0.5)
399         minFaceWeight 0.05;
400
```



```
401         // minVolRatio (0 -> 1)
402         minVolRatio 0.01;
403
404         // must be >0 for Fluent compatibility
405         minTriangleTwist -1;
406
407         //- if >0 : preserve single cells with all points on the surface if the
408         //  resulting volume after snapping (by approximation) is larger than
409         //  minVolCollapseRatio times old volume (i.e. not collapsed to flat cell).
410         //  If <0 : delete always.
411         //minVolCollapseRatio 0.5;
412
413         // Advanced
414
415         // Number of error distribution iterations
416         nSmoothScale 4;
417         // amount to scale back displacement at error points
418         errorReduction 0.75;
419
420         // Optional : some meshing phases allow usage of relaxed rules.
421         // See e.g. addLayersControls::nRelaxedIter.
422         relaxed
423         {
424             //- Maximum non-orthogonality allowed. Set to 180 to disable.
425             maxNonOrtho 45;
426         }
427     }
428
429     // Advanced
430
431     // Flags for optional output
432     // 0 : only write final meshes
433     // 1 : write intermediate meshes
434     // 2 : write volScalarField with cellLevel for postprocessing
435     // 4 : write current intersections as .obj files
436     debug 0;
437
438     // Merge tolerance. Is fraction of overall bounding box of initial mesh.
439     // Note: the write tolerance needs to be higher than this.
440     mergeTolerance 1e-6;
441
442     // ************************************************************************* //
```

To start meshing, make sure that the first switch is set to true while the others are set to false (lines 18 through 20). Then, within the case directory, type:

```
snappyHexMesh
```

The first switch, controlling the castellatedMeshControls uses the cells of the background mesh to divide the domain in small cubes according to the instructions specified in the dictionary. A new directory has appeared (named *1* if deltaT and

writeInterval have been previously set to 1). It contains the information of the first meshing process of snappyHexMesh.

The results are shown in the following figure:

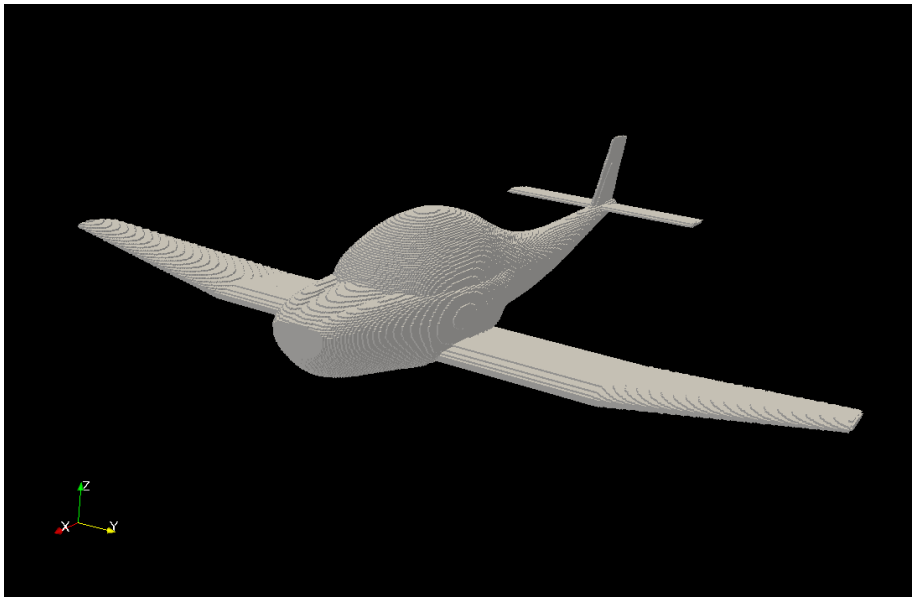


Figure 6.6: Shape of the aircraft at the first step of the meshing process of snappyHexMesh

To obtain the results shown in Figure 6.6, launch ParaView, go to time interval 1 and select `aircraftPatch` located in Mesh Parts.

As it was previously said, the internal cells of the aircraft are removed. This characteristic, as well as the cell refinement distribution along the domain can be observed in the following figures:

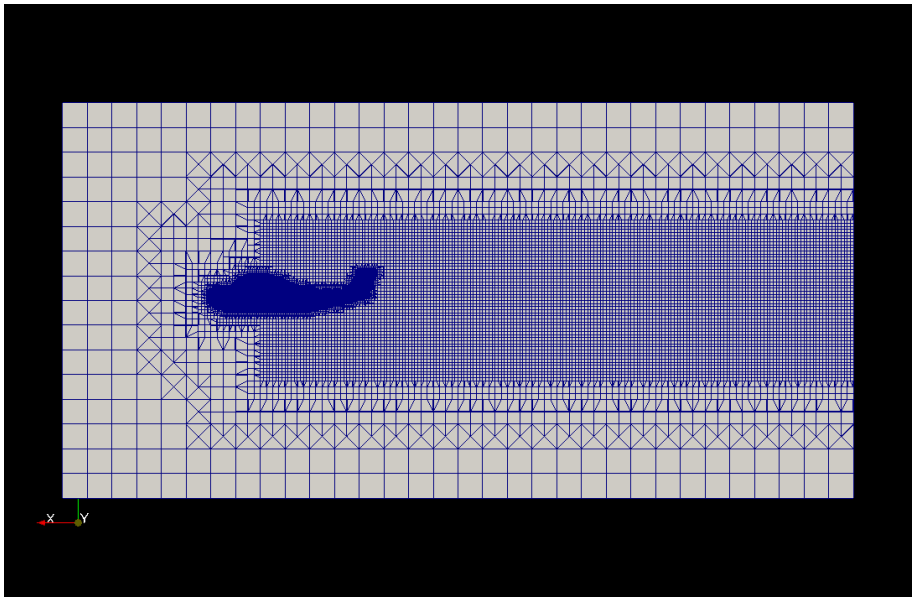


Figure 6.7: Mesh of the domain at the first step of the meshing process of snappyHexMesh

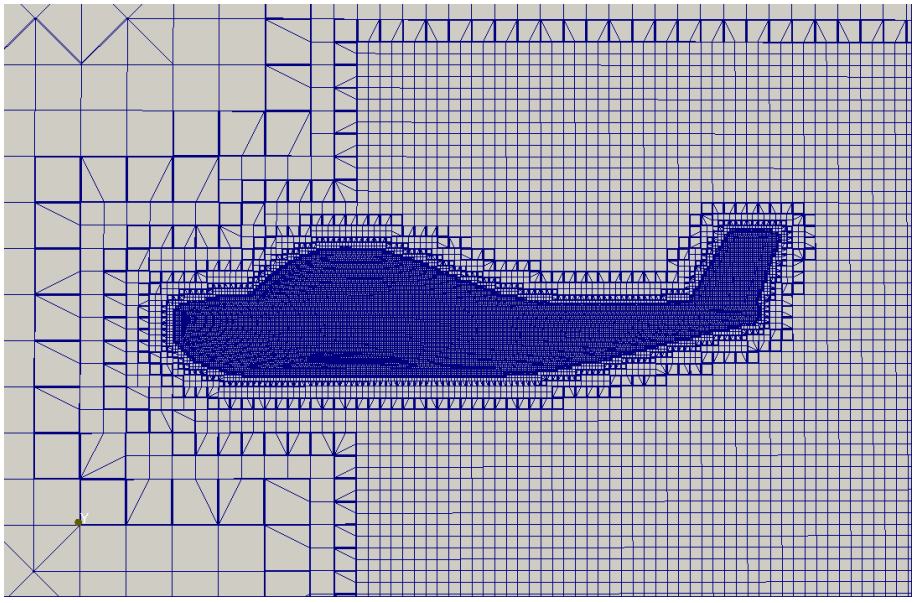


Figure 6.8: Detail of the mesh at the first step of the meshing process of snappyHexMesh

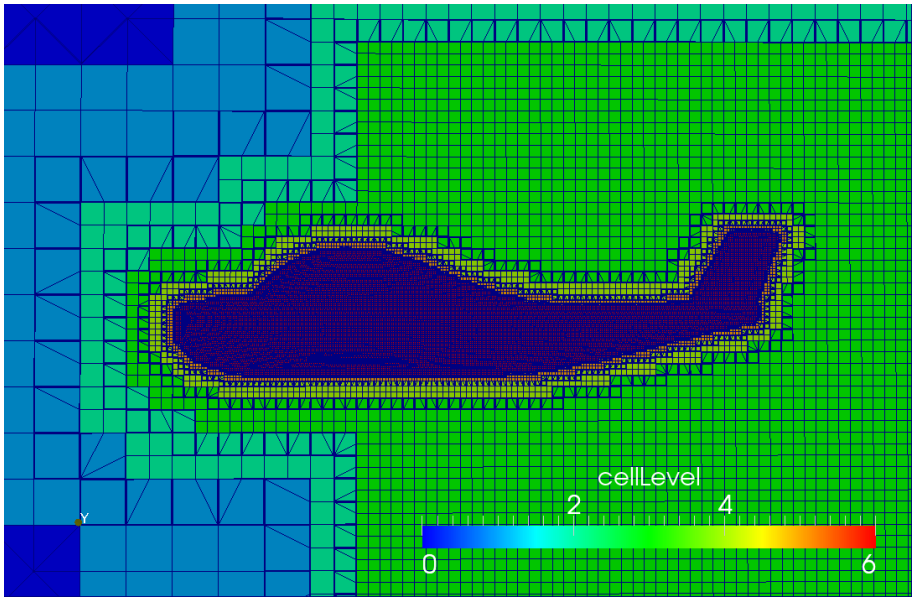


Figure 6.9: Detail of the mesh with a representation of the cell refinement at the first step of the meshing process of snappyHexMesh

For the representation of the previous Figures, `internalMesh` has to be selected instead of `aircraftPatch`. Moreover, as the aircraft shape is internal, a cut using the Clip icon has to be used. To show the cell refinement as a field, click the Volume Fields box located within Properties. Then, select the cellLevel option in the first drop-down menu at the top of ParaView's screen.

As it can be seen in previous figures, the surface of the aircraft is irregular and cube-based. The second switch of snappyHexMesh controls the snapControls, involving the displacement of boundary vertices to conform to surface. All vertex displacements are reversible to ensure mesh quality.

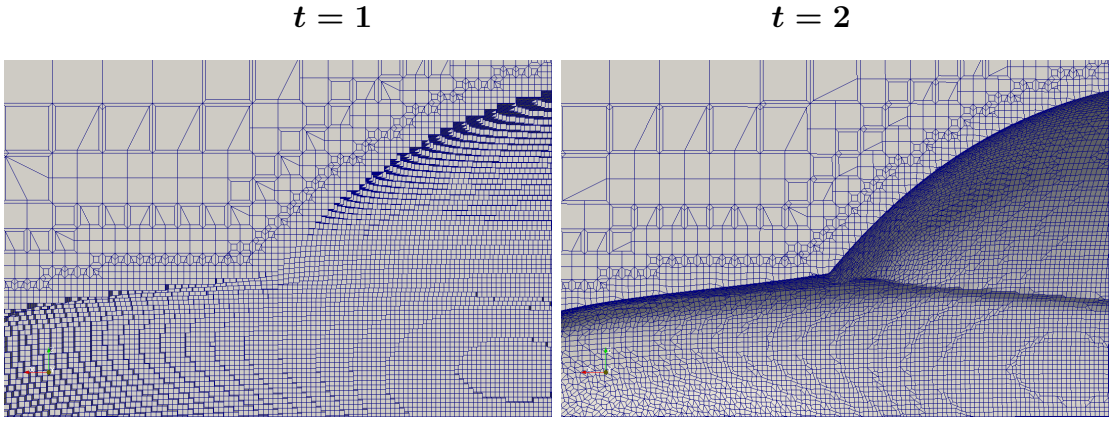
To proceed to the next step, set the first switch to false, the second to true and type:

```
snappyHexMesh
```

Once it has been run, the user can observe the *2* directory within the case containing the information generated. During this second process, castellated mesh boundary and internal mesh have been smoothed. Furthermore, mesh boundary is *snapped* to geometry surface.

Now, in ParaView, the user can use the time control buttons to view the background mesh ($t = 0$), the castellated mesh ($t = 1$) and the mesh generated in the second step of snappyHexMesh ($t = 2$). In the following figures it is possible to observe the

differences between directories *1* and *2*:



Now, the mesh of the *Very Light Aircraft* case is ready.

It exists a third switch in snappyHexMesh which introduces cell layers at the patches of the case. When doing it, surface points are identified and are not displaced. In the current tutorial, this third process of snappyHexMesh is not going to be used.

Finally, it is important to mention that the *snappyHexMeshDict* file is complex to fully understand; it includes a lot of instructions and only with time and experience it is possible to reach a broad domain of its contents. Nevertheless, the user can identify some basic instructions that are relevant when preparing the meshing process:

- **Name of the geometry used for the meshing process**: line 30
- **Definition of cell refinement level at the patches of the domain**: lines 118 through 140
- **Specification of a point contained in the volume which will be meshed**: line 184

Regarding the last sentence, as in the current case the point specified is located outside the aircraft, its internal cells have been removed while the background mesh has become the fluid domain.

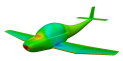
Advice:

> If the user wants to remesh the case (changing or not the instructions of *snappyHexMeshDict*) it is first necessary to remove *1* and *2* directories. Then, after setting the switches properly, snappyHexMesh can be run again

6.0.21.2 Boundary and initial conditions

Besides the *p* and *U* files, as the case is set turbulent, three new files are necessary to be created. Their names are *k*, *nut* and *omega*. These dictionaries contain the boundary conditions of the parameters used to implement the SST k-w turbulence model. Before discussing their calculation, the boundary conditions for p and **U** are:

```
1  /*---------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.2.1                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      p;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 2 -2 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     inlet
24     {
25         type            freestreamPressure;
26     }
27
28     outlet
29     {
30         type            freestreamPressure;
31     }
32
33     aircraftPatch
34     {
35         type            zeroGradient;
36     }
37
38     top
39     {
40         type            slip;
41     }
42
43     bottom
44     {
45         type            slip;
46     }
47
```



```
48      frontAndBack
49      {
50          type            slip;
51      }
52
53  }
54
55  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       volVectorField;
13      object      U;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (-45 0 0);
20
21  boundaryField
22  {
23      inlet
24      {
25          type            freestream;
26          freestreamValue uniform (-45 0 0);
27      }
28
29      outlet
30      {
31          type            freestream;
32          freestreamValue uniform (-45 0 0);
33      }
34
35      aircraftPatch
36      {
37          type            fixedValue;
38          value           uniform (0 0 0);
39      }
40
41      top
42      {
43          type            slip;
44      }
45
46      bottom
47      {
```

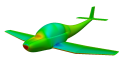
```
48          type            slip;
49      }
50
51      frontAndBack
52      {
53          type            slip;
54      }
55  }
56
57  // ************************************************************************* //
```

Caution:

As in the current case there is not *0* directory, the dictionaries of *p*, *U*, *k*, *nut* and *omega* are included within the last directory generated with snappyHexMesh

The dictionaries of *k*, *nut* and *omega* are:

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      binary;
12      class       volScalarField;
13      location    "2";
14      object      k;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  dimensions      [0 2 -2 0 0 0 0];
19
20  internalField   uniform 1.215;
21
22  boundaryField
23  {
24      inlet
25      {
26          type            fixedValue;
27          value           uniform 1.215;
28      }
29
30      outlet
31      {
32          type            inletOutlet;
33          inletValue      uniform 1.215;
34          value           uniform 1.215;
```

