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.

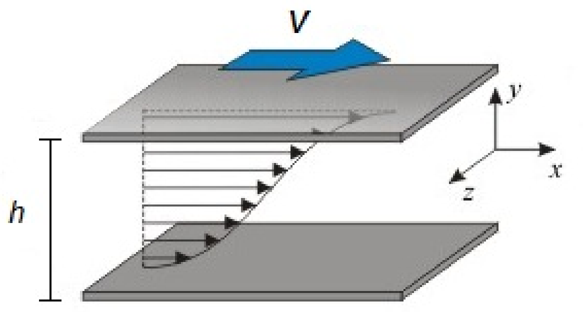


Figure 2.1: Viscous incompressible flow between two plane-parallel plates with relative movement

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 $y = \dfrac{h}{2}$ and $u = 0$ in $y = -\dfrac{h}{2}$, 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.

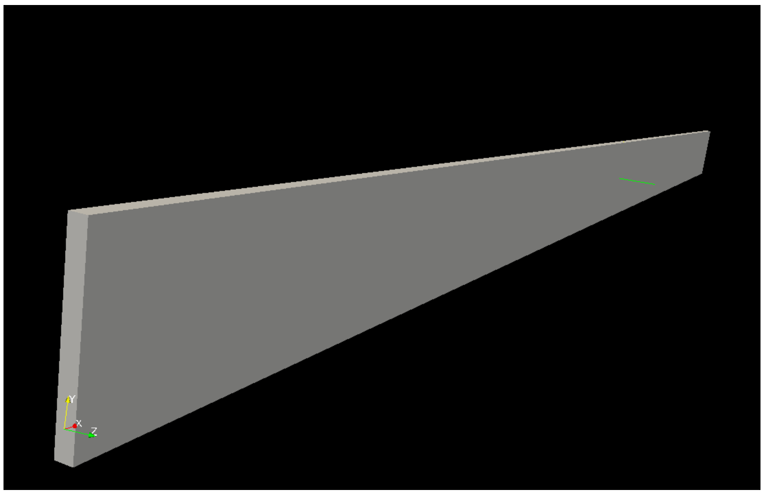


Figure 2.2: Domain of the *Couette flow* case

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)
       );
   }
```

```
    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
(
);

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

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

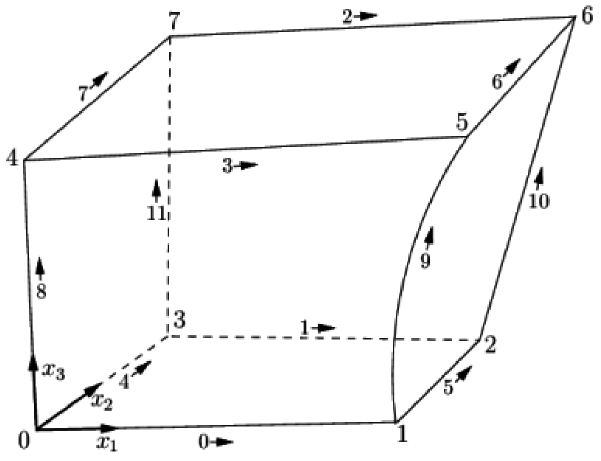


Figure 2.3: Specifications of a single block

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.

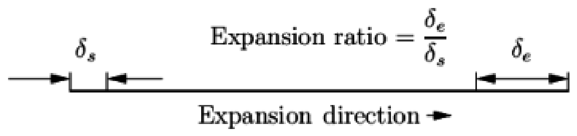


Figure 2.4: Mesh grading along a block edge

- 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.

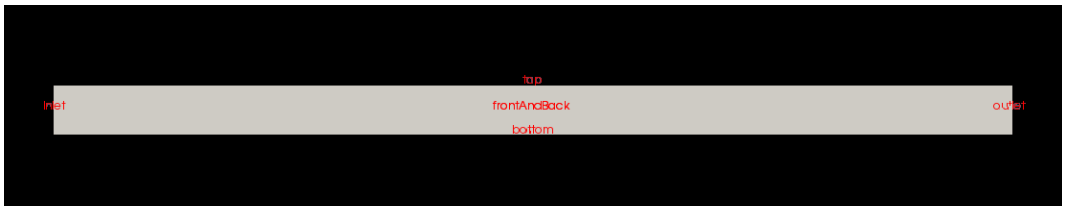


Figure 2.5: Patches defined in the domain of ppWall

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.

