Membrane Gas Separation Simulation with OpenFOAM: A Short Practical Course

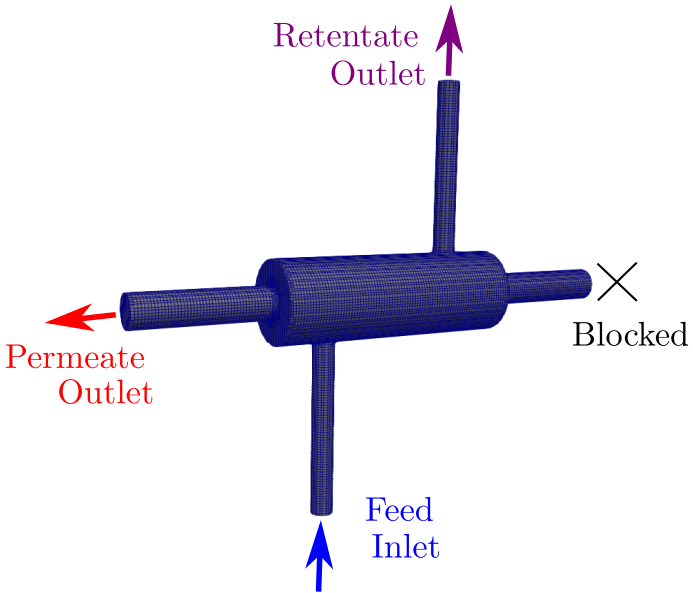
Lecture 01

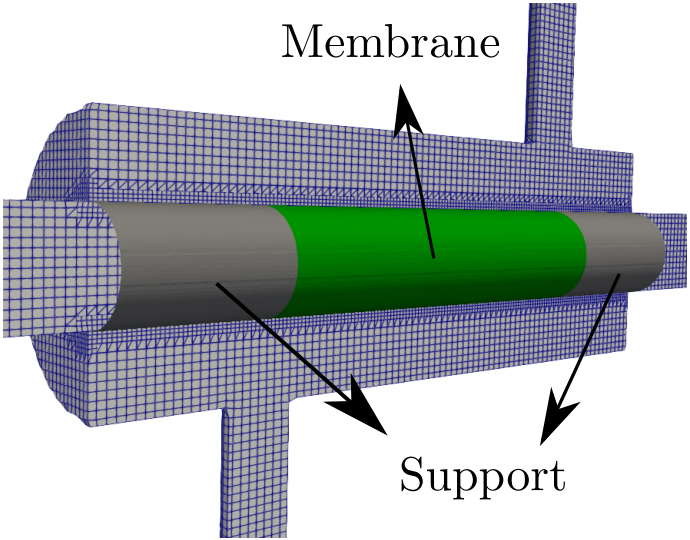
Professor: Adriano Possebon Rosa (aprosa@unb.br)

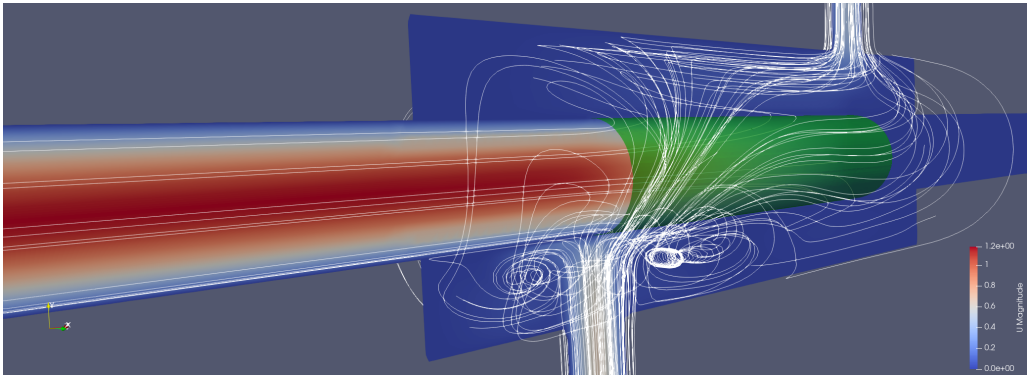
Energy and Envinronemnt Laboratory (lea.unb.br)

University de Brasília

Goal: simulate a membrane gas separation module using Computational Fluid Dynamics (CFD).


Retentate
Outlet
Blocked
Permeate
Outlet
Feed
Inlet


Membrane
Support


1.2e+00
1
0.8
0.6
0.4
0.2
0.0e+00
U Magnitude

Lectures:

- **Lecture 00:** Membrane Gas Separation; Governing Equations; CFD Basics; Finite Volume Method; OpenFOAM; Simulation: Lid-Driven Cavity.
- **Lecture 01:** Governing Equations (Basic Problems); Simulation: Parallel Plates; Mesh Basics; Mesh Generation with Gmsh; Simulation: T-Junction; Baffles.
- **Lecture 02:** Baffles; Set fields; Simulation: T-Junction with Membrane; SnappyHexMesh; Mesh Generation with SnappyHexMesh; Simulation: 3D T-Junction with Membrane and SHM.
- **Lecture 03:** Running the Complete Simulation (Membrane Module); Simulation Setup; Boundary Conditions; Initial Conditions; Simulation and Experiment; Parallelization; Post Process.

Sumário

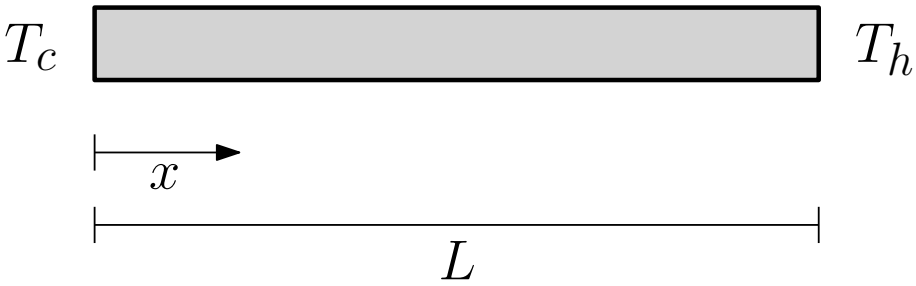
Before we proceed to simulate a membrane gas separation module, let's examine two basic problems:

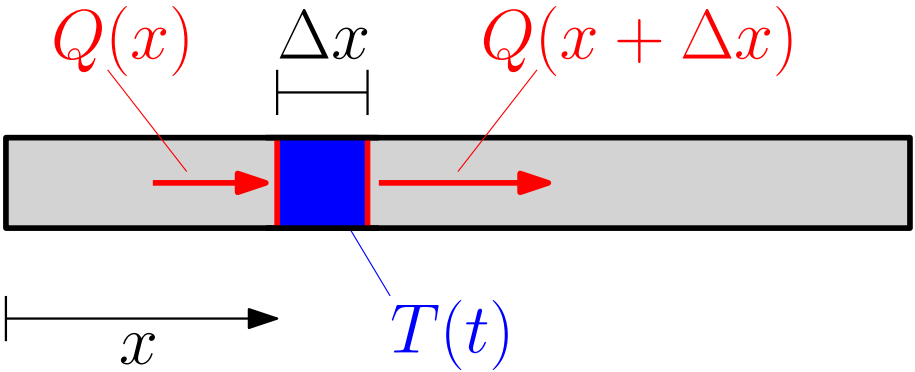
- Heat Conduction in a 1-D Bar
- Flow Between Parallel Plates

Though not our final goal, these problems provide critical insights in:

- Governing PDEs for transport phenomena
- Boundary condition handling
- Concepts directly applicable to membranes

Sumário


T_c
T_h
x
L


$Q(x)$
Δx
$Q(x + \Delta x)$
x
$T(t)$

The First Law of Thermodynamics: the net energy change in a system equals the energy entering minus the energy leaving.

For heat transfer with no work done, this simplifies to:

$$\Delta U = Q$$

where ΔU is the change in internal energy and Q is the heat transferred.

Internal energy is directly linked to temperature:

$$\Delta U = mc\Delta T = mc\left[T(t + \Delta t) - T(t)\right]$$

Here, m is the mass and c is the specific heat capacity.

Consider a small section of the bar with width Δx.

The net heat transfer through this segment is the difference between heat entering ($Q(x)$) and exiting ($Q(x + \Delta x)$):

$$Q = Q(x) - Q(x + \Delta x)$$

Expressed in terms of heat transfer rate $\dot{Q}$ over time Δt:

$$Q = \Delta t \left(\dot{Q}(x) - \dot{Q}(x + \Delta x) \right)$$

The mass of this segment is $m = \rho A \Delta x$, where ρ is density and A is cross-sectional area.

Combining the energy balance $\Delta U = Q$ with the expressions for ΔU and Q, we obtain:

$$\rho c A \Delta x \, \Delta T = \Delta t \left(\dot{Q}(x) - \dot{Q}(x + \Delta x) \right)$$

Or:

$$\rho c A \frac{\Delta T}{\Delta t} = \Delta t \left(\frac{\dot{Q}(x) - \dot{Q}(x + \Delta x)}{\Delta x} \right)$$

Taking the limit as $\Delta t \to 0$ and $\Delta x \to 0$ yields the partial differential form:

$$\frac{\partial \left(\rho c T \right)}{\partial t} = -\frac{1}{A} \frac{\partial \dot{Q}}{\partial x}$$

This links temperature changes to heat flux variations.

Fourier's Law relates heat flux to temperature gradients:

$$\dot{Q} = -kA\frac{\partial T}{\partial x}$$

where k is thermal conductivity.

Substituting into the energy balance gives:

$$\frac{\partial \left(\rho c T\right)}{\partial t} = \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)$$

ρ, c and k constants:

$$\rho c\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}$$

Defining thermal diffusivity $\alpha = k/(\rho c)$, we arrive at the classic heat equation:

$$\boxed{\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}}$$

This PDE governs transient **1D heat conduction.**

The steady-state solution of the heat equation:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

reduces to:

$$\alpha \frac{\partial^2 T}{\partial x^2} = 0 \quad \text{or} \quad \frac{d^2 T}{dx^2} = 0$$

Key observations:

- Temperature becomes purely spatial: $T = T(x)$
- General solution is linear:

$$T(x) = ax + b$$

- Constants a and b are determined by boundary conditions

Now we are going to solve the 1D transient heat conduction equation in OpenFOAM, with different boundary and initial conditions.

All the simulations presented here can be found in the '*simulations*' folder.

Example 0. $L = 0.1\,m$ and $\alpha = 10^{-6}\,m^2/s$.

$$T = T(x,t)$$

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}\,, \quad 0 < x < L\,, \quad t > 0$$

$$T(0,t) = 273\,K\,, \quad T(L,t) = 283\,K\,, \quad t \geq 0$$

$$T(x,0) = 273\,K\,, \quad 0 < x < L$$

Solver in OpenFOAM: laplacianFoam

Commands:

- blockMesh (to generate the mesh).