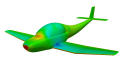


```
35      }
36
37      aircraftPatch
38      {
39          type            kqRWallFunction;
40          value           uniform 1.215;
41      }
42
43      top
44      {
45          type            slip;
46      }
47
48      bottom
49      {
50          type            slip;
51      }
52
53      frontAndBack
54      {
55          type            slip;
56      }
57  }
58
59  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      binary;
12      class       volScalarField;
13      location    "2";
14      object      nut;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  dimensions      [0 2 -1 0 0 0 0];
19
20  internalField   uniform 0;
21
22  boundaryField
23  {
24      inlet
25      {
26          type            calculated;
27          value           uniform 0;
28      }
29
30      outlet
```

```
31      {
32          type            calculated;
33          value           uniform 0;
34      }
35
36      aircraftPatch
37      {
38          type            nutkWallFunction;
39          value           uniform 0;
40      }
41
42      top
43      {
44          type            calculated;
45          value           uniform 0;
46      }
47
48      bottom
49      {
50          type            calculated;
51          value           uniform 0;
52      }
53
54      frontAndBack
55      {
56          type            calculated;
57          value           uniform 0;
58      }
59  }
60
61  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      binary;
12      class       volScalarField;
13      location    "2";
14      object      omega;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  dimensions      [0 0 -1 0 0 0 0];
19
20  internalField   uniform 2147.745;
21
22  boundaryField
23  {
24      inlet
```



```
25      {
26          type            fixedValue;
27          value           uniform 2147.745;
28      }
29
30      outlet
31      {
32          type            inletOutlet;
33          inletValue      uniform 2147.745;
34          value           uniform 2147.745;
35      }
36
37      aircraftPatch
38      {
39          type            omegaWallFunction;
40          value           uniform 2147.745;
41      }
42
43      top
44      {
45          type            slip;
46      }
47
48      bottom
49      {
50          type            slip;
51      }
52
53      frontAndBack
54      {
55          type            slip;
56      }
57  }
58
59  // ************************************************************************* //
```

k, ν_t and ω are parameters required to simulate the case with a turbulence model. Mathematically, they are determined as:

$$k = \frac{3}{2}(|\mathbf{U}|I)^2 \tag{6.5}$$

$$\omega = \frac{C_\mu^{-0.25} k^{0.5}}{l} \tag{6.6}$$

$$C_\mu = 0.09k \tag{6.7}$$

Where $|\mathbf{U}|$ is the mean flow velocity, I is the turbulence intensity and l the turbulent length scale. They can be computed as:

$$I \approx 2\%$$

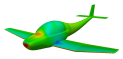
$$l \approx 0.07 \cdot c$$

With these assumptions, $k = 1.215$, $C_\mu = 0.109$ and $\omega = 2147.745$.

6.0.21.3 Physical properties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              nu [0 2 -1 0 0 0 0] 1.5e-07;

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel            kOmegaSST;

turbulence          on;

```



```
printCoeffs           on;

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

Remember that although the medium is air, $\nu = 1.5 \times 10^{-7}$ to maintain the Reynolds number as the dimensional analysis showed.

6.0.21.4 Control

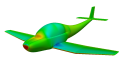
```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         300;

deltaT          1;

writeControl    timeStep;

writeInterval   1;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

```
46
47  functions
48  {
49      #include "readFields"
50      #include "cuttingPlane"
51      #include "forceCoeffs"
52  }
53
54  // ************************************************************************* //
```

As it can be seen, at lines 47 through 52 the program is using external functions to develop specific tasks. While the case is running, these external functions are called using its dictionaries located within *system*. It facilitates the treatment of the code. For instance, it is not necessary to include all the instructions to calculate the force coefficients directly in *controlDict*. The dictionaries are shown in Section 6.0.21.6.

6.0.21.5 Discretization and linear-solver settings

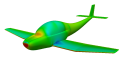
```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      fvSchemes;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  ddtSchemes
18  {
19      default         steadyState;
20  }
21
22  gradSchemes
23  {
24      default         Gauss linear;
25      grad(U)         cellLimited Gauss linear 1;
26  }
27
28  divSchemes
29  {
30      default         none;
31      div(phi,U)      bounded Gauss linearUpwindV grad(U);
32      div(phi,k)      bounded Gauss upwind;
33      div(phi,omega)  bounded Gauss upwind;
34      div((nuEff*dev(T(grad(U))))) Gauss linear;
35  }
```



```
36
37  laplacianSchemes
38  {
39      default         Gauss linear corrected;
40  }
41
42  interpolationSchemes
43  {
44      default         linear;
45  }
46
47  snGradSchemes
48  {
49      default         corrected;
50  }
51
52  fluxRequired
53  {
54      default         no;
55      p;
56  }
57
58  // ************************************************************************* //
```

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      fvSolution;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  solvers
18  {
19      p
20      {
21          solver           GAMG;
22          tolerance        1e-7;
23          relTol           0.01;
24          smoother         GaussSeidel;
25          nPreSweeps       0;
26          nPostSweeps      2;
27          cacheAgglomeration on;
28          agglomerator     faceAreaPair;
29          nCellsInCoarsestLevel 10;
30          mergeLevels      1;
31      }
32
```

```
33          U
34          {
35              solver          smoothSolver;
36              smoother        GaussSeidel;
37              tolerance       1e-8;
38              relTol          0.1;
39              nSweeps         1;
40          }
41
42          k
43          {
44              solver          smoothSolver;
45              smoother        GaussSeidel;
46              tolerance       1e-8;
47              relTol          0.1;
48              nSweeps         1;
49          }
50
51          omega
52          {
53              solver          smoothSolver;
54              smoother        GaussSeidel;
55              tolerance       1e-8;
56              relTol          0.1;
57              nSweeps         1;
58          }
59      }
60
61      SIMPLE
62      {
63          nCorrectors                 1;
64          nNonOrthogonalCorrectors    2;
65          pRefCell                    0;
66          pRefValue                   0;
67      }
68
69      potentialFlow
70      {
71          nNonOrthogonalCorrectors    10;
72      }
73
74      relaxationFactors
75      {
76          fields
77          {
78              p               0.3;
79          }
80          equations
81          {
82              U               0.5;
83              k               0.5;
84              omega           0.5;
85          }
86      }
87
88      cache
89      {
```



```
90      grad(U);
91  }
92
93  // ************************************************************************* //
```

6.0.21.6 External functions

This section contains the dictionaries used by the functions of *controlDict*. They must be copied into files named *readFiles*, *forceCoeffs* and *cuttingPlane* respectively. They are located within *system*.

readFields

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8
9   // Make sure all fields for functionObjects are loaded. Prevents any
10  // problems running with execFlowFunctionObjects.
11  readFields
12  {
13      // Where to load it from (if not already in solver)
14      functionObjectLibs ("libfieldFunctionObjects.so");
15
16      type            readFields;
17      fields          (p U k);
18  }
19
20
21  // ************************************************************************* //
```

forceCoeffs

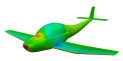
```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8
9   forces
10  {
11          type                forces;
12          functionObjectLibs  ( "libforces.so" ); //Lib to load
```