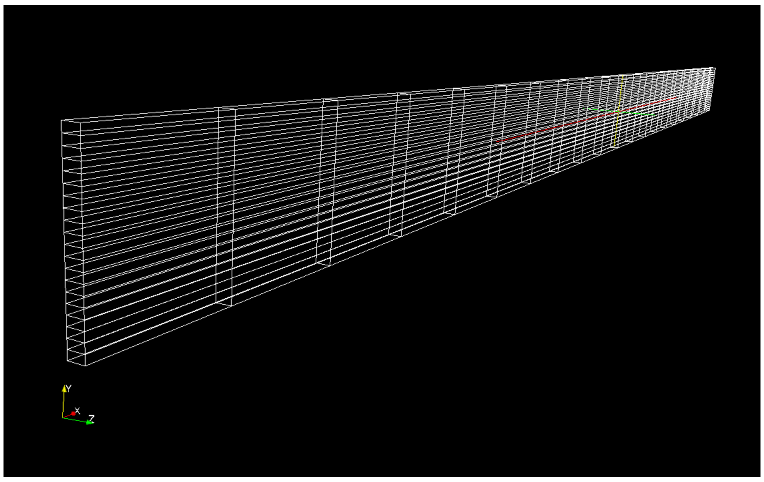


Figure 2.6: Initial mesh of the ppWall case

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:

```
/*--------------------------------*- 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       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    top
    {
        type            zeroGradient;
    }

    bottom
    {
        type            zeroGradient;
    }

    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    frontAndBack
    {
        type            empty;
    }
}

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

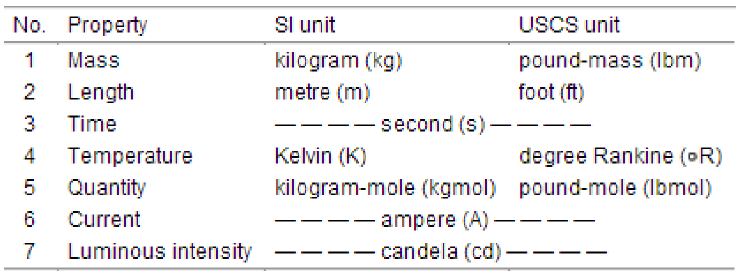
The second file must be named *U* and contains the following instructions:

```
/*--------------------------------*- 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       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
    top
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}


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

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 ($\boldsymbol{U}$) 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 1.5×10^{-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

```
/*--------------------------------*- 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         5;

deltaT          0.005;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

```

```
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*.

```
/*--------------------------------*- 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      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)          Gauss linear;
}

divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
}

laplacianSchemes
{
    default          none;
    laplacian(nu,U)  Gauss linear orthogonal;
    laplacian((1|A(U)),p) Gauss linear orthogonal;
}

interpolationSchemes
{
    default          linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default          orthogonal;
}