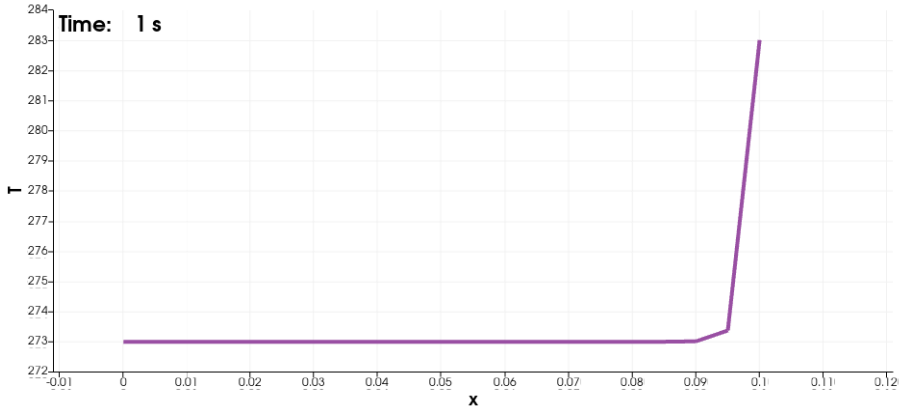
- laplacianFoam (to run the simulation).

- paraFoam (to visualize the results).

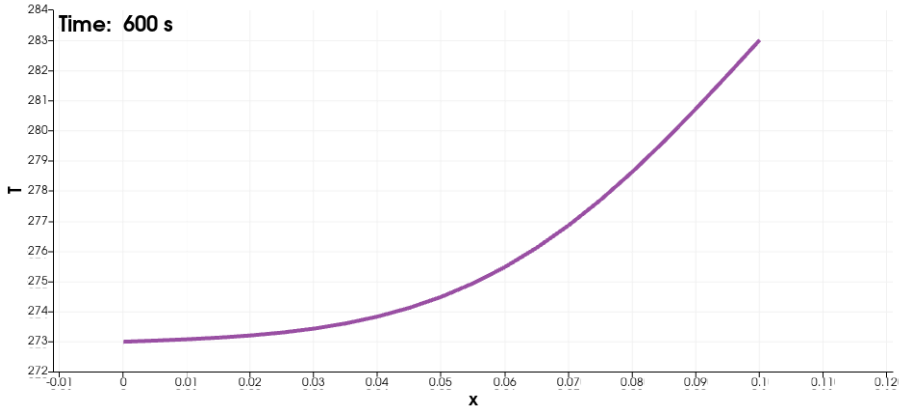
0/T

```
1
2  dimensions      [0 0 0 1 0 0 0];
3
4  internalField   uniform 273.0;
5
6  boundaryField
7  {
8      leftWall
9      {
10         type           fixedValue;
11         value          uniform 273.0;
12     }
13
14     rightWall
15     {
16         type           fixedValue;
17         value          uniform 283.0;
18     }
19
20    ...
21 }
```

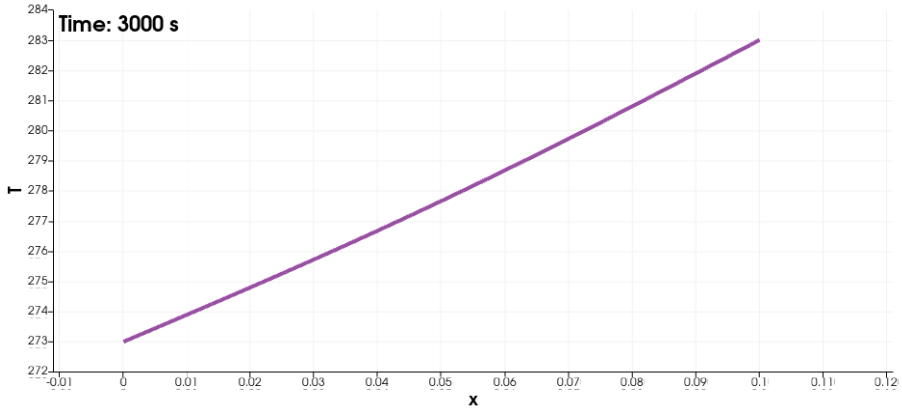
Time: 1 s


Time: 1 s
T
x
284
283
282
281
280
279
278
277
276
275
274
273
272
-0.01
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0.11
0.12

Time: 600 s


Time: 600 s
T
x
284
283
282
281
280
279
278
277
276
275
274
273
272
-0.01
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0.11
0.12

Time: 3000 s


Time: 3000 s
T
x
284
283
282
281
280
279
278
277
276
275
274
273
272
-0.01
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0.11
0.12

Example 1. $L = 0.1\,m$ and $\alpha = 10^{-6}\,m^2/s$.

$$T = T(x,t)$$

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}\,, \quad 0 < x < L\,, \quad t > 0$$

$$T(0,t) = 273\,K\,, \quad \frac{\partial T}{\partial x}(L,t) = 0\,, \quad t \geq 0$$

$$T(x,0) = 283\,K\,, \quad 0 < x < L$$

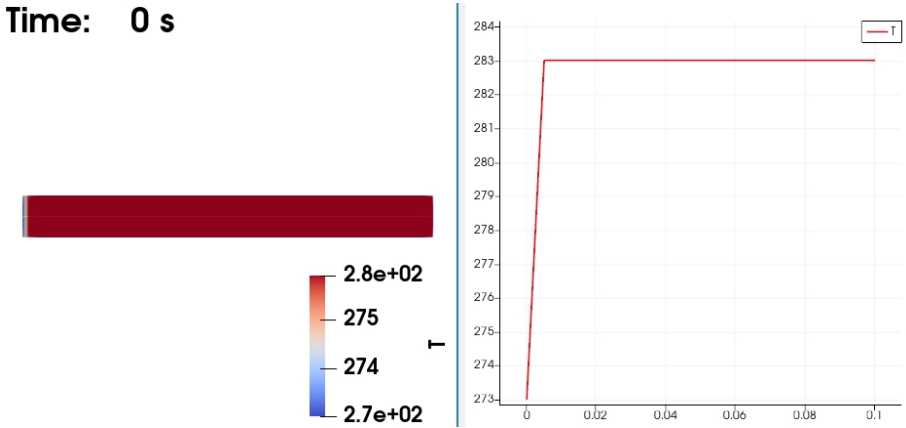
Solver in OpenFOAM: laplacianFoam

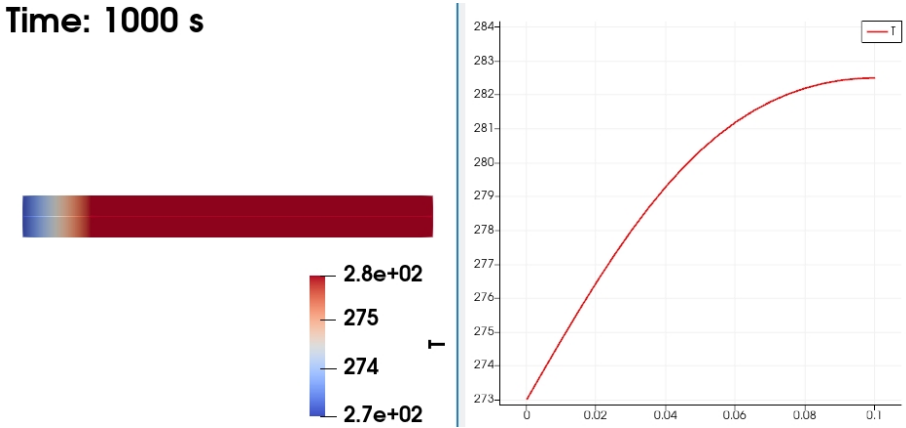
Commands:

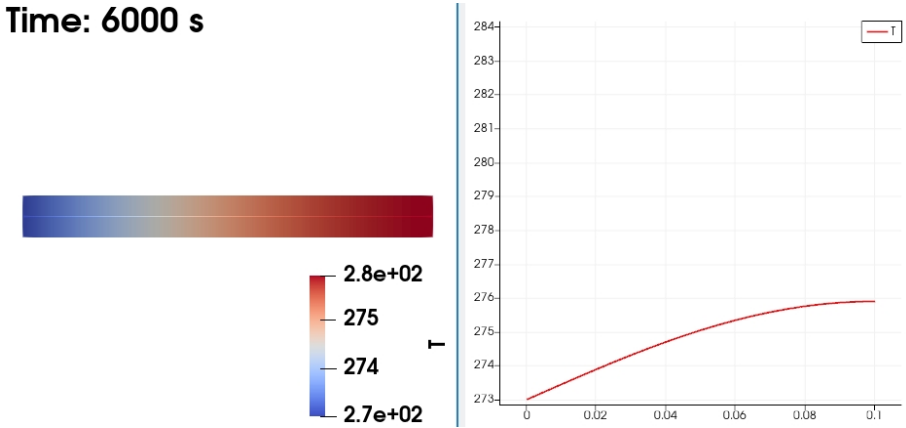
- blockMesh (to generate the mesh).

- laplacianFoam (to run the simulation).

- paraFoam (to visualize the results).


Time: 0 s
2.8e+02
275
274
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
0
0.02
0.04
0.06
0.08
0.1


Time: 1000 s
2.8e+02
275
274
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
0
0.02
0.04
0.06
0.08
0.1


Time: 6000 s
2.8e+02
275
274
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
0
0.02
0.04
0.06
0.08
0.1

Example 2. $L = 0.1\,m$ and $\alpha = 10^{-6}\,m^2/s$.

$$T = T(x,t)$$

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}\,, \quad 0 < x < L\,, \quad t > 0$$

$$T(0,t) = 273\,K\,, \quad T(L,t) = 273\,K\,, \quad t \geq 0$$

$$T(x,0) = 273\,K\,, \quad 0 < x < 0.4L$$
$$T(x,0) = 283\,K\,, \quad 0.4L \leq x \leq 0.6L$$
$$T(x,0) = 273\,K\,, \quad 0.6L < x < L$$

Here we use the OpenFOAM utility 'setFields' to generate the initial condition.

Solver in OpenFOAM: laplacianFoam

Commands:

- blockMesh (to generate the mesh).

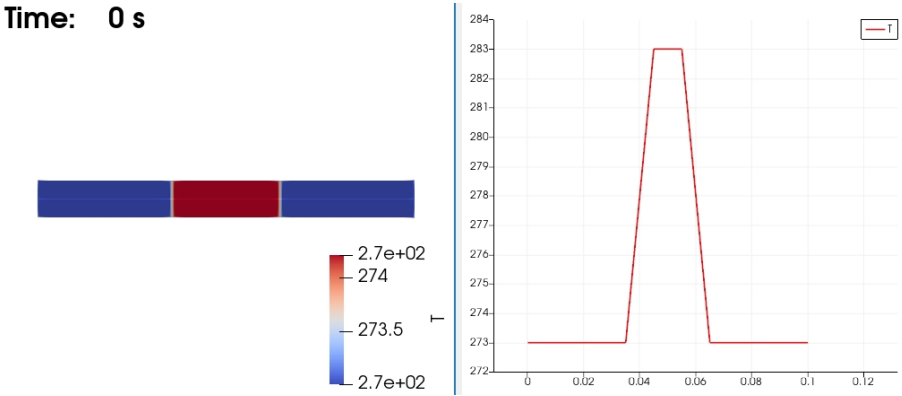
- setFields (to generate the initial condition).