```
13          outputControl       timeStep;
14          outputInterval      1;
15          patches             (aircraftPatch); //Patch name over forces will be
                calculated
16          pName               p;
17          UName               U;
18          rhoName             rhoInf; //Reference density
19          log                 true;
20          rhoInf              1.225; //Air density
21          CofR                (0 0 0); //Origin for moment calculations
22  }
23
24  forceCoeffs
25  {
26  type forceCoeffs;
27  functionObjectLibs ("libforces.so");
28  patches (aircraftPatch);
29  // pName p;
30  // UName U;
31  rhoName rhoInf;
32  rhoInf 1.225;
33  CofR (0 0 0);
34  liftDir (0 1 0);
35  dragDir (-1 0 0);
36  pitchAxis (0 0 1);
37  magUInf 45; // Free stream velocity
38  lRef 0.01276; // Mean Chord
39  Aref 0.001218; // Ref. Area
40  outputControl timeStep;
41  outputInterval 1;
42  }
43
44  // ************************************************************************* //
```

cuttingPlane

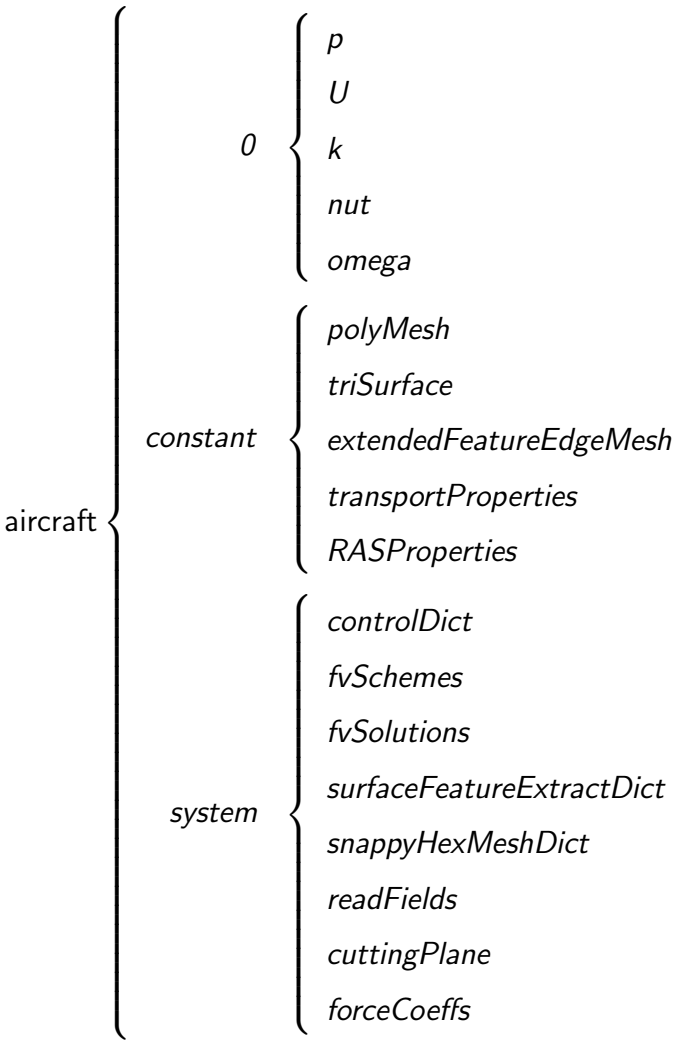
The *cuttingPlane* function generates a VTK plane cutting the fluid domain in a specified direction. It can be easily open from the *postProcessing* directory once the case has been run and allows a practial and rapid view of the p and **U** fields.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.2.1                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8
9   cuttingPlane
10  {
11      type            surfaces;
12      functionObjectLibs ("libsampling.so");
13      outputControl   outputTime;
14
```



```
15      surfaceFormat   vtk;
16      fields          ( p U );
17
18      interpolationScheme cellPoint;
19
20      surfaces
21      (
22          yNormal
23          {
24              type            cuttingPlane;
25              planeType       pointAndNormal;
26              pointAndNormalDict
27              {
28                  basePoint       (0 0 0);
29                  normalVector    (0 1 0);
30              }
31              interpolate     true;
32          }
33      );
34  }
35
36  // ************************************************************************* //
```

At the end of the pre-processing, the structure of directories, subdirectories and files within aircraft should be as follows:



6.0.22 Post-processing

6.0.22.1 Results of the simulation

The pressure field around the aircraft:

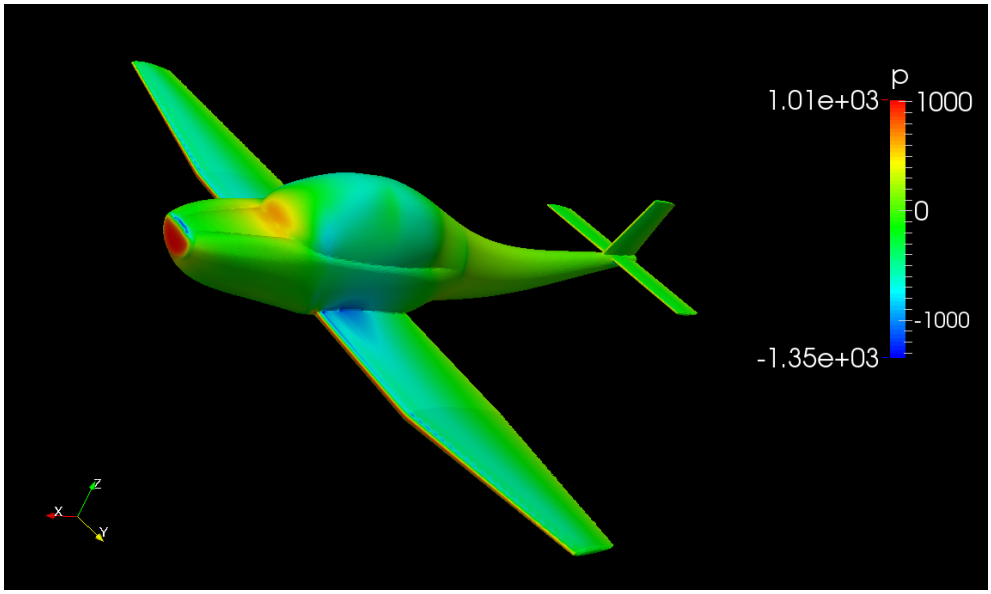


Figure 6.10: Pressure field around the aircraft (m^2/s^2)

The velocity field in the whole domain:

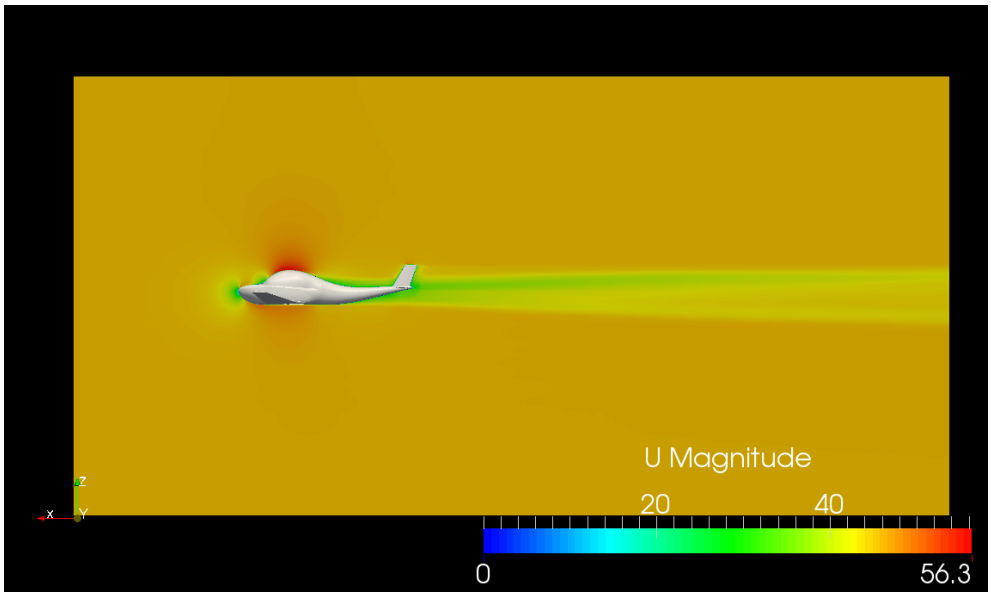


Figure 6.11: Velocity field in the domain of the aircraft case (m/s)