fluxRequired
{
    default          no;
    p                ;
}


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

```
/*--------------------------------*- 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      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }

    U
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors      2;
    nNonOrthogonalCorrectors 0;
    pRefCell         0;
    pRefValue        0;
}

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

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:

- ppWall
  - *0*
    - *p*
    - *U*
  - *constant*
    - *polyMesh*
      - *blockMeshDict*
    - *transportProperties*
  - *system*
    - *controlDict*
    - *fvSchemes*
    - *fvSolutions*

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

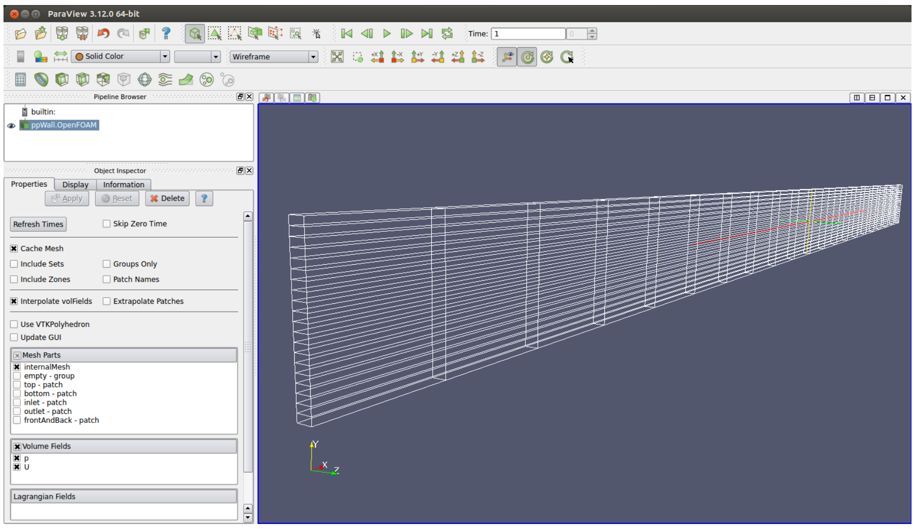
```
Mesh OK.
```

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.



☐ Skip Zero Time
☒ Cache Mesh
☐ Include Sets ☐ Groups Only
☐ Include Zones ☐ Patch Names
☒ Interpolate volFields ☐ Extrapolate Patches
☐ Use VTKPolyhedron
☐ Update GUI

Mesh Parts
☒ internalMesh
☐ empty - group
☐ top - patch
☐ bottom - patch
☐ inlet - patch
☐ outlet - patch
☐ frontAndBack - patch

☒ Volume Fields
☒ p
☒ U

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.

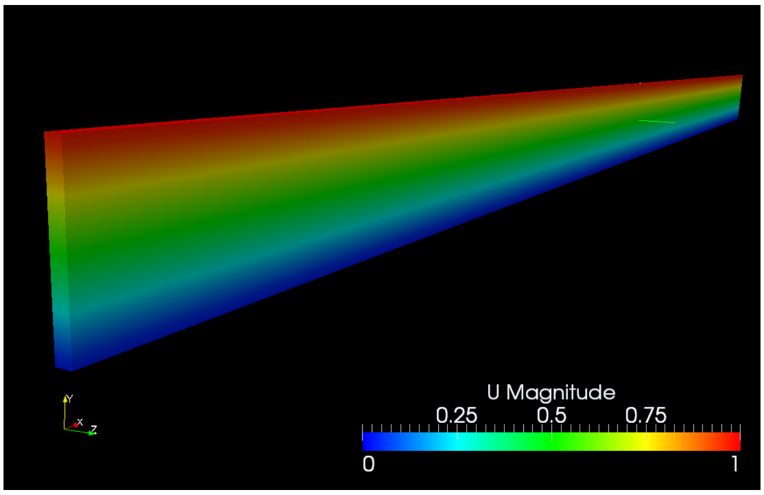