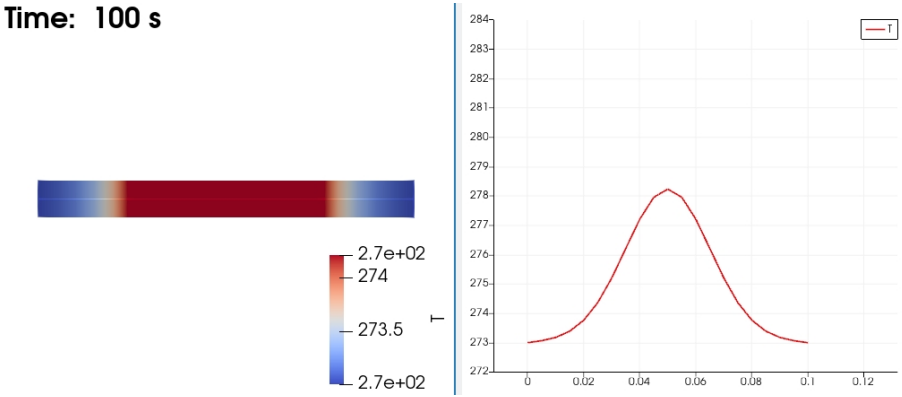
- laplacianFoam (to run the simulation).

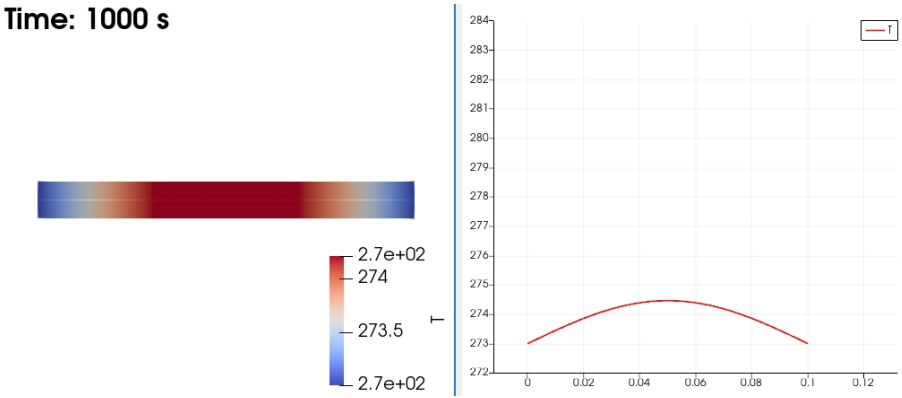
- paraFoam (to visualize the results).

system/setFields

```
1
2  defaultFieldValues
3  (
4      //volScalarFieldValue T 273
5  );
6
7  regions
8  (
9      // Set cell values
10     boxToCell
11     {
12         box (0.04 0 0) (0.06 0.01 0.01);
13
14         fieldValues
15         (
16             volScalarFieldValue T 283.0
17         );
18     }
19
20 );
```


Time: 0 s
2.7e+02
274
273.5
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
272
0
0.02
0.04
0.06
0.08
0.1
0.12


Time: 100 s
2.7e+02
274
273.5
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
272
0
0.02
0.04
0.06
0.08
0.1
0.12


Time: 1000 s
2.7e+02
274
273.5
2.7e+02
T
T
284
283
282
281
280
279
278
277
276
275
274
273
272
0
0.02
0.04
0.06
0.08
0.1
0.12

1D Heat conduction equation:

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\alpha \frac{\partial T}{\partial x}\right)$$

3D Heat conduction equation:

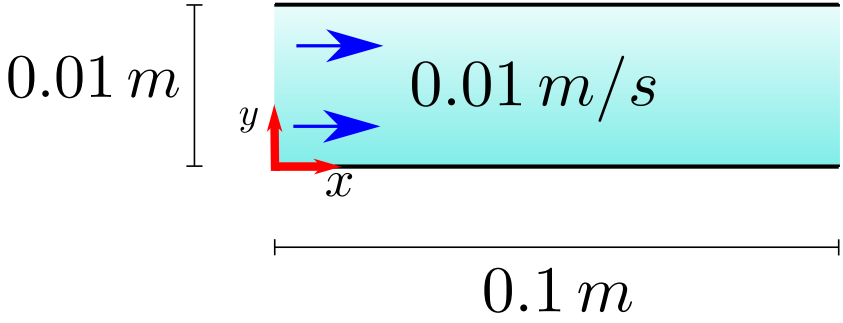
$$\frac{\partial T}{\partial t} = \nabla \cdot (\alpha \nabla T)$$

OpenFOAM code (laplacian.C):

```
while (simple.correctNonOrthogonal())
{
    fvScalarMatrix TEqn
    (
        fvm::ddt(T) - fvm::laplacian(DT, T)
     ==
        fvModels.source(T)
    );
```

Sumário

Let's now study the behavior of an incompressible Newtonian fluid in a flow between two parallel plates. This is a 2D problem.



Governing equations. Newtonian and incompressible flow.

Mass conservation (continuity equation):

$$\nabla \cdot \mathbf{U} = 0 \ ,$$

Navier-Stokes (momentum):

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) = -\nabla \left(\frac{P}{\rho} \right) + \nabla \cdot (\nu \nabla \mathbf{U}) \ ,$$

We want to find the pressure $p(x, y, t) = P(x, y, t)/\rho$ and the velocity $\mathbf{U}(x, y, t)$.

Now we consider fluid inflow and outflow.

We will modify the boundary conditions for p and U to implement inflow/outflow.

The initial setup is based on the Lid-Driven Cavity case from **Lecture 00**.

The following files need adjustment:

- The mesh generation file (*blockMeshDict*)
- The initial/boundary condition files (*p* and *U* in the *0* folder)
- The controlDict file

We'll start with the ***blockMeshDict***.

system/blockMeshDict

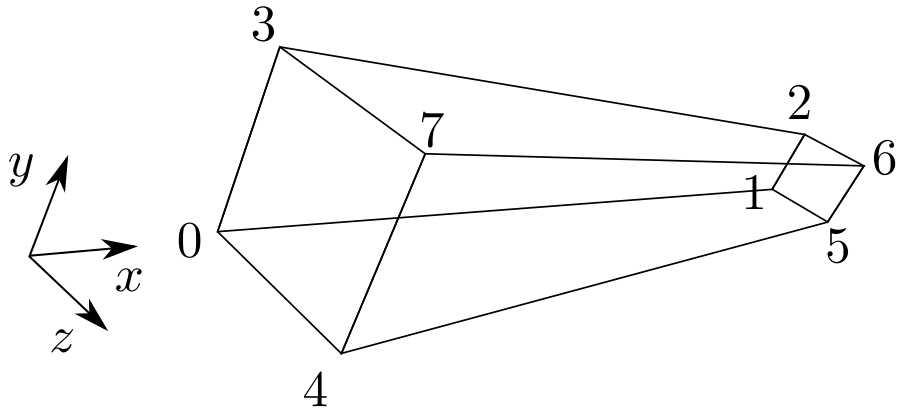
```
convertToMeters 0.01;

vertices
(
    (0 0 0)   // 0
    (10 0 0)  // 1
    (10 1 0)  // 2
    (0 1 0)    // 3
    (0 0 0.1) // 4
    (10 0 0.1)   // 5
    (10 1 0.1) // 6
    (0 1 0.1) // 7
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (100 10 1) simpleGrading (1 1 1)
);
```

```
19 boundary
20 (
21     inlet
22     {
23         type patch;
24         faces
25         (
26             (0 4 7 3)
27         );
28     }
29     outlet
30     {
31         type patch;
32         faces
33         (
34             (2 6 5 1)
35         );
36     }
```

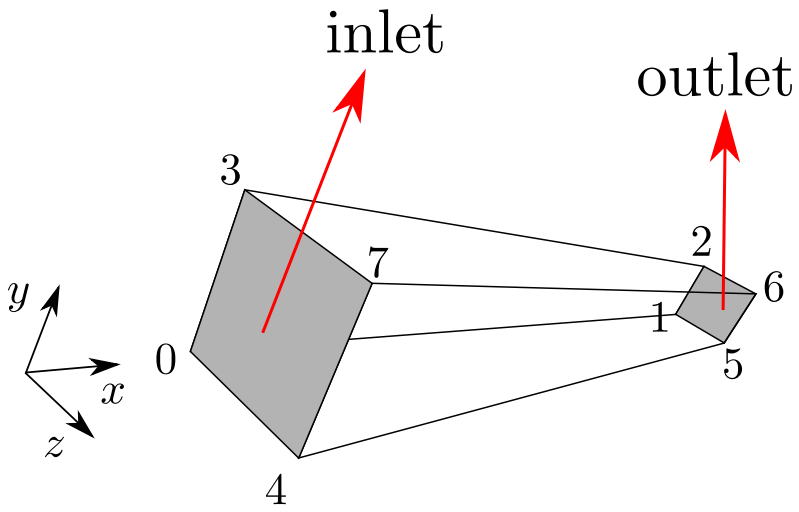
```
37      topWall
38      {
39          type wall;
40          faces
41          (
42              (6 2 3 7)
43          );
44      }
45      bottomWall
46      {
47          type wall;
48          faces
49          (
50              (0 1 5 4)
51          );
52      }
```

```
53      frontAndBack
54      {
55          type empty;
56          faces
57          (
58          (0 3 2 1)
59          (4 5 6 7)
60          );
61      }
62  );
```

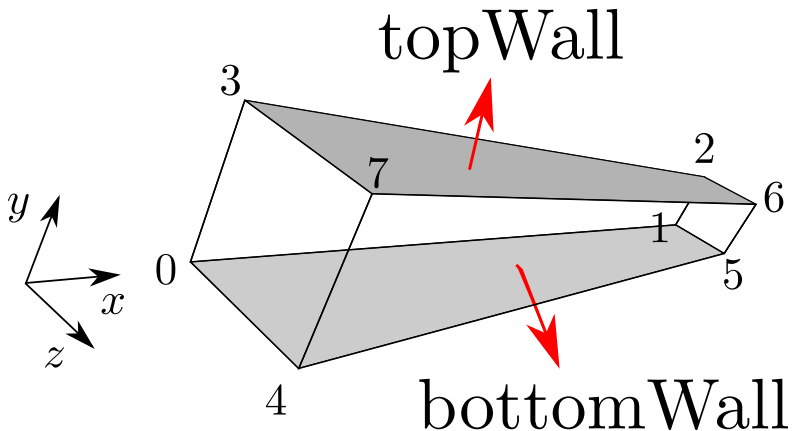
Representation of the points.



Representation of the inlet and outlet faces.