The velocity field around the aircraft:

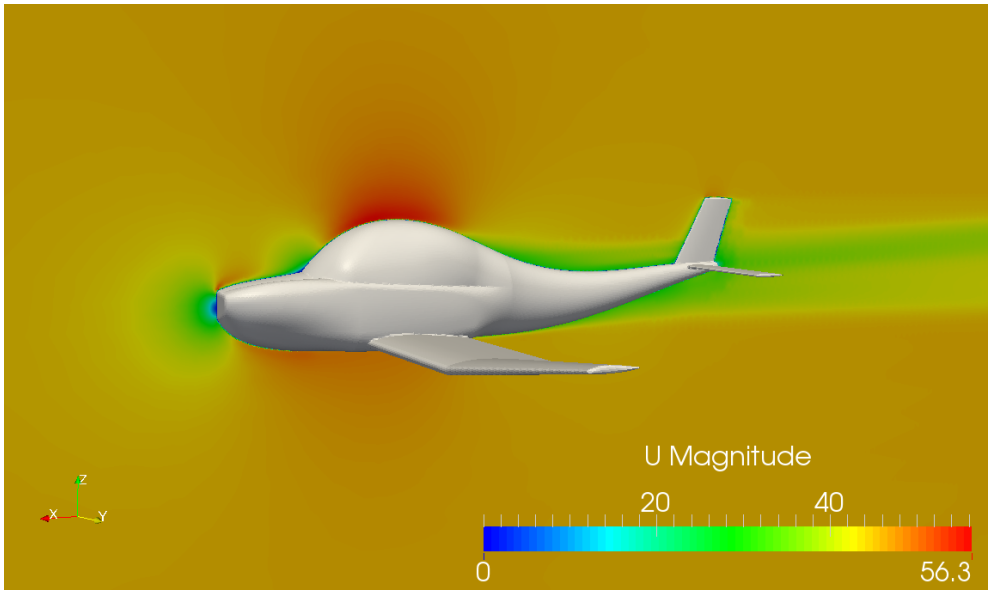


Figure 6.12: Velocity field around the aircraft (m/s)

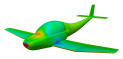
At the end of the simulation, the values of drag coefficient and lift coefficient stabilized at:

- $C_D = 0.0276$
- $C_L = 0.005$

Although the value of the drag coefficient matches with other real very light aircrafts (as for instance the Cessna 172), the lift coefficient is very low, and therefore the aerodynamic efficiency ($E = L/D$) is far from the current commercial aircrafts. As it was explained, the aircraft geometry used in this chapter was not designed according to a realistic aerodynamic study (as it would have been CFD or wind tunnel experiments).

Another consideration that must be done is the fact that although the case is set turbulent, the wake of the aircraft shows a continuous appearance. It is because a RAS turbulence model has been used, based on average flow conditions. If other turbulence models had been used (as for instance LES or DNS), it would have been possible to observe turbulence in the wake.

For Figures 6.11 and 6.12, the VTK plane obtained from the *postProcessing* directory has been used.

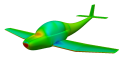


A. Additional codes

The code to create the mesh in Chapter 5:

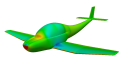
```
1  function foilgmsh(archi,alfa,yplus,eter,Re,M,T0,N,bump)
2  %foilgmsh(archi,alfa,yplus,eter,Re,M,T0,N,bump)
3  %
4  %foilgmsh will create a GEO file with all necessary instructions to mesh an
       airfoil geometry only with hexahedra in Gmsh, departuring from a set of points
       defining the airfoil section contour. So far this code can only deal with
       monoelement airfoils and 2D simulations for finite volume CFD codes.
5  %The on-screen output displays a summary of the input, some statistics of the mesh
       to be created and useful information for definition of the case in CFD
       softwares, specially Code_Saturne.
6  %
7  %- archi: string which defines the complete name of the airfoil coordinates file.
       The coordinates must be given in XFoil format starting from the trailing edge
       and circling counterclockwise. The coordinates don't need to be
       adimensionalized, but the program won't do it either, as the airfoil chord
       will be estimated automatically from it. The author consider's this is useful
       to compare further results with experimental data. The coordinates can be
       off-centered by small amounts.
8  %The file must be located in the active directory. The output file will have the
       same name with the string '.geo' appended. If you wish to include text in your
       coordinates file, please add a percentage sign on the first column, otherwise
       an error will occur.
9  %
10 %- alfa: desired angle of attack in degrees. The mesh generated will be so in wind
       axis system, where wind velocity coincides in sense and direction with the
       positive X axis, so when defining inlet speed, Y and Z components should be
       zero. Also, the section plane coincides with the mesh XY plane.
11 %
12 %- yplus: desired y+ value around the airfoil. Spacing between susecquent cells
       normal to the airfoil surface is done with an authomatically defined geometric
       progression.
13 %
14 %- eter: string defining material medium where airfoil is submerged. Only three
       posibilities are allowed, 'a' for normal air, 'h' for distilled water and 'n'
       for nitrogen.
15 %
16 %- Re: Reynolds number based on airfoil chord and freestream speed. The reference
       chord and that of the airfoil in the coordinates file must coincide.
17 %
18 %- M: if the medium is air or nitrogen, it is the Mach number. If the medium is
       water, it is the freestream speed magnitude in m/s. Although the applied
       formulas hold for transonic and supersonic flows, the resulting mesh might not
       be suitable for those cases. Compressible fully subsonic flows should not be a
       problem.
19 %
```

```
20  %- T0: if the medium is air or nitrogen, it is the stagnation temperature in
        Kelvin degrees. If the medium is water, it is the non-stagnated flow
        temperature in Celsius degrees.
21  %
22  %- N: four integers vector whre N(1) is the number of hexahedra along the airfoil
        chord, therefore the whole contour of the airfoil will be discretized in
        2*N(1) elements. N(2) is the number of elements normal to the chord and inside
        a semiellipse closely enclosing the airfoil (a value between 25 and 100 should
        suffice for most applications). N(3) is the number of elements into which the
        horizontal leading edge-inlet and trailing edge-outlet gaps will be
        discretized. N(4) is similar to N(3) but applied to the vertical gaps between
        the airfoil and the top/bottom walls defining the box.
23  %
24  %- bump: a value which defines the mesh concentration degree around the leading
        and trailing edge. If bump=1 the cell lenghts will be uniform along the
        airfoil contour, if bump>1 the elements will be concentrated around the
        midchord. If 0<bump<1 the elements will concentrate around the edges, and
        don't be surprised if you need very low values like 0.1 or less to achieve a
        noticeable concentration.
25  %
26  %To mesh, simply open in Gmsh the generated GEO file, go to Mesh with the menu or
        by pressing 'm' and click on "3D". Save the mesh as a MED file for use with
        Code_Saturne (remember to apply 'check cell orientation' in the GUI or
        preprocessor). The generated groups are "inlet", "outlet", "airfoil",
        "symmetry" and "walls".
27  %The on-screen output text provides information to define all necessary variables
        in CS's GUI. The hydraulic diameter value is just a dummy number suitable to
        initialize properly the turbulence model for external flows.
28  %
29  %Send some feedback if you wish to cesar_vecchio@gmx.com (I also accept Ferraris
        and Porsches). I hope you find this software useful.
30  %----------------------
31  %Cesar A. Vecchio Toloy
32  %----------------------
33  %Disclaimer: I am giving you this software as is fully for free. I will not be
        responsible for any harm of any kind this code and the uses you give to it may
        cause. You are using this code under your own responsability and risk.
34
35  more off
36
37  N = N+1; %number of nodes on upper and lower surface
38  alfa = -alfa*pi/180; %conversion to radians and Gmsh references
39  inic = load(archi); %loading coordinates file...
40  [m void] = size(inic);
41  inic(1,2)=(inic(1,2)+inic(m,2))/2; %the trailing edge is closed. Sorry for the
        inconvenience.
42  percor=inic(1:m-1,:); %the (now) extra trailing edge point is removed.
43  m=m-1;disp(m)
44  z0 = 0;
45  [maxx posmaxx] = max(percor(:,1));
46  [minx posminx] = min(percor(:,1));
47  cuerda = maxx-minx; %computation of airfoil chord
48  [maxy posmaxy] = max(percor(:,2));
49  [miny posminy] = min(percor(:,2));
50
51  fid = fopen(strcat(archi,'.geo'),'w'); %opening the output file
52
```



```
53  %writing the points which define the airfoil
54  for i = 1:m
55  fprintf(fid,'Point(%i) =
        {%.10g,%.10g,%.10g,%.10g};\n',i,percor(i,1),percor(i,2),z0,cuerda/100);
56  end
57
58  %writing the points which define the enclosing semiellipse
59  fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+1,minx-cuerda/20,0,z0,cuerda/25);
60  fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+2,maxx,maxy+cuerda/4,z0,cuerda/25);
61  fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+3,maxx,miny-cuerda/4,z0,cuerda/25);
62
63  fprintf (fid, 'Spline(1) = {'); %defining an interpolating spline for the upper
        surface
64  for i = posminx:m
65  fprintf (fid, '%i,', i);
66  end
67  %The following and all the lines where you see 'Transfinite' is a way of
        indicating to Gmsh that a structured mesh will be made.
68  fprintf (fid, '%i}; Transfinite Line{1} = %i Using Bump %f;\n', 1,N(1),bump);
69  fprintf (fid, 'Spline(2) = {'); %lower surface interpolating spline
70  for i = 1:posminx-1
71  fprintf (fid, '%i,', i);
72  end
73  fprintf (fid, '%i}; Transfinite Line{2} = %i Using Bump %f;\n', posminx,N(1),bump);
74
75  %Defining the lines of our enclosing semiellipse
76  fprintf (fid, 'Ellipse(3) = {%i,%i,%i,%i}; Transfinite Line{3} = %i Using
        Progression 1;\n', m+1,1,posminx,m+2,N(1));
77  fprintf (fid, 'Ellipse(4) = {%i,%i,%i,%i}; Transfinite Line{4} = %i Using
        Progression 1;\n', m+1,1,posminx,m+3,N(1));
78  %Calculating minimum cell distance from wall and geometric progression with
        subfunctions