Figure 2.9: Velocity field obtained with icoFoam in the *Couette flow* case (m/s)

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.

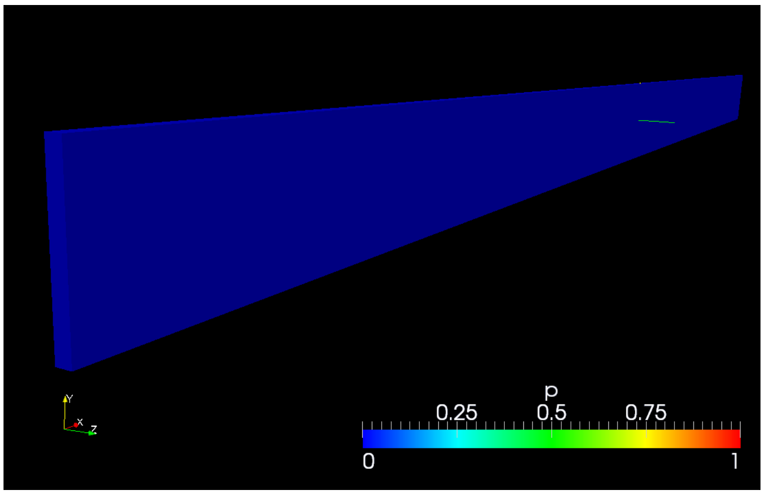


Figure 2.10: Pressure field obtained with icoFoam in the *Couette flow* case (m^2/s^2)

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.

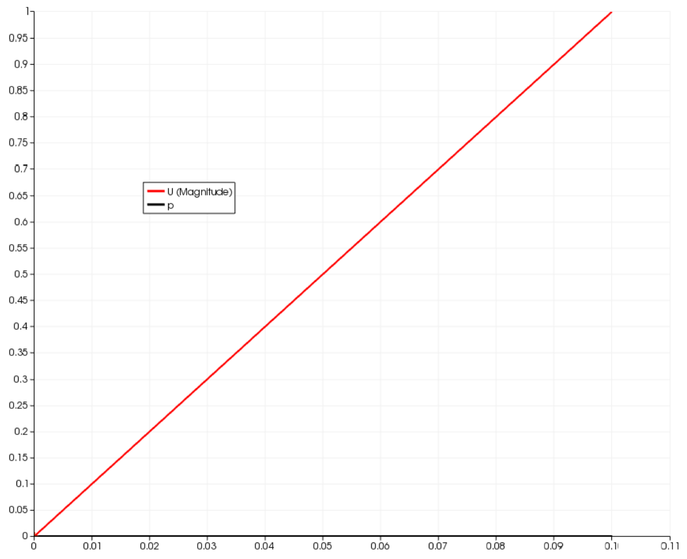


Figure 2.11: Distribution of $|\mathbf{U}|$ and p (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow* case

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.

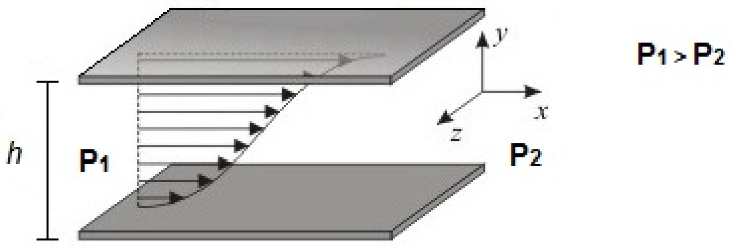


Figure 2.12: Viscous incompressible flow between two plane-parallel plates with a pressure gradient in the x-direction

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];
```

```

internalField          uniform (0 0 0);

boundaryField
{
    top
    {
        type                 fixedValue;
        value                uniform (0 0 0);
    }

    bottom
    {
        type                 fixedValue;
        value                uniform (0 0 0);
    }

    inlet
    {
        type                 zeroGradient;
    }

    outlet
    {
        type                 zeroGradient;
    }

    frontAndBack
    {
       type                  empty;
    }
}


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

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
```

```
    {