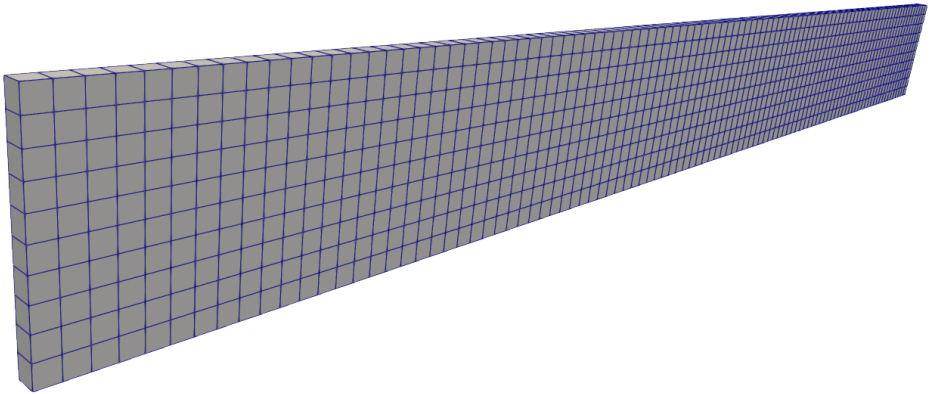


Representation of the top and bottom faces.



Execute the command

$: blockMesh



With the mesh prepared, we proceed now to configure the boundary conditions.

At the walls, we apply the no-slip condition for velocity (zero velocity) and zero normal gradient for pressure.

At the inlet, we specify a fixed velocity value while maintaining a zero normal pressure gradient.

At the outlet, we set a fixed pressure value and apply a zero normal gradient condition for velocity.

0/U

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

internalField uniform (0 0 0);

boundaryField
{
   inlet
   {
      type           fixedValue;
      value          uniform (0.01 0
          0);
   }
   outlet
   {
      type           zeroGradient;
   }
   topWall
   {
      type           noSlip;
   }
   bottomWall
   {
      type           noSlip;
   }
   frontAndBack
   {
      type           empty;
   }
}
```

0/p

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

internalField  uniform 0;

boundaryField
{
   inlet
   {
      type           zeroGradient;
   }
   outlet
   {
      type           fixedValue;
      value          uniform 0;
   }
   topWall
   {
      type           zeroGradient;
   }
    bottomWall
   {
      type           zeroGradient;
   }
   frontAndBack
   {
      type           empty;
   }
}
```

Examine the simulation parameters in the `constant` and `system` folders.

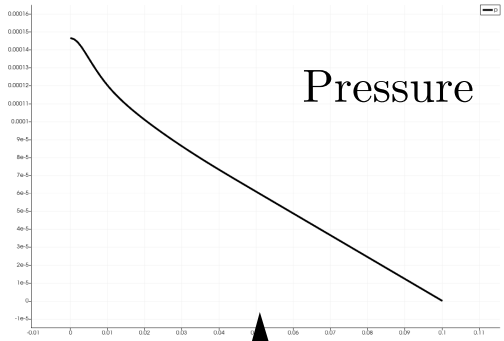
Then execute these commands:

`$: foamRun` (run the simulation)

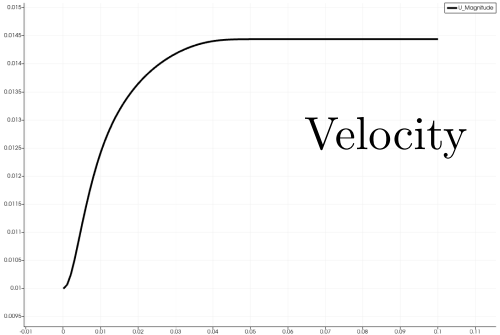
`$: paraFoam` (visualize results in ParaView)


1.4e-02
0.01
0.005
0.0e+00
U Magnitude

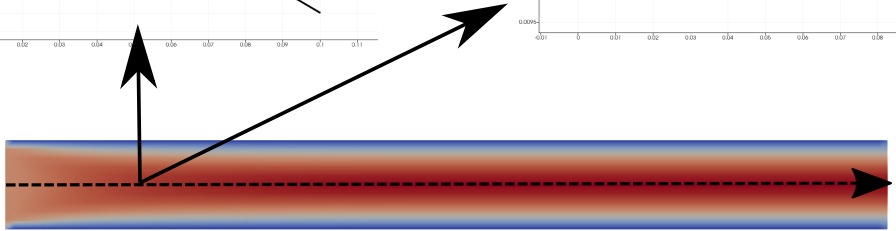

1.7e-04
0.0001
5e-5
0.0e+00
p



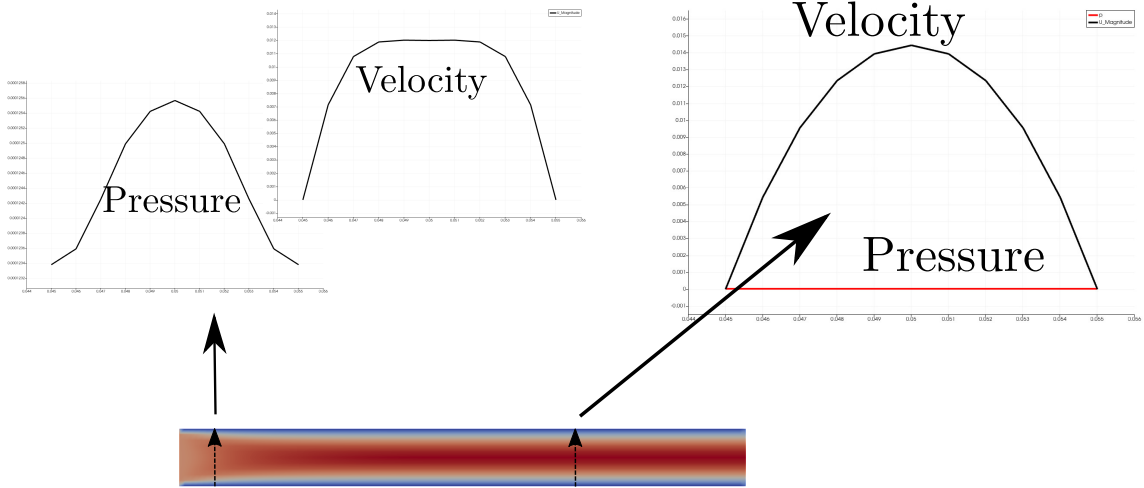
Pressure



Velocity



$t = 4\,s$



$t = 4\,s$

We can compare our results with the analytical solution for the 2D, permanent and fully developed flow.

Mass Conservation (continuity):

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

Navier-Stokes (momentum):

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

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

Seeking solutions for: $p(x, y, t)$ and $\mathbf{U}(x, y, t) = (u(x, y, t), v(x, y, t))$

Assumptions.

Parallel Plates Geometry:

- Plates separated by distance D in y-direction
- Flow driven by pressure gradient in x-direction

Flow Conditions:

- Steady state: $\frac{\partial}{\partial t} = 0$
- Fully developed: $\frac{\partial u}{\partial x} = 0$
- No-slip boundaries: $u(y = 0) = u(y = D) = 0$
- $v = 0$ everywhere (results in $p = p(x)$)

Analytical Solution

Simplified Governing Equation:

$$\mu \frac{d^2u}{dy^2} = \frac{dp}{dx}$$

Dynamic viscosity: μ. Kinematic viscosity: $\nu = \mu/\rho$.

Boundary Conditions:

$$u(0) = 0 \quad \text{and} \quad u(D) = 0$$

Velocity Profile Solution:

$$u(y) = \frac{1}{2\mu}\left(-\frac{dp}{dx}\right)(Dy - y^2)$$

Key Features:

- Parabolic velocity profile
- Maximum velocity at centerline:

$$u_{max} = \frac{D^2}{8\mu}\left(-\frac{dp}{dx}\right)$$

- Linear pressure variation along x

Sumário

To run a CFD simulation, we need to **discretize** (divide) the domain (geometry) into small **cells** (elements). This structure is called a **mesh** (or grid).

The Navier-Stokes equations are solved within these cells.

The mesh replaces the continuous domain with a **discrete** one, composed of cells bounded by **faces**.

A discretized domain must be able to resolve the **physics** of the problem and capture the **geometric details**.

Mesh generation is a crucial step in CFD, as the mesh affects the **accuracy** and **efficiency** of the solution.

CFD users spend more than 50% of their time working on mesh generation.

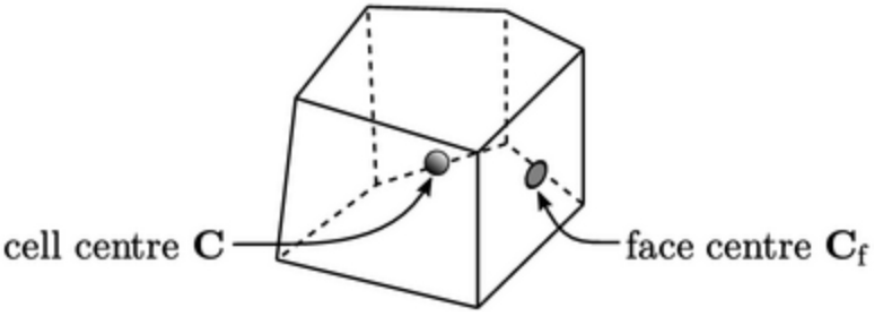
For a mesh to be valid, it must cover the entire computational domain with cells. Additionally, all cells must be **closed and convex**, with their centroid inside the cell.

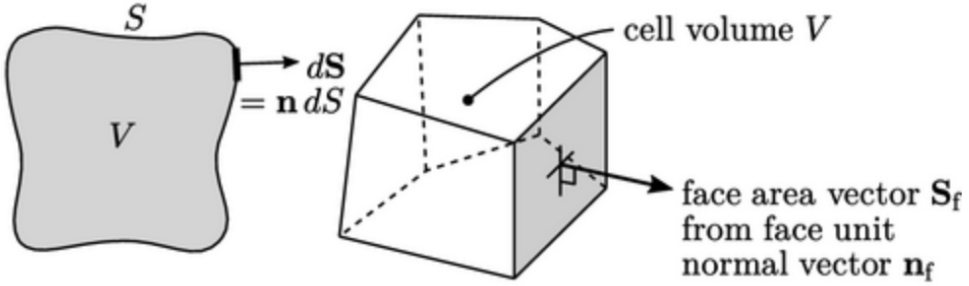
Let's review some definitions.

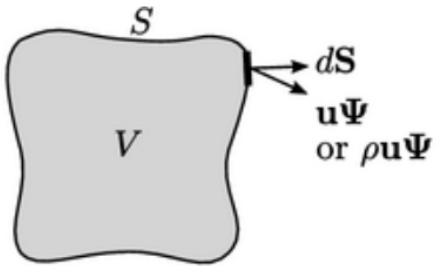
Each cell has a **center** and **faces** enclosing it.

Internal faces connect cells. **Faces on the domain boundaries** define the **boundary conditions** of the problem. (Boundary surfaces are called *patches* in OpenFOAM.)