79  Ymin = ypar (yplus,cuerda,Re,M,T0,eter); Prog5 =
        mindist(Ymin,norm(percor(posmaxx,:)-[maxx,maxy+cuerda/4]),N(2));
80  fprintf (fid, 'Line(5) = {%i,%i}; Transfinite Line{5} = %i Using Progression
        %.10g;\n', 1,m+2,N(2),Prog5);
81  Prog6 = mindist(Ymin,norm(percor(posmaxx,:)-[maxx,miny-cuerda/4]),N(2));
82  fprintf (fid, 'Line(6) = {%i,%i}; Transfinite Line{6} = %i Using Progression
        %.10g;\n', 1,m+3,N(2),Prog6);
83  Prog7 = mindist(Ymin,norm(percor(posminx,:)-[minx-cuerda/20,0]),N(2));
84  fprintf (fid, 'Line(7) = {%i,%i}; Transfinite Line{7} = %i Using Progression
        %.10g;\n', posminx,m+1,N(2),Prog7);
85
86  %2D surfaces are created from the available liens so far
87  fprintf (fid, 'Line Loop(1) = {-2,5,-3,-7};\n');
88  fprintf (fid, 'Ruled Surface(1) = {1};\n');
89  fprintf (fid, 'Transfinite Surface(1) = {%i,%i,%i,%i};\n', 1,posminx,m+1,m+2);
90  fprintf (fid, 'Line Loop(2) = {1,6,-4,-7};\n');
91  fprintf (fid, 'Ruled Surface(2) = {2};\n');
92  fprintf (fid, 'Transfinite Surface(2) = {%i,%i,%i,%i};\n', 1,posminx,m+1,m+3);
93
94  %Let's tell Gmsh to rotate the airfoil+semielipse our desired angle of attack
95  fprintf (fid, 'Rotate {{0,0,1},{%.10g,0,0},%.10g} {Surface{1,2};}\n',
        minx+cuerda/2,alfa);
```

```
96
97  %Now some points to define the flowfield boundaries. Notice the resulting box will
        be 15*chord long and 8*chord high.
98  fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+4,maxx-(1-cos(alfa))*cuerda/2-sin(alfa)*(cuerda/4+maxy),4*cuerda,z0,cuerda);
99  fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+5,minx-2*cuerda,4*cuerda,z0,cuerda);
100 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+6,minx-4*cuerda,4*cuerda,z0,cuerda);
101 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+7,minx-4*cuerda,0-0.55*cuerda*sin(alfa),z0,cuerda);
102 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+8,minx-4*cuerda,-4*cuerda,z0,cuerda);
103 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+9,minx-2*cuerda,-4*cuerda,z0,cuerda);
104 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+10,maxx-(1-cos(alfa))*cuerda/2-sin(alfa)*(-cuerda/4+miny),-4*cuerda,z0,cuerda);
105 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+11,maxx+10*cuerda,-4*cuerda,z0,cuerda);
106 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+12,maxx+10*cuerda,percor(posmaxx,2)+cos(alfa)*(miny-cuerda/4)+cuerda/2*sin(alfa),z0,cuerda);
107 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+13,maxx+10*cuerda,sin(alfa)*cuerda/2,z0,cuerda);
108 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+14,maxx+10*cuerda,percor(posmaxx,2)+cos(alfa)*(maxy+cuerda/4)+cuerda/2*sin(alfa),z0,cuerda);
109 fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',
        m+15,maxx+10*cuerda,4*cuerda,z0,cuerda);
110
111 %Now we join the previous points with some lines
112 Prog =
        mindist(Ymin*Prog7^(N(2)-1),abs(3.95*cuerda+(1-cos(alfa))*0.55*cuerda),N(3));
113 fprintf (fid, 'Line(8) = {%i,%i}; Transfinite Line{8} = %i Using Progression
        %.10g;\n', m+1,m+7,N(3),Prog);
114
115 Prog = mindist(cuerda/N(1),abs(10*cuerda+(1+cos(alfa))*0.5*cuerda),N(3));
116 fprintf (fid, 'Line(9) = {%i,%i}; Transfinite Line{9} = %i Using Progression
        %.10g;\n', 1,m+13,N(3),Prog);
117
118 L = 4*cuerda-(percor(posmaxx,2)+cuerda/4)*cos(alfa)-cuerda/2*sin(alfa);
119 Prog = mindist(Ymin*Prog5^(N(2)-1),L,N(4));
120 fprintf (fid, 'Line(10) = {%i,%i}; Transfinite Line{10} = %i Using Progression
        %.10g;\n', m+2,m+4,N(4),Prog);
121
122 L =
        norm([minx-cuerda*2,cuerda*4]-[(minx-cuerda/20)-(1-cos(alfa))*0.55*cuerda,percor(posminx,2)-sin(
123 Prog = mindist(Ymin*Prog7^(N(2)-1),L,N(4));
124 fprintf (fid, 'Line(11) = {%i,%i}; Transfinite Line{11} = %i Using Progression
        %.10g;\n', m+1,m+5,N(4),Prog);
125
126 L = 4*cuerda+(miny-cuerda/4)*cos(alfa)+cuerda/2*sin(alfa);
127 Prog = mindist(Ymin*Prog6^(N(2)-1),L,N(4));
128 fprintf (fid, 'Line(12) = {%i,%i}; Transfinite Line{12} = %i Using Progression
        %.10g;\n', m+3,m+10,N(4),Prog);
129
130 L =
        norm([minx-cuerda*2,-cuerda*4]-[(minx-cuerda/20)-(1-cos(alfa))*0.55*cuerda,percor(posminx,2)-sin(
131 Prog = mindist(Ymin*Prog7^(N(2)-1),L,N(4));
```



```
132 fprintf (fid, 'Line(13) = {%i,%i}; Transfinite Line{13} = %i Using Progression
        %.10g;\n', m+1,m+9,N(4),Prog);
133
134 Prog =
        mindist(cuerda/N(1),abs(10*cuerda+(1-cos(alfa))*0.5*cuerda+sin(alfa)*(maxy+cuerda/4)),N(3));
135 fprintf (fid, 'Line(14) = {%i,%i}; Transfinite Line{14} = %i Using Progression
        %.10g;\n', m+2,m+14,N(3),Prog);
136
137 Prog =
        mindist(cuerda/N(1),abs(10*cuerda+(1-cos(alfa))*0.5*cuerda+sin(alfa)*(miny-cuerda/4)),N(3));
138 fprintf (fid, 'Line(15) = {%i,%i}; Transfinite Line{15} = %i Using Progression
        %.10g;\n', m+3,m+12,N(3),Prog);
139
140 fprintf (fid, 'Line(16) = {%i,%i}; Transfinite Line{16} = %i Using Progression
        1.00;\n', m+4,m+5,N(1));
141
142 fprintf (fid, 'Line(17) = {%i,%i}; Transfinite Line{17} = %i Using Progression
        1.00;\n', m+10,m+9,N(1));
143
144 Prog =
        mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(maxy+cuerda/4))/N(1),2*cuerda,N(3));
145 fprintf (fid, 'Line(18) = {%i,%i}; Transfinite Line{18} = %i Using Progression
        %.10g;\n', m+5,m+6,N(3),Prog);
146
147 Prog =
        mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(miny-cuerda/4))/N(1),2*cuerda,N(3));
148 fprintf (fid, 'Line(19) = {%i,%i}; Transfinite Line{19} = %i Using Progression
        %.10g;\n', m+9,m+8,N(3),Prog);
149
150 Prog = mindist(cuerda/N(1),4*cuerda+cuerda*0.55*sin(alfa),N(4));
151 fprintf (fid, 'Line(20) = {%i,%i}; Transfinite Line{20} = %i Using Progression
        %.10g;\n', m+7,m+6,N(4),Prog);
152
153 Prog = mindist(cuerda/N(1),4*cuerda-cuerda*0.55*sin(alfa),N(4));
154 fprintf (fid, 'Line(21) = {%i,%i}; Transfinite Line{21} = %i Using Progression
        %.10g;\n', m+7,m+8,N(4),Prog);
155
156 Prog =
        mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(maxy+cuerda/4))/N(1),abs(10*cuerda+(1-cos(alfa))*
157 fprintf (fid, 'Line(22) = {%i,%i}; Transfinite Line{22} = %i Using Progression
        %.10g;\n', m+4,m+15,N(3),Prog);
158
159 Prog =
        mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(miny-cuerda/4))/N(1),abs(10*cuerda+(1-cos(alfa))*
160 fprintf (fid, 'Line(23) = {%i,%i}; Transfinite Line{23} = %i Using Progression
        %.10g;\n', m+10,m+11,N(3),Prog);
161
162 L = 4*cuerda+(miny-cuerda/4)*cos(alfa)+cuerda/2*sin(alfa);
163 Prog = mindist(Ymin*Prog6^(N(2)-1),L,N(4));
164 fprintf (fid, 'Line(24) = {%i,%i}; Transfinite Line{24} = %i Using Progression
        %.10g;\n', m+12,m+11,N(4),Prog);
165
166 Prog = mindist(Ymin,abs(miny-cuerda/4)*cos(alfa),N(2));
167 fprintf (fid, 'Line(25) = {%i,%i}; Transfinite Line{25} = %i Using Progression
        %.10g;\n', m+13,m+12,N(2),Prog);
168
169 Prog = mindist(Ymin,abs(maxy+cuerda/4)*cos(alfa),N(2));
```

```
170 fprintf(fid, 'Line(26) = {%i,%i}; Transfinite Line{26} = %i Using Progression
        %.10g;\n', m+13,m+14,N(2),Prog);
171
172 L = 4*cuerda-(maxy+cuerda/4)*cos(alfa)-cuerda/2*sin(alfa);
173 Prog = mindist(Ymin*Prog5^(N(2)-1),L,N(4));
174 fprintf(fid, 'Line(27) = {%i,%i}; Transfinite Line{27} = %i Using Progression
        %.10g;\n', m+14,m+15,N(4),Prog);
175
176 %2D surfaces are defined from the previous lines.
177 fprintf(fid, 'Line Loop(3) = {3,10,16,-11};\n');
178 fprintf(fid, 'Ruled Surface(3) = {3};\n');
179 fprintf(fid, 'Transfinite Surface(3) = {%i,%i,%i,%i};\n', m+1,m+2,m+4,m+5);