        type            fixedValue;
        value           uniform 0;
    }

    frontAndBack
    {
        type            empty;
    }
}

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

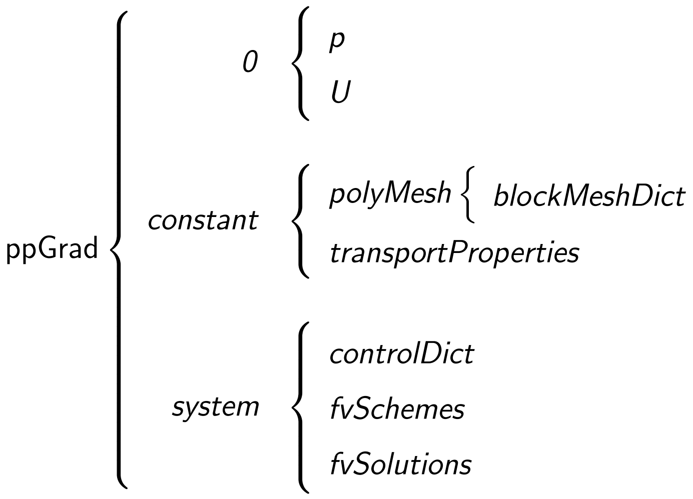
2.4.2.2 Control

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;
```

```

purgeWrite          0;

writeFormat         ascii;

writePrecision      6;

writeCompression    off;

timeFormat          general;

timePrecision       6;

runTimeModifiable   true;

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

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.

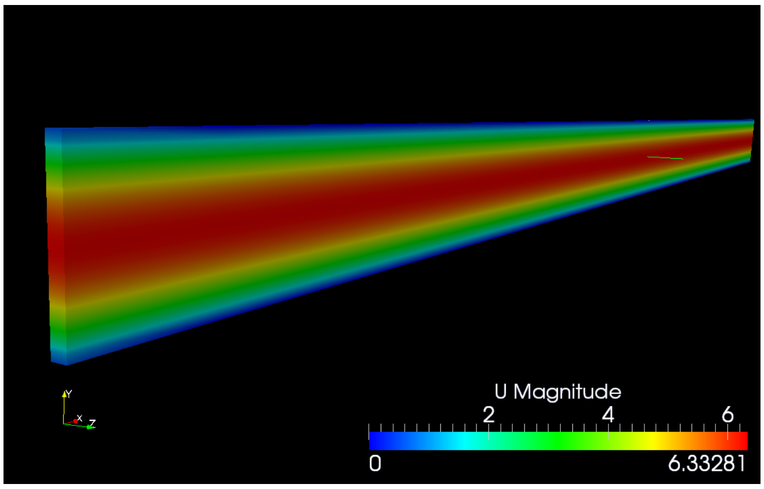


Figure 2.13: Velocity field obtained with icoFoam in the *plane-Poiseuille flow* case (m/s)

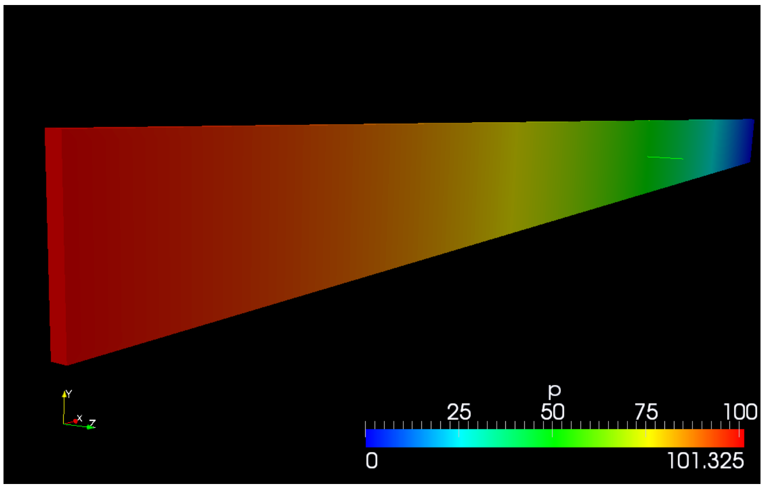


Figure 2.14: Pressure field obtained with icoFoam in the *plane-Poiseuille flow* case (m^2/s^2)

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.

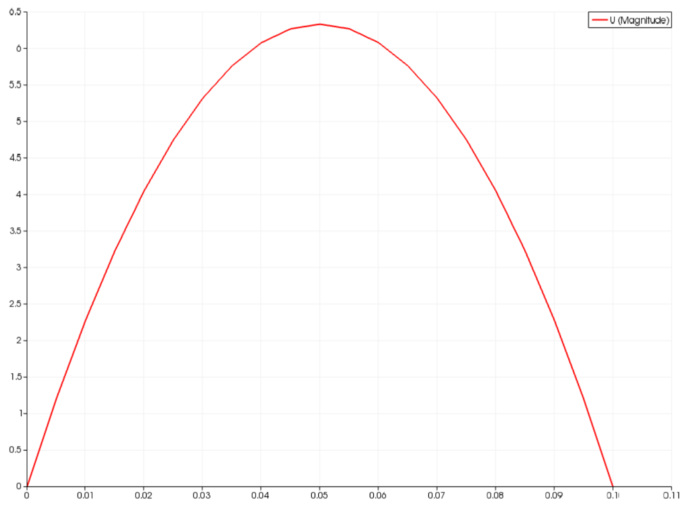


Figure 2.15: Distribution of $|\mathbf{U}|$ (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *plane-Poiseuille flow* case

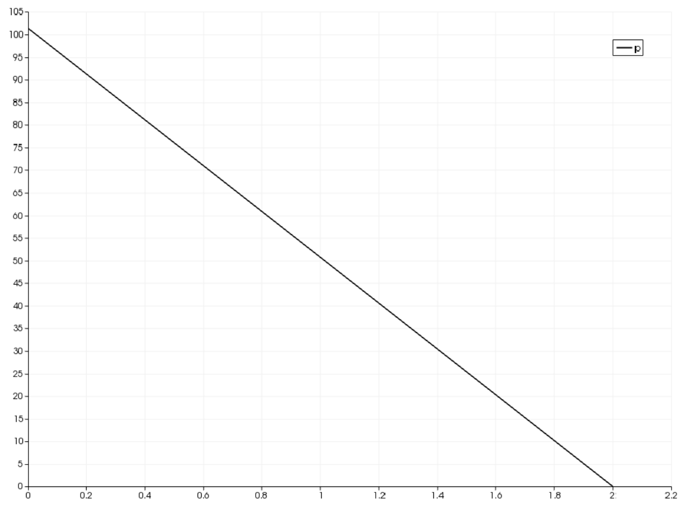


Figure 2.16: Distribution of p (ordinate axis) along the x-axis (abscissa axis) obtained with the icoFoam simulation of the *plane-Poiseuille flow* case

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 ($\frac{\partial p}{\partial x} \neq 0$).