The values of variables (pressure, velocity, temperature, etc.) are stored at the cell center. To obtain these values at the faces, we need to interpolate.


cell centre $\mathbf{C}$
face centre $\mathbf{C}_\mathrm{f}$


S
V
dS
= n dS
cell volume V
face area vector $\mathbf{S}_f$
from face unit
normal vector $\mathbf{n}_f$


S
V
dS
uΨ
or ρuΨ

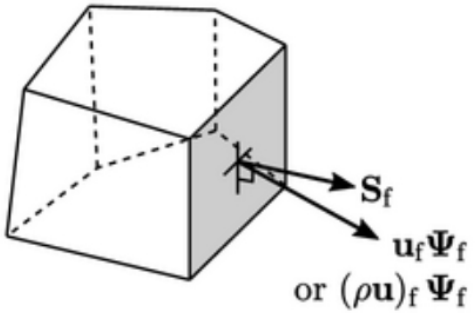
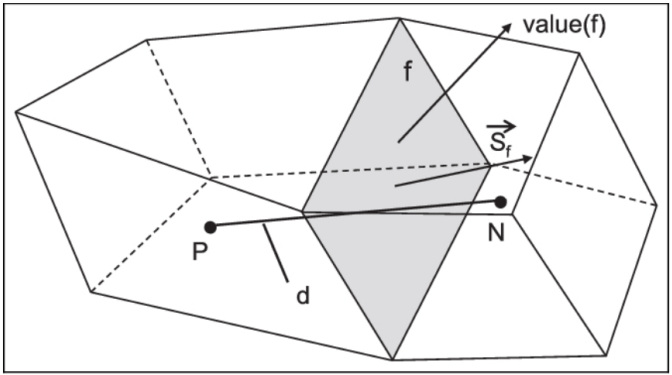
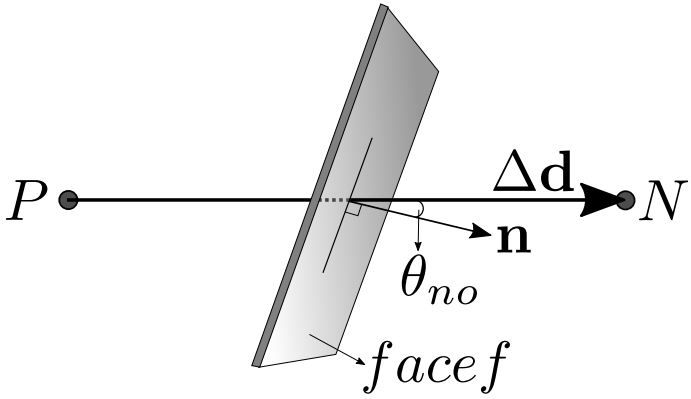
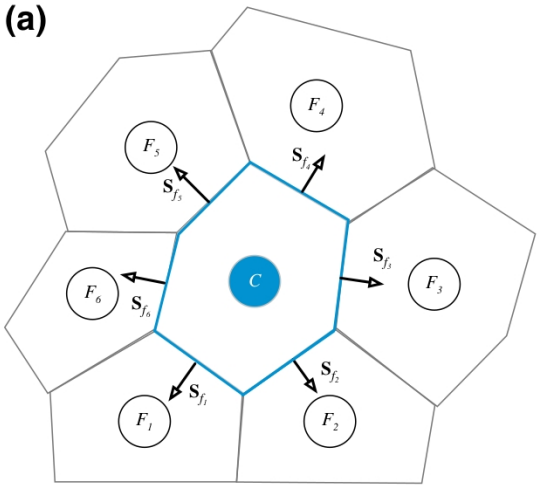

S_f
u_f Ψ_f
or (ρu)_f Ψ_f

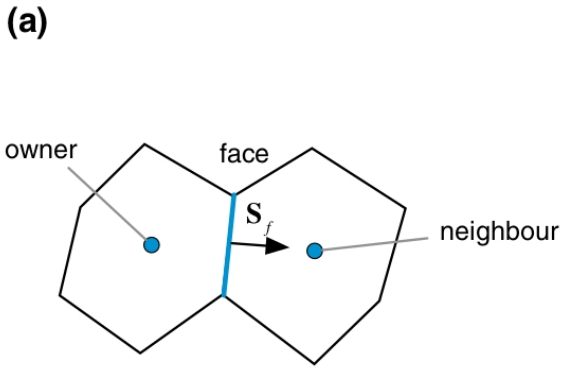
Diagram with two cells. One cell has its center at P and the other at N. The area vector $\vec{S}_f$ is centered at the face center and points outward from the cell (in this case, cell P is the cell under consideration). The vector $\vec{d}$ connects the cell centers.

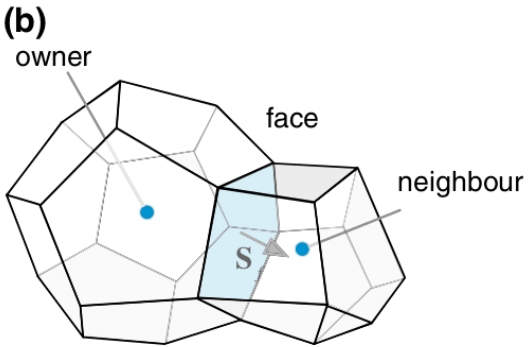


The angle between the vector connecting the centers and the face normal is the **non-orthogonality angle**.



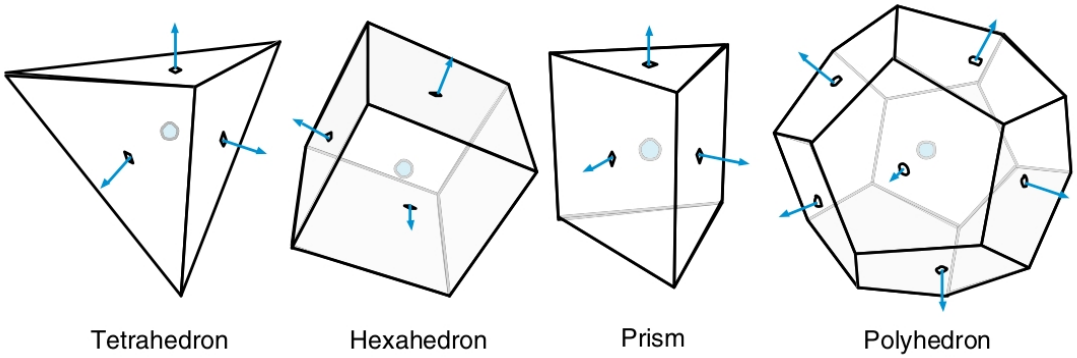

(a)
F_4
F_5
$\mathbf{S}_{f_4}$
$\mathbf{S}_{f_5}$
$\mathbf{S}_{f_3}$
C
F_3
F_6
$\mathbf{S}_{f_6}$
$\mathbf{S}_{f_1}$
$\mathbf{S}_{f_2}$
F_1
F_2


(a)
owner
face
$\mathbf{S}_f$
neighbour


(b)
owner
face
neighbour
S

In **2D**, the most common geometries for the cells are **triangles** and **quadrilaterals**.

In **3D**, we have **tetrahedrons, hexahedrons, prisms, and polyhedrons.**



Meshes: Structured and Unstructured.

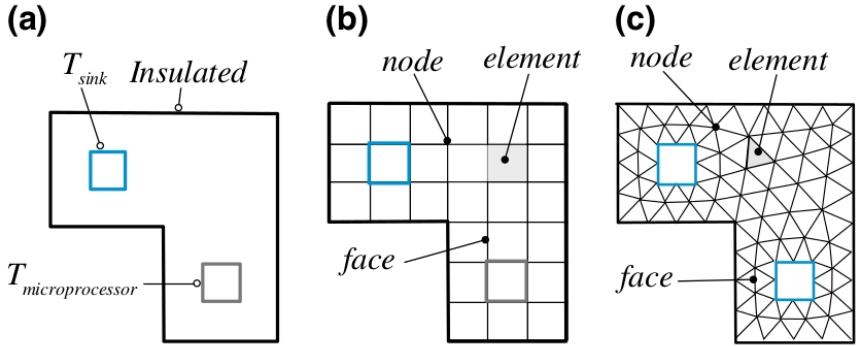


Fig. 6.1 **a** Domain of interest, **b** domain discretized using a uniform grid system, and **c** domain discretized using an unstructured grid system with triangular elements

Structured.

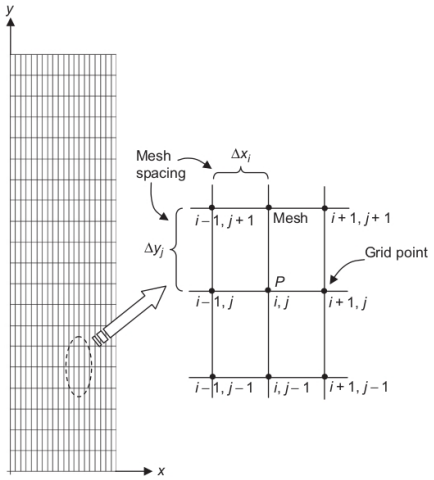


FIG. 4.1 A uniform rectangular mesh.

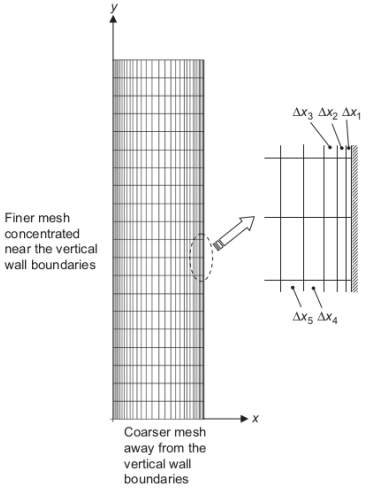


FIG. 4.2 A nonuniform rectangular mesh.

Structured.

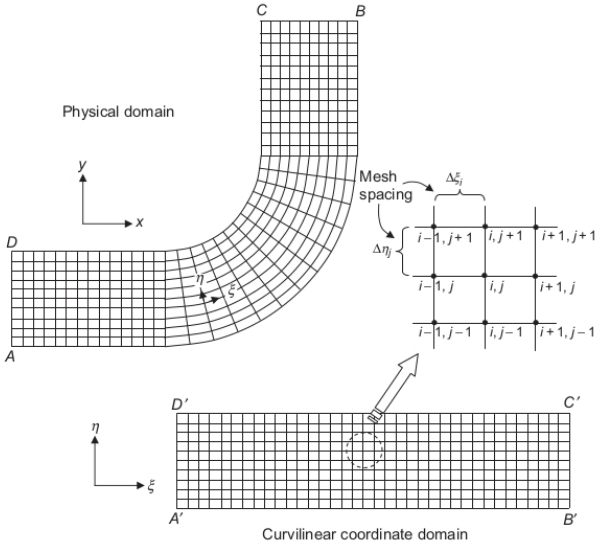


FIG. 4.4 An example of a body-fitted or curvilinear mesh for the 90° bend geometry and corresponding computational geometry.

Unstructured.

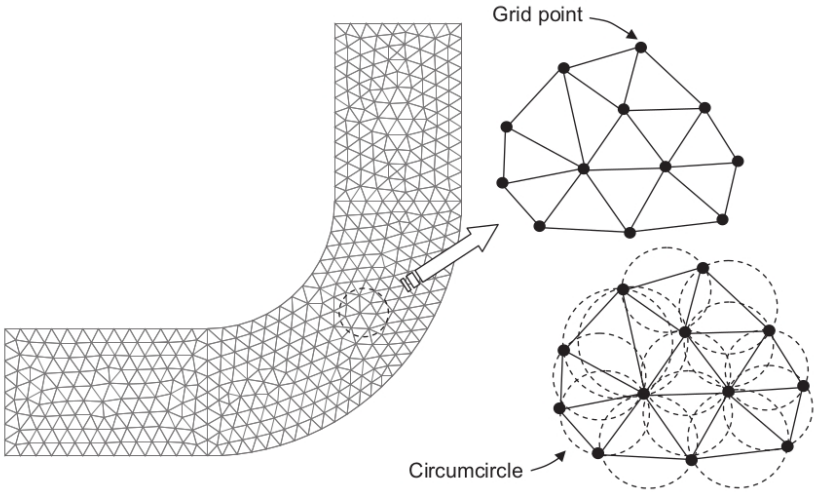


FIG. 4.5 An example of a triangular mesh for the 90° bend geometry.