180
181 fprintf(fid, 'Line Loop(4) = {4,12,17,-13};\n');
182 fprintf(fid, 'Ruled Surface(4) = {4};\n');
183 fprintf(fid, 'Transfinite Surface(4) = {%i,%i,%i,%i};\n', m+1,m+3,m+10,m+9);
184
185 fprintf(fid, 'Line Loop(5) = {-18,-11,8,20};\n');
186 fprintf(fid, 'Ruled Surface(5) = {5};\n');
187 fprintf(fid, 'Transfinite Surface(5) = {%i,%i,%i,%i};\n', m+1,m+5,m+6,m+7);
188
189 fprintf(fid, 'Line Loop(6) = {-19,-13,8,21};\n');
190 fprintf(fid, 'Ruled Surface(6) = {6};\n');
191 fprintf(fid, 'Transfinite Surface(6) = {%i,%i,%i,%i};\n', m+1,m+7,m+8,m+9);
192
193 fprintf(fid, 'Line Loop(7) = {5,14,-26,-9};\n');
194 fprintf(fid, 'Ruled Surface(7) = {7};\n');
195 fprintf(fid, 'Transfinite Surface(7) = {%i,%i,%i,%i};\n', m+2,m+14,m+13,1);
196
197 fprintf(fid, 'Line Loop(8) = {6,15,-25,-9};\n');
198 fprintf(fid, 'Ruled Surface(8) = {8};\n');
199 fprintf(fid, 'Transfinite Surface(8) = {%i,%i,%i,%i};\n', m+3,m+12,m+13,1);
200
201 fprintf(fid, 'Line Loop(9) = {14,27,-22,-10};\n');
202 fprintf(fid, 'Ruled Surface(9) = {9};\n');
203 fprintf(fid, 'Transfinite Surface(9) = {%i,%i,%i,%i};\n', m+2,m+14,m+15,m+4);
204
205 fprintf(fid, 'Line Loop(10) = {15,24,-23,-12};\n');
206 fprintf(fid, 'Ruled Surface(10) = {10};\n');
207 fprintf(fid, 'Transfinite Surface(10) = {%i,%i,%i,%i};\n', m+3,m+10,m+11,m+12);
208
209 fprintf(fid, 'Recombine Surface{1,2,3,4,5,6,7,8,9,10}=0;\n'); %This is important,
        it tells Gmsh to attemp to join the default triangles into quadrangles
        (2triangles=1quadrangle)
210
211 %The next lines extrude a small height the 2D surface to have a one-cell-depth
        volume, necessary for finite volume codes. It is not necessary for finite
        elements, but who uses them? :D
212 fprintf(fid, 'j1[] = Extrude {0,0,%.10g} {Surface{1};Layers{1};Recombine;};\n',
        cuerda/10);
213 fprintf(fid, 'j2[] = Extrude {0,0,%.10g} {Surface{2};Layers{1};Recombine;};\n',
        cuerda/10);
214 fprintf(fid, 'j3[] = Extrude {0,0,%.10g} {Surface{3};Layers{1};Recombine;};\n',
        cuerda/10);
215 fprintf(fid, 'j4[] = Extrude {0,0,%.10g} {Surface{4};Layers{1};Recombine;};\n',
        cuerda/10);
```

```
fprintf (fid, 'j5[] = Extrude {0,0,%.10g} {Surface{5};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j6[] = Extrude {0,0,%.10g} {Surface{6};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j7[] = Extrude {0,0,%.10g} {Surface{7};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j8[] = Extrude {0,0,%.10g} {Surface{8};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j9[] = Extrude {0,0,%.10g} {Surface{9};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j10[] = Extrude {0,0,%.10g} {Surface{10};Layers{1};Recombine;};\n',
    cuerda/10);

%Grouping the faces and the volumes...
fprintf (fid, 'Physical Surface("inlet") = {j5[5],j6[5]};\n');
fprintf (fid, 'Physical Surface("outlet") = {j7[4],j8[4],j9[3],j10[3]};\n');
fprintf (fid, 'Physical Surface("airfoil") = {j1[2],j2[2]};\n');
fprintf (fid, 'Physical Surface("walls") =
    {j3[4],j4[4],j5[2],j6[2],j9[4],j10[4]};\n');
fprintf (fid, 'Physical Surface("symmetry") =
    {1,2,3,4,5,6,7,8,9,10,j1[0],j2[0],j3[0],j4[0],j5[0],j6[0],j7[0],j8[0],j9[0],j10[0]};\n');

fprintf (fid, 'Physical Volume("Volumen") =
    {j1[1],j2[1],j3[1],j4[1],j5[1],j6[1],j7[1],j8[1],j9[1],j10[1]};\n');

fclose(fid);

N = N-1;
%In case you wonder, the info is just below:
printf ('The mesh is made of %i linear hexahedra.\n',
    2*(N(1)*(N(2)+N(4))+N(4)*2*N(3)+N(2)*N(3)))
printf
    ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\n')

end



%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%



function Prog = mindist(Ymin,L,N)

%For a line of length L to be discretized in N elements, of which the shortest is
    at the beggining and has a length Ymin, we compute the Prog such that
    L=Ymin*sum(Prog^n, from n=0 to n=N). Ymin can be but is not limited to the
    Ymin defined in the next function.

e = 1;
Prog = 1.001;

while e > 0.001

  Prog_t = Prog +
      ((Prog^4-2*Prog^3+Prog^2)*L+(Prog^3-Prog^2+Prog^N*(Prog-Prog^2))*Ymin)/((Prog-1)*Prog^N*Ymin*N-
  e = abs((Prog_t-Prog)/Prog);
```

```
    Prog = Prog_t;

end

if Prog < 1

    Prog = 1;

end

end



%%%%%%%%%%——%%%%%%%%%%——%%%%%%%%%%——%%%%%%%%%%——%%%%%%%%%%——%%%%%%%%%%



function Ymin = ypar (yplus,cuerda,Re,M,T0,eter)

%This function computes the minimum cell distance from a wall, according to
    equations for turbulent flow over a flat plate at zero incidence. It also
    computes some bonus data to define simulation parameters.

switch eter

    case 'a'
      T = T0/(1+0.2*M^2);
      V = M*sqrt(1.4*287.074*T);
      muT = 1.458e-6*T^1.5/(110.4+T);
      rho = Re*muT/(V*cuerda);
      P = rho*287.074*T;
      P0 = P*(T0/T)^(1.4/0.4);
      printf
          ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\
      printf ('Medium: air\nRe = %g\nChord = %f [m]\nDensity = %g [Kg/m^3]\nDynamic
          viscosity = %g [Kg/(ms)]\nFreestream speed = %f [m/s]\nFreestream Mach =
          %f\nStatic pressure (absolute) = %f [Pa]\nStagnation pressure = %f
          [Pa]\nTemperature = %f [K]\nStagnation temeprature = %f[K]\nY+ = %f\n\n',
          Re,cuerda,rho,muT,V,M,P,P0,T,T0,yplus)

    case 'n'
      T = T0/(1+0.2*M^2);
      V = M*sqrt(1.4*297*T);
      muT = 1.781e-5*(111+300.55)/(111+T)*(T/300.55)^1.5;
      rho = Re*muT/(V*cuerda);
      P = rho*297*T;
      P0 = P*(T0/T)^(1.4/0.4);
      printf
          ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\
      printf ('Medium: nitrogen\nRe = %g\nChord = %f [m]\nDensity = %g
          [Kg/m^3]\nDynamic viscosity = %g [Kg/(ms)]\nFreestream speed = %f
          [m/s]\nFreestream Mach = %f\nStatic pressure (absolute) = %f
          [Pa]\nStagnation pressure = %f [Pa]\nTemperature = %f [K]\nStagnation
          temperature = %f[K]\nY+ = %f\n\n', Re,cuerda,rho,muT,V,M,P,P0,T,T0,yplus)

    case 'h'
```

```
        V = M;
        rho = 1000*(1-(T0+288.9414)/(508929.2*(T0+68.12963))*(T0-3.9863)^2);
        muT = rho*V*cuerda/Re;
        printf
            ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\
        printf ('Medium: water\nRe = %g\nChord = %f [m]\nDensity = %g
            [Kg/m^3]\nDynamic viscosity = %g [Kg/(ms)]\nFreestream speed = %f
            [m/s]\nTemperature = %f [K]\nY+ = %f\n\n', Re,cuerda,rho,muT,V,yplus)

end

Cf = 0.02;

while i<10

    funcion = 4.15*sqrt(Cf)*log10(Re*Cf)+1.7*sqrt(Cf)-1.0;
    derfunc = (4.15*log10(exp(1.0))+0.5*4.15*log10(Re*Cf)+1.7/2.0)/sqrt(Cf);
    fsd = funcion/derfunc;

    if abs(fsd/Cf) <= exp(-5.0)
      break
    end

    Cfo = Cf-fsd;

    if Cfo <= 0.0
      Cf = 0.5*Cf;
    else
      Cf = Cfo;
    end

    i=i+1;

end

%Cfo = (1./(4.15*log10(Re*Cf)+1.7))^2;
%tau = 0.5*rho*V*V*Cf
%aus = sqrt(tau/rho)

Ymin = yplus*muT/(V*sqrt(Cf/2));

printf ('To obtain CFL(max) <= 20 across the whole flowfield, a timestep dt <=
    %.10gs is recomended for transient simulations.\n\n', 20*Ymin/V)
printf ('The following values are recomended to initialize external flowfield
    variables:\nK = %g [m^2/s^2]\nEpsilon = %g [m^2/s^3]\nOmega = %g
    [1/s]\nTurbulent intensity = 6.6667e-7 [%%]\nHydraulic diameter = %f [m]\n\n',
    1e-6*V^2,4.5e-7*V^3/cuerda,0.45*V/cuerda,0.0052164*cuerda)

end
```

Bibliography

[1] *OpenFOAM, the open source CFD toolbox User Guide*. OpenFOAM, 2013.

[2] B Mutlu Sumer and Jorgen Fredsoe. *Hydrodynamics around cylindrical structures*. World Scientific, 2006.