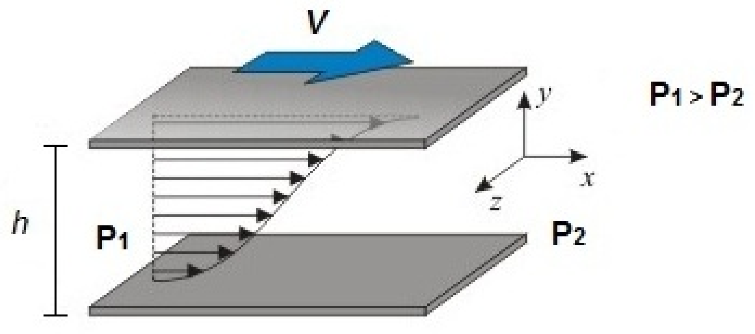


Figure 2.17: Viscous incompressible flow between two plane-parallel plates with relative movement and with a pressure gradient in the x-direction

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      }
```

```

    inlet
    {
       type                zeroGradient;
    }

    outlet
    {
       type                zeroGradient;
    }

    frontAndBack
    {
      type                 empty;
    }
}

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

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.

- ppWallGrad
  - *0*
    - *p*
    - *U*
  - *constant*
    - *polyMesh*
      - *blockMeshDict*
    - *transportProperties*
  - *system*
    - *controlDict*
    - *fvSchemes*
    - *fvSolutions*

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.

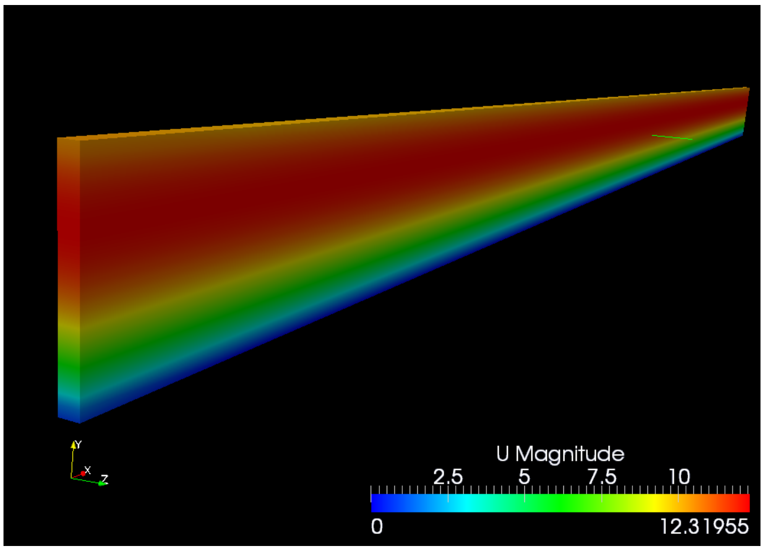


Figure 2.18: Velocity field obtained with icoFoam in the *Couette flow with pressure gradient* case (m/s)

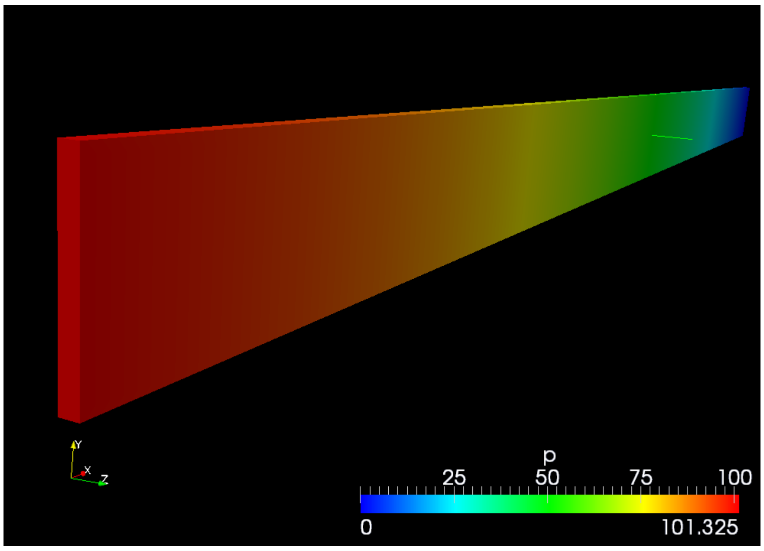


Figure 2.19: Pressure field obtained with icoFoam in the *Couette flow with pressure gradient* case (m^2/s^2)

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.

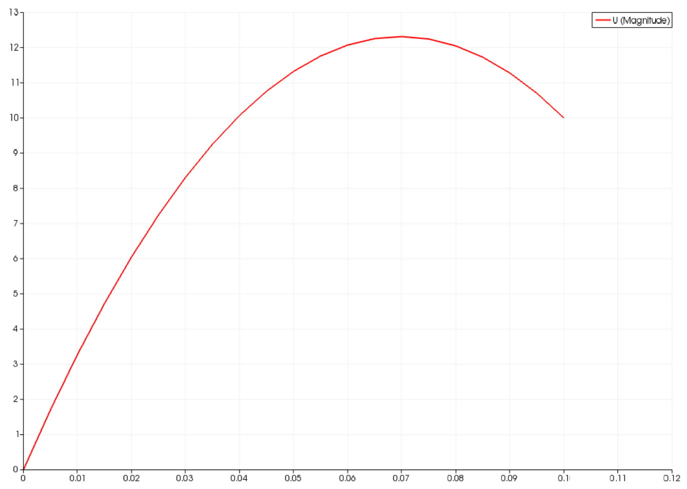


Figure 2.20: Distribution of $|\mathbf{U}|$ (ordinate axis) along the y-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow with pressure gradient* case