Unstructured.

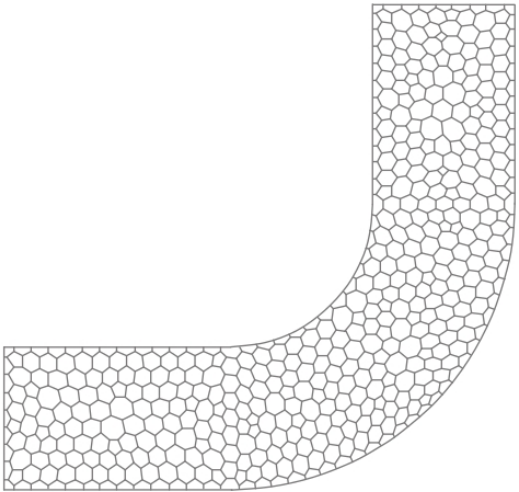


FIG. 4.6 A grid consisting of polyhedral cells of the 90° bend geometry.

Structured Meshes.

Organized and regular structure. These meshes are typically used for simple geometries. Each node (or point) is connected to a fixed number of other nodes in the mesh.

The main characteristic of structured meshes is that the relationships between nodes are known in advance and can be easily described in terms of indices.

Advantages of Structured Meshes:

- Easy generation, especially for simple geometries;
- Computationally efficient for certain types of problems;
- Generally better numerical stability for some differential equations.

Disadvantages of Structured Meshes:

- Difficult to generate for complex geometries;
- Inefficient in terms of computational resources for irregular geometries;
- Require significant adjustments when dealing with local refinement.

Unstructured Meshes.

Unstructured meshes are characterized by cells that don't follow a regular structure.

The connections between nodes don't follow a predefined pattern and may vary from cell to cell.

This makes unstructured meshes more **flexible** and suitable for **complex and irregular geometries**, such as those found in real-world problems.

Advantages of Unstructured Meshes:

- Adaptability to complex and irregular geometries;
- Ease of local refinement where greater detail is needed;
- Efficient for problems with variable or evolving geometries.

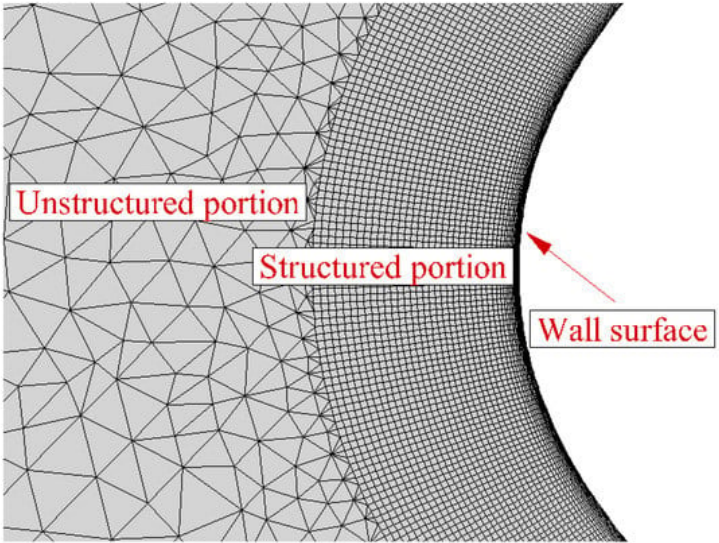
Disadvantages of Unstructured Meshes:

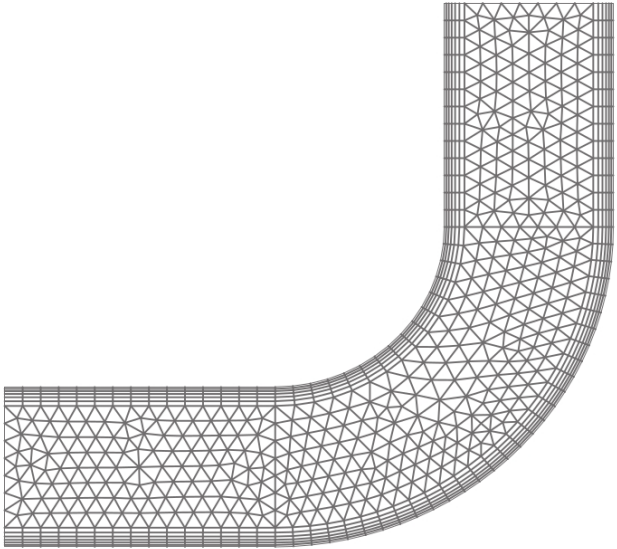
- More challenging to generate and require more complex algorithms;
- May lead to numerical instabilities if not handled properly;
- Generally more computationally intensive due to their unstructured nature.

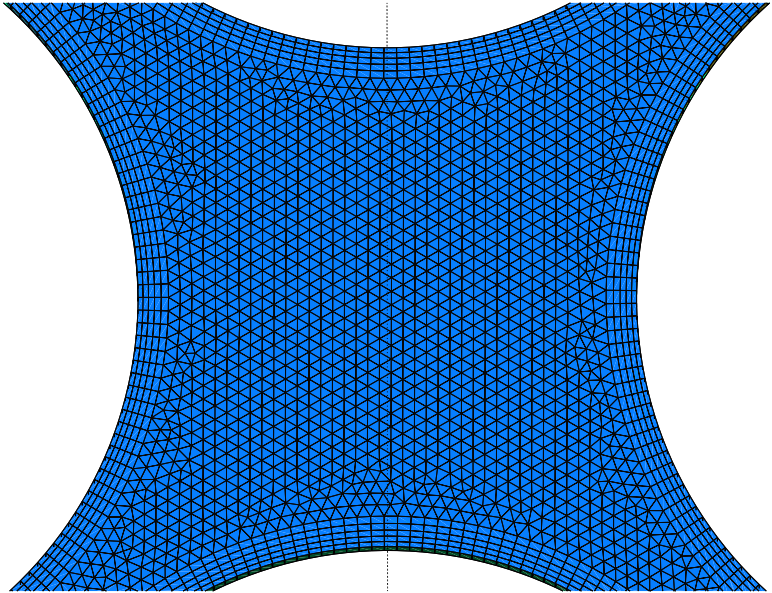
The choice between **structured** and **unstructured** meshes depends on the complexity of the problem geometry and the requirements for computational accuracy and efficiency.

Problems with simple and regular geometries may benefit from using structured meshes, while problems with complex and irregular geometries typically require unstructured meshes for accurate representation of the problem domain.

Hybrid meshes.





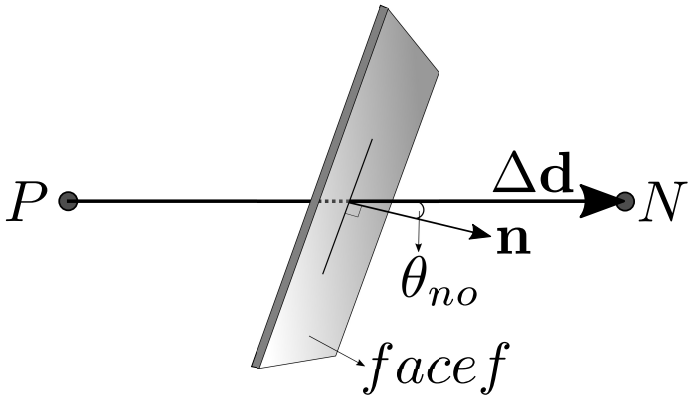

Y
Z
X

Mesh Quality Parameters

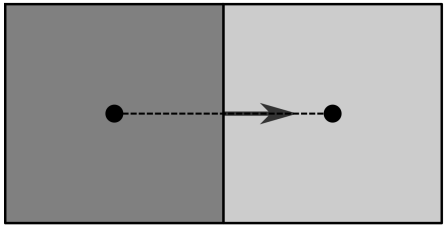
There is no single definition for a **good mesh**. It largely depends on the simulation type and geometry characteristics.

Generally, we have 3 parameters that quantify mesh quality: non-orthogonality (***non-orthogonality***), skewness (***skewness***), and aspect ratio (***aspect ratio***).

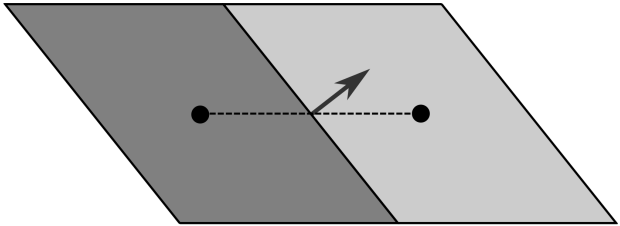
Non-orthogonality is defined as the angle between the vector connecting cell centers and the face normal.



Non-orthogonality $= 0$



Non-orthogonality $\neq 0$



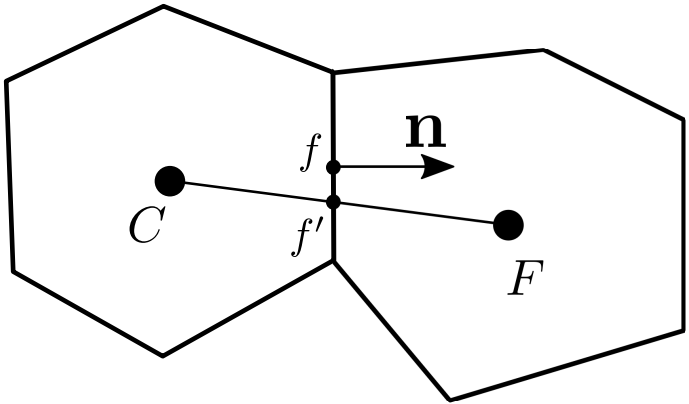
It's recommended to keep non-orthogonality as low as possible, especially near boundary surfaces.

OpenFOAM by default recommends **maximum non-orthogonality of** 70°.

For meshes with maximum non-orthogonality greater than 70°, the best approach is to remake the mesh.

Calculation corrections can be applied for non-orthogonal meshes. This is particularly important for diffusive terms (*laplacianSchemes*).

Skewness (***skewness***) occurs when there's a discrepancy between the face center location f and the point where the vector connecting cell centers intersects the face f'.



The skewness value is quantified as

$$\epsilon = \frac{|f - f'|}{|F - C|}$$

In an ideal mesh, the skewness value is zero.

OpenFOAM recommends keeping maximum skewness below 4.

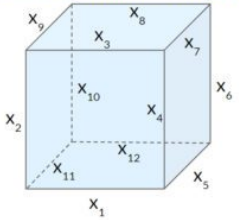
The aspect ratio (***aspect ratio***) measures the ratio between a cell's maximum and minimum width.