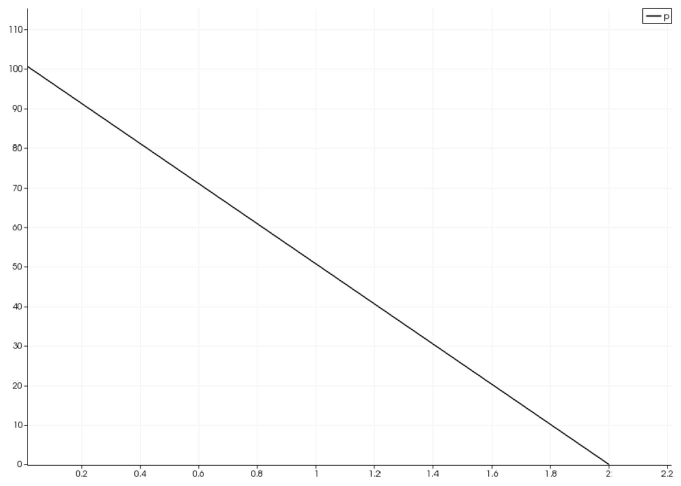


Figure 2.21: Distribution of p (ordinate axis) along the x-axis (abscissa axis) obtained with the icoFoam simulation of the *Couette flow with pressure gradient* case

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.

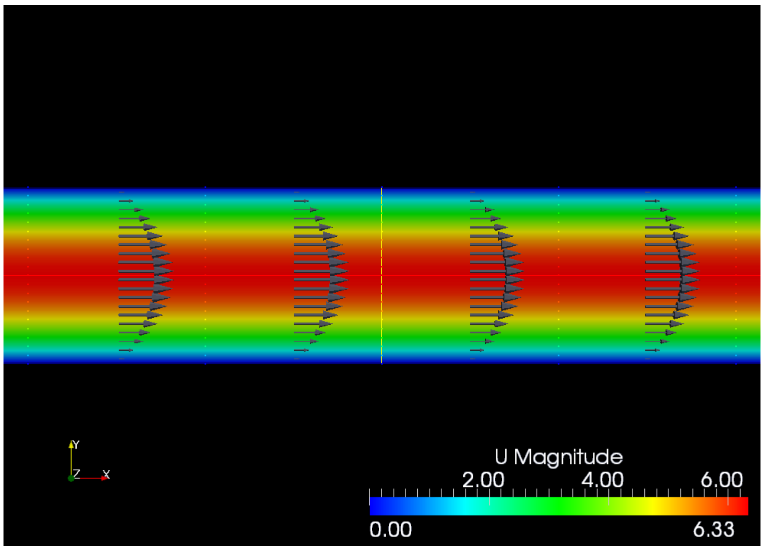


Figure 2.22: Vectors of the flow velocity in the *plane-Poiseuille flow* case (m/s)

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.

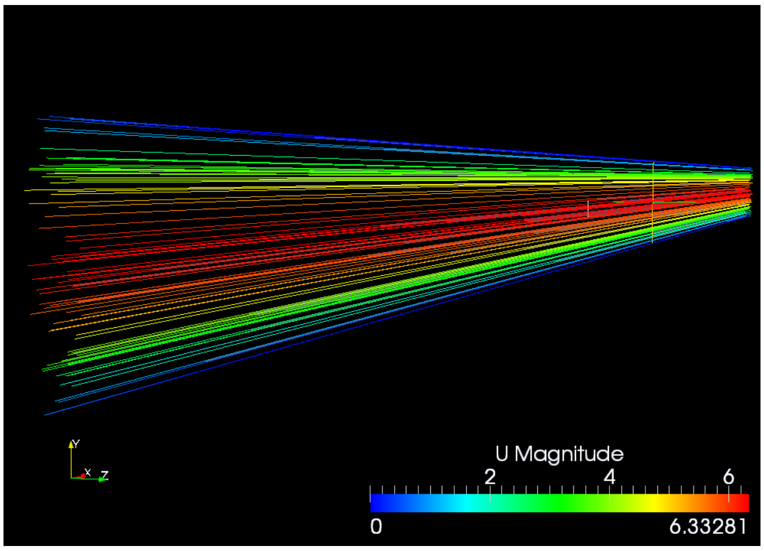


Figure 2.23: Streamlines in the *plane-Poiseuille flow* case (m/s)

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.

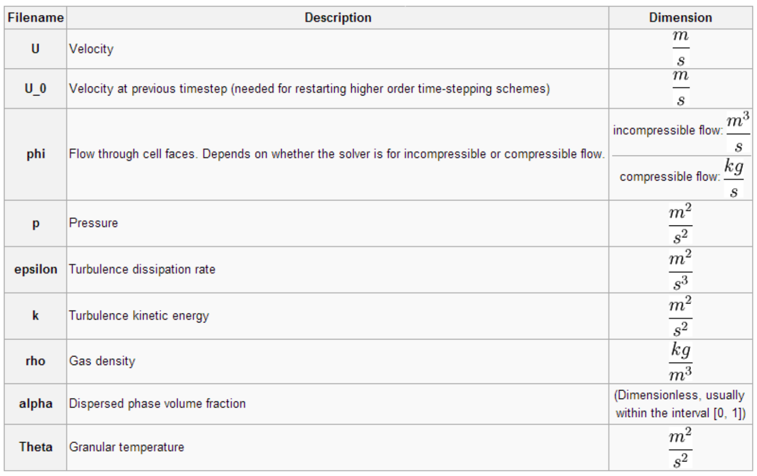
Advice:

> 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;
```

```

stopAt              endTime;

endTime             2.5;

deltaT              0.00005;

writeControl        timeStep;

writeInterval       2000;

purgeWrite          0;

writeFormat         ascii;

writePrecision      6;

writeCompression    off;

timeFormat          general;

timePrecision       6;

runTimeModifiable   true;

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

Caution:

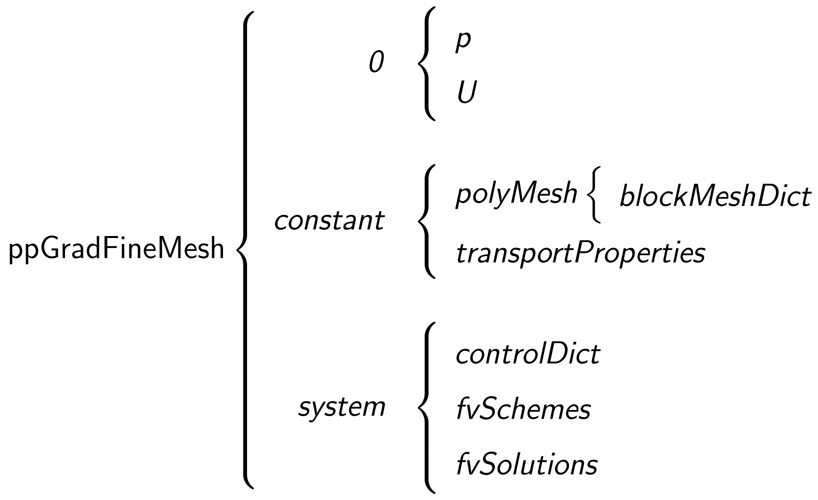
Do not forget to run blockMesh again in order to create the new mesh

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:

```
/*--------------------------------*- 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;


```

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

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:

```
/*--------------------------------*- 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;

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

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

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.

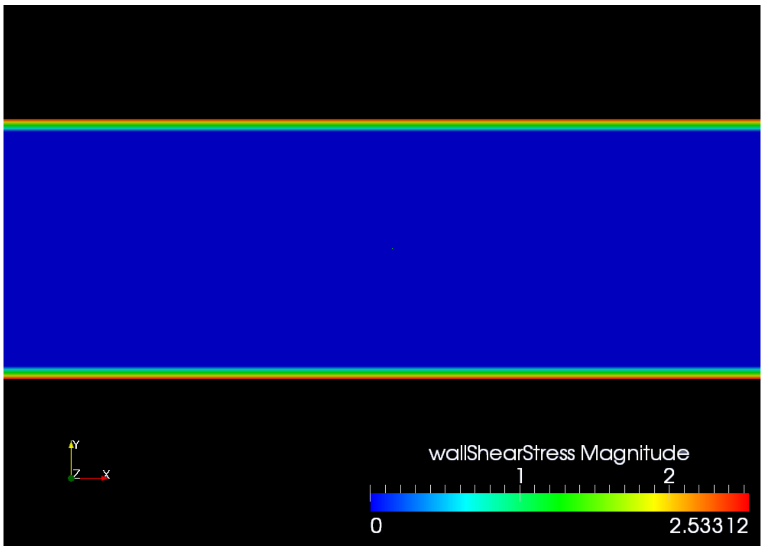


Figure 2.25: Shear stress at the walls of the *plane-Poiseuille flow* case (m^2/s^2)

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

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

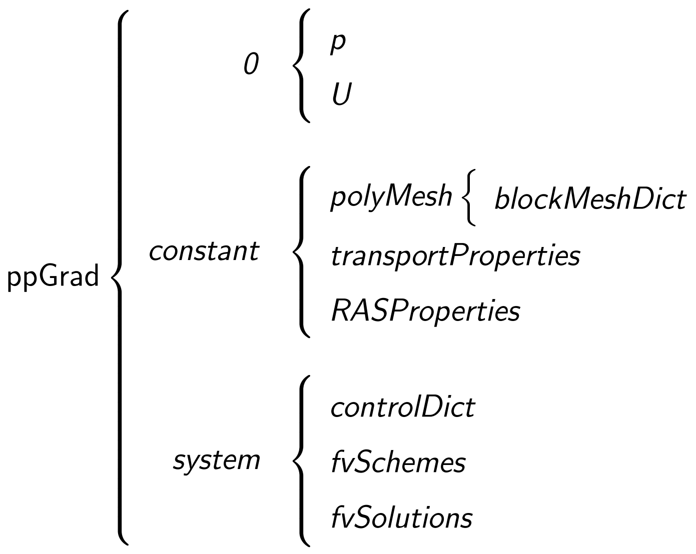
At the top wall, the shear stress is equal but negative, according to the sign convention.

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:

```
/*--------------------------------*- 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)
    (0 1 0)
    (20 1 0)
    (0 0 0.1)
    (20 0 0.1)
    (0 1 0.1)
    (20 1 0.1)
    (0 -1 0)
    (20 -1 0)
    (0 -1 0.1)
    (20 -1 0.1)
);

blocks
(
    hex (0 1 3 2 4 5 7 6) (20 20 1) simpleGrading (1 0.1 1)
    hex (8 9 1 0 10 11 5 4) (20 20 1) simpleGrading (1 10 1)
);

edges
(
);

boundary
(
    movingWall
    {
        type wall;
        faces
        (
             (2 6 7 3)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (8 9 11 10)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (10 4 0 8)
            (4 6 2 0)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (1 3 7 5)
            (9 1 5 11)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (11 5 4 10)
            (5 7 6 4)
            (8 0 1 9)
            (0 2 3 1)
        );
    }
);

mergePatchPairs
(
);

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

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:

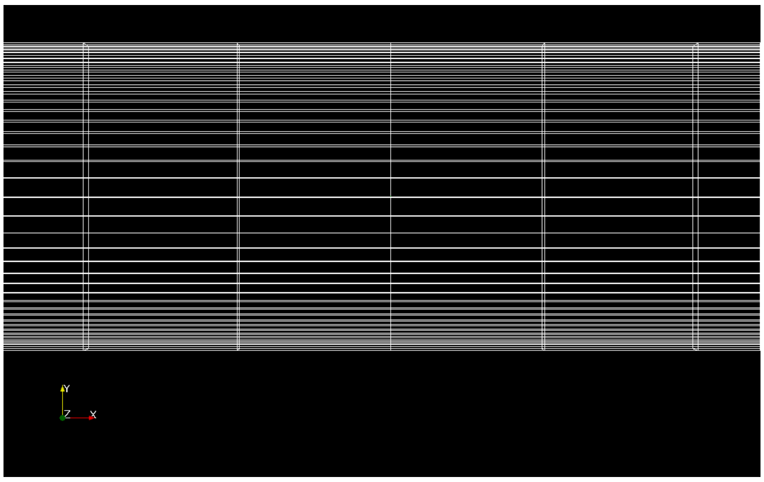


Figure 2.26: Graded mesh used to obtain more accurate values of the wall shear stress in the *plane-Poiseuille flow* case