High aspect ratios lead to reduced solver efficiency and accuracy in OpenFOAM.

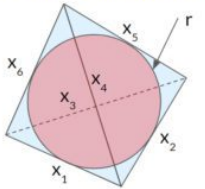
The aspect ratio should be kept as close to 1 as possible. Recommended to keep below 10.

OpenFOAM allows aspect ratios up to 1000.

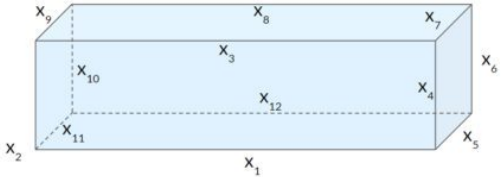
Hexahedral cell with aspect ratio of 1



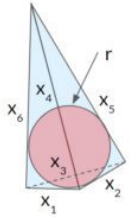
Tetrahedral cell with aspect ratio of 1



Hexahedral cell with aspect ratio of 4



Tetrahedral cell with aspect ratio of 4

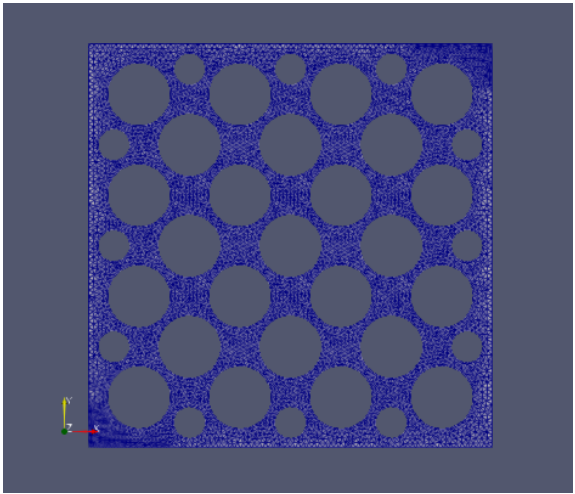


In the following slides, I present some meshes and the results when running the command:

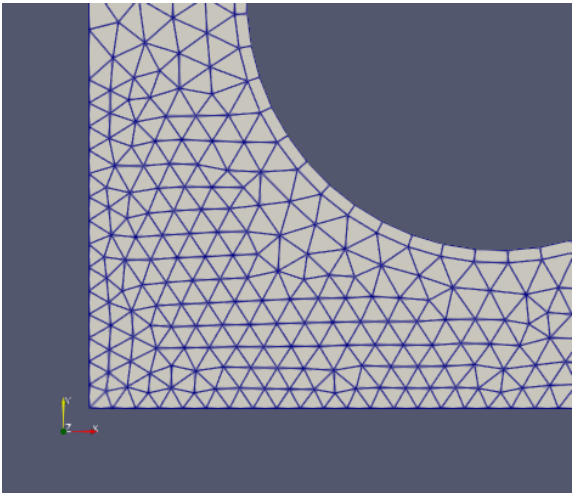
```
$ checkMesh
```

This command performs a mesh analysis and displays the main parameters on screen.

Example 0.



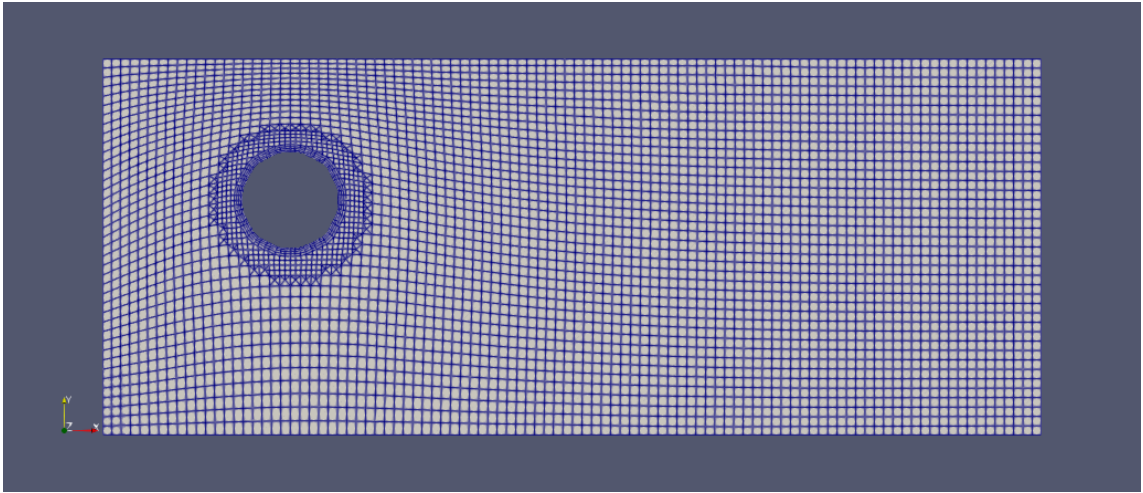
Example 0.



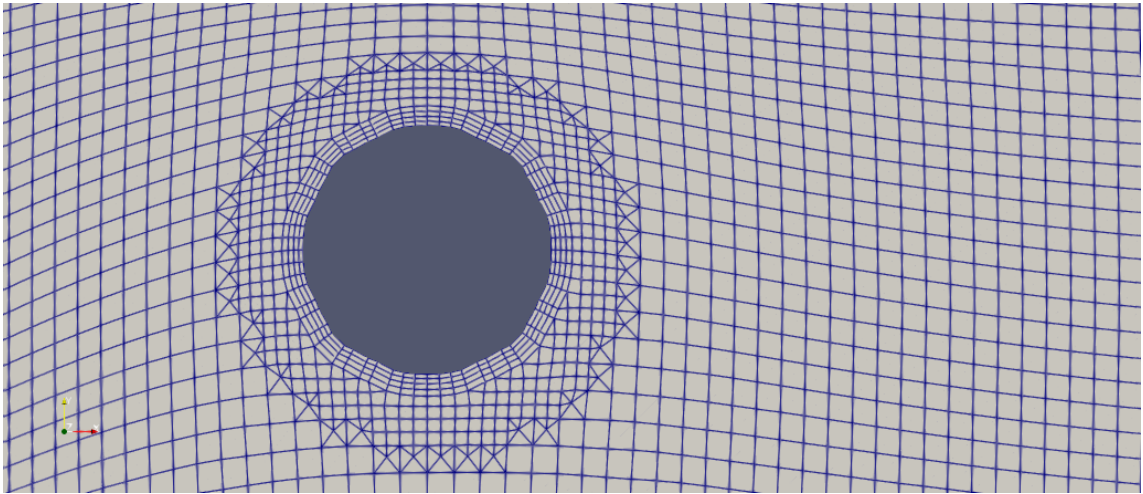
Example 0.

```
Checking geometry...
    Overall domain bounding box (0 0 0) (0.01 0.01 0.001)
    Mesh has 2 geometric (non-empty/wedge) directions (1 1 0)
    Mesh has 2 solution (non-empty) directions (1 1 0)
    All edges aligned with or perpendicular to non-empty directions.
    Boundary openness (-5.18388e-18 -3.59863e-18 -3.25883e-15) OK.
    Max cell openness = 2.28872e-16 OK.
    Max aspect ratio = 2.69672 OK.
    Minimum face area = 1.36255e-09. Maximum face area = 1.90612e-07.  Face area magnitudes OK.
    Min volume = 1.36255e-12. Max volume = 1.13124e-11.  Total volume = 4.81406e-08.  Cell volumes OK.
    Mesh non-orthogonality Max: 25.9529 average: 5.41437
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 0.44609 OK.
    Coupled point location match (average 0) OK.

Mesh OK.

End
```

Example 1.



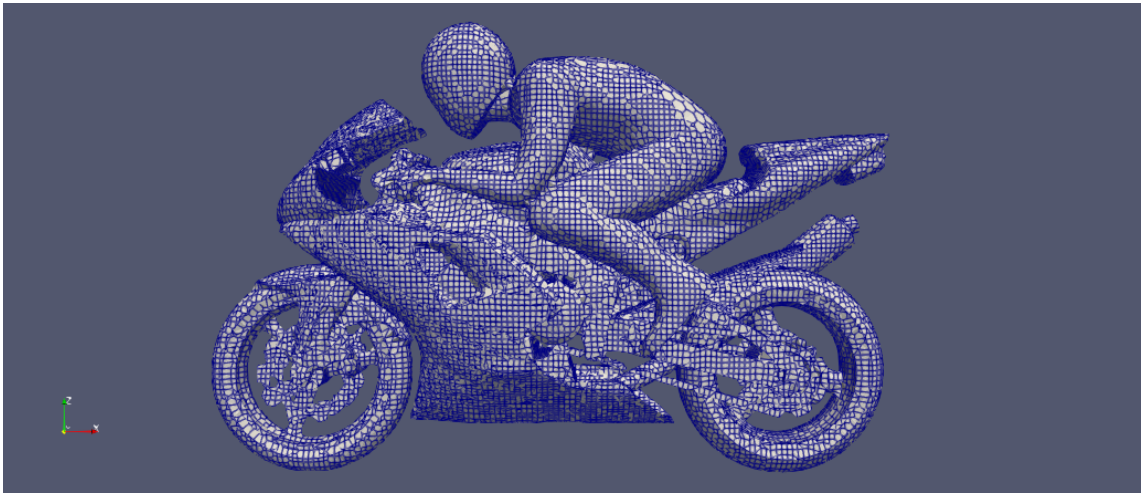
Example 1.



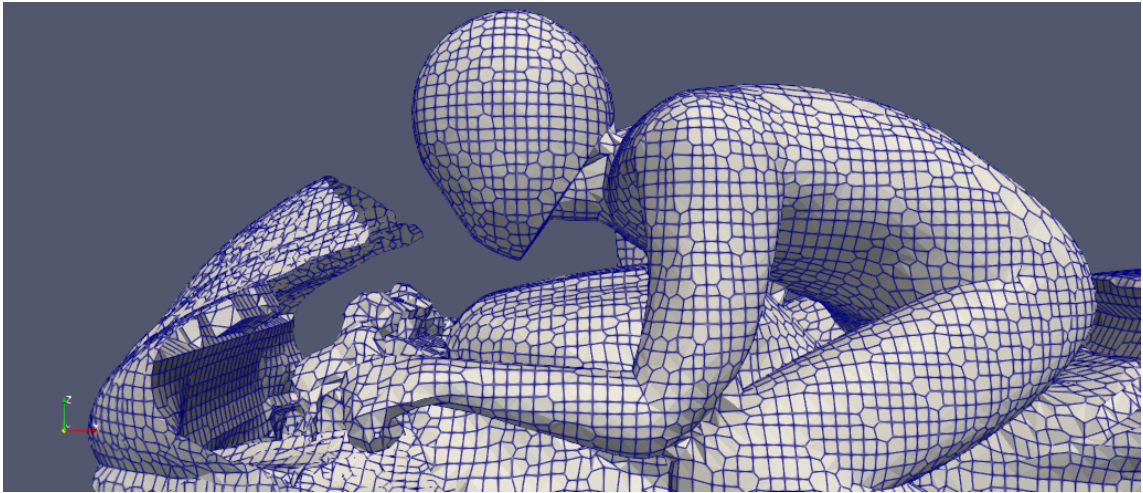
Example 1.

```
Checking geometry...
    Overall domain bounding box (0 0 1) (10 4 1.5)
    Mesh has 2 geometric (non-empty/wedge) directions (1 1 0)
    Mesh has 2 solution (non-empty) directions (1 1 0)
    All edges aligned with or perpendicular to non-empty directions.
    Boundary openness (7.08795e-18 -5.67036e-17 3.92887e-16) OK.
    Max cell openness = 2.0494e-16 OK.
    Max aspect ratio = 2.56368 OK.
    Minimum face area = 0.000752205. Maximum face area = 0.0761607.  Face area magnitudes OK.
    Min volume = 0.000376102. Max volume = 0.00762923.  Total volume = 19.5936.  Cell volumes OK.
    Mesh non-orthogonality Max: 37.6474 average: 7.32506
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 0.508419 OK.
    Coupled point location match (average 0) OK.

Mesh OK.

End
```

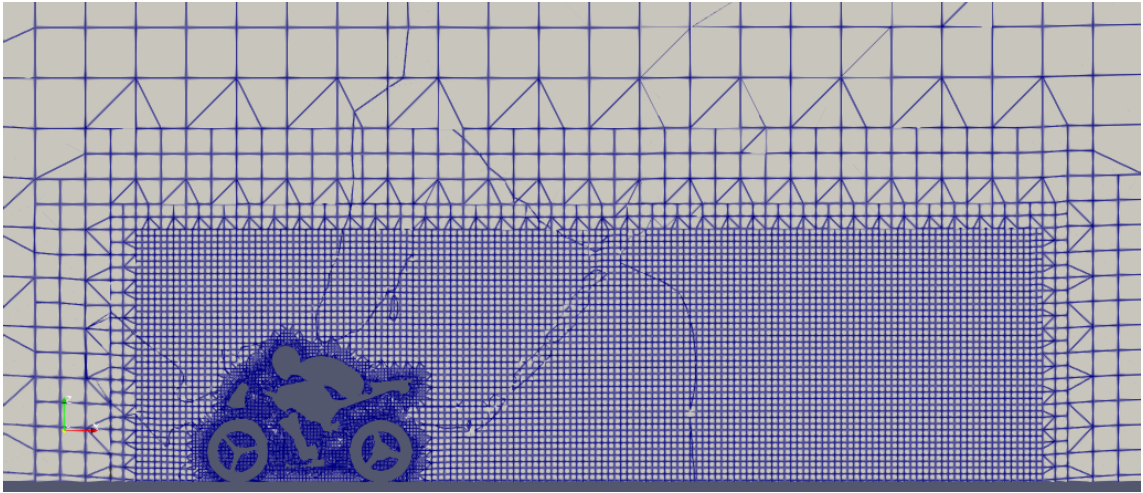
Example 2.



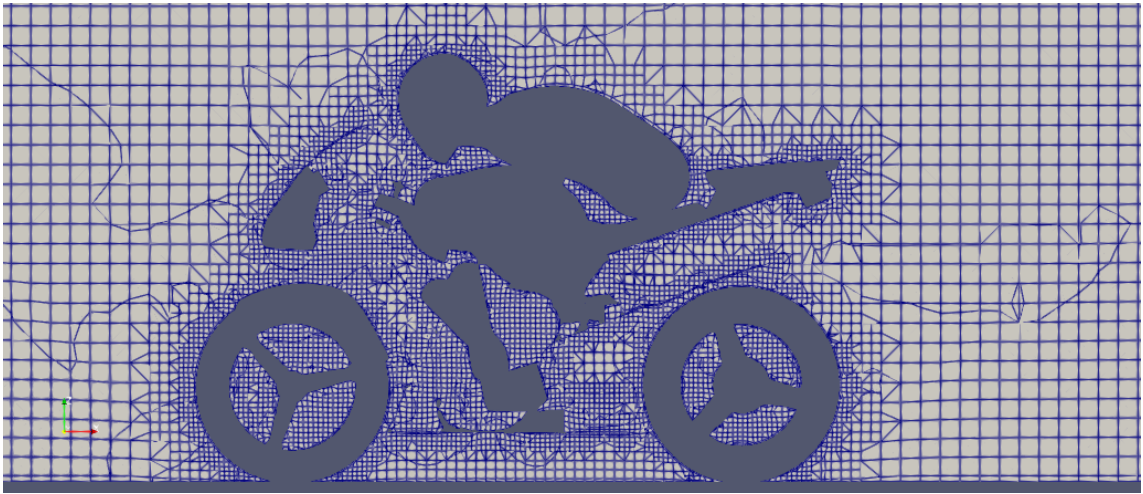
Example 2.



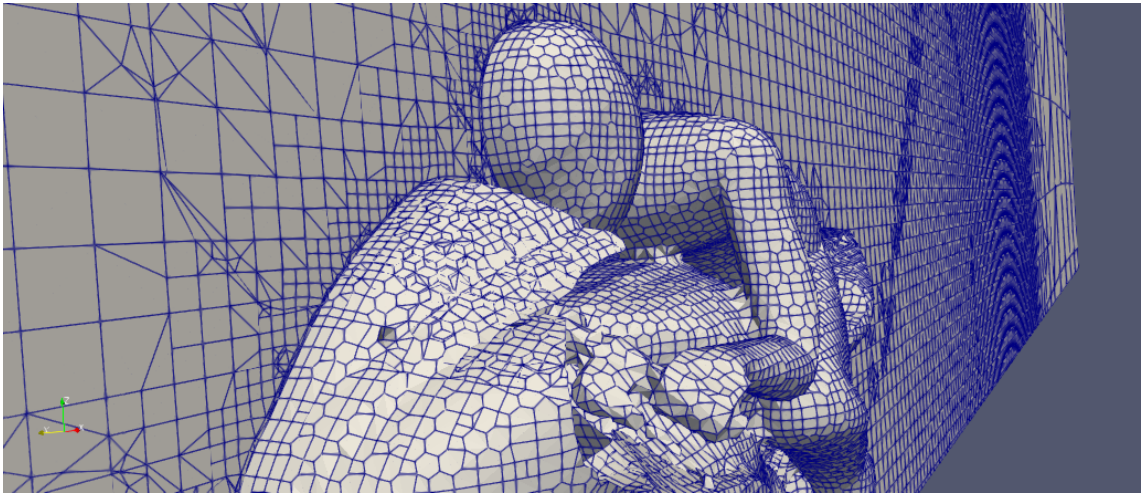
Example 2.



Example 2.



Examplo 2.

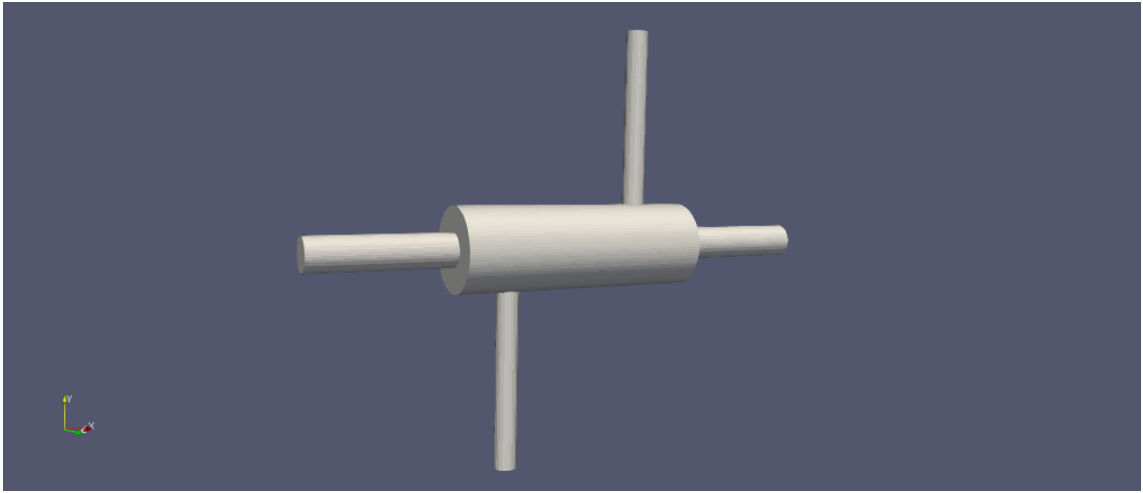


Example 2.

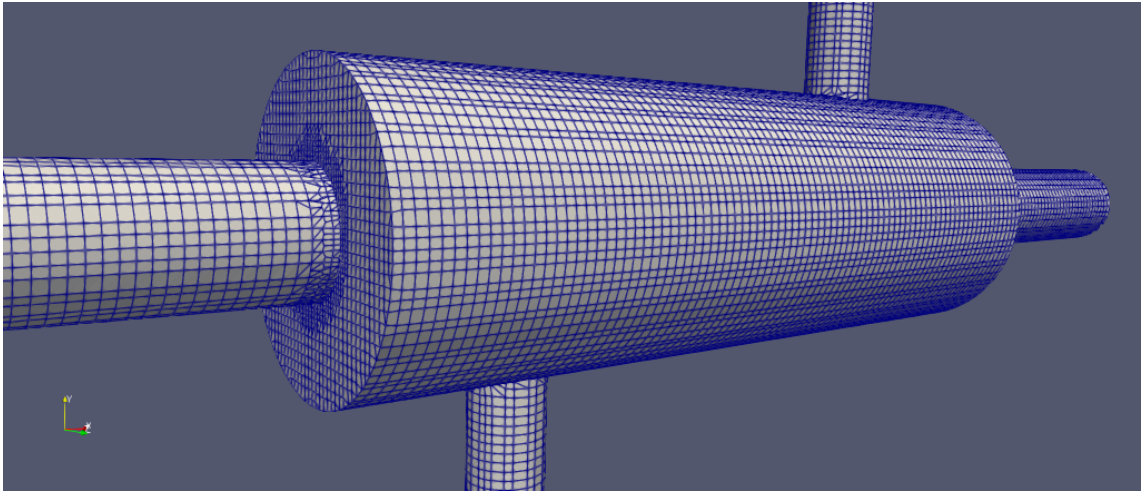
```
Checking geometry...
    Overall domain bounding box (-5 -4 0) (15 4 8)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (6.99091e-18 1.8099e-16 -2.15845e-16) OK.
    Max cell openness = 7.56608e-16 OK.
    Max aspect ratio = 41.0436 OK.
    Minimum face area = 8.4233e-07. Maximum face area = 1.01338.  Face area magnitudes OK.
    Min volume = 1.64487e-08. Max volume = 1.00461.  Total volume = 1279.67.  Cell volumes OK.
    Mesh non-orthogonality Max: 64.9965 average: 9.91917
    Non-orthogonality check OK.
    Face pyramids OK.
 ***Max skewness = 11.223, 14 highly skew faces detected which may impair the quality of the results
  <<Writing 14 skew faces to set skewFaces
    Coupled point location match (average 0) OK.

Failed 1 mesh checks.

End
```

Note the warning. You can run the simulation, but be aware that the results may be compromised due to mesh quality issues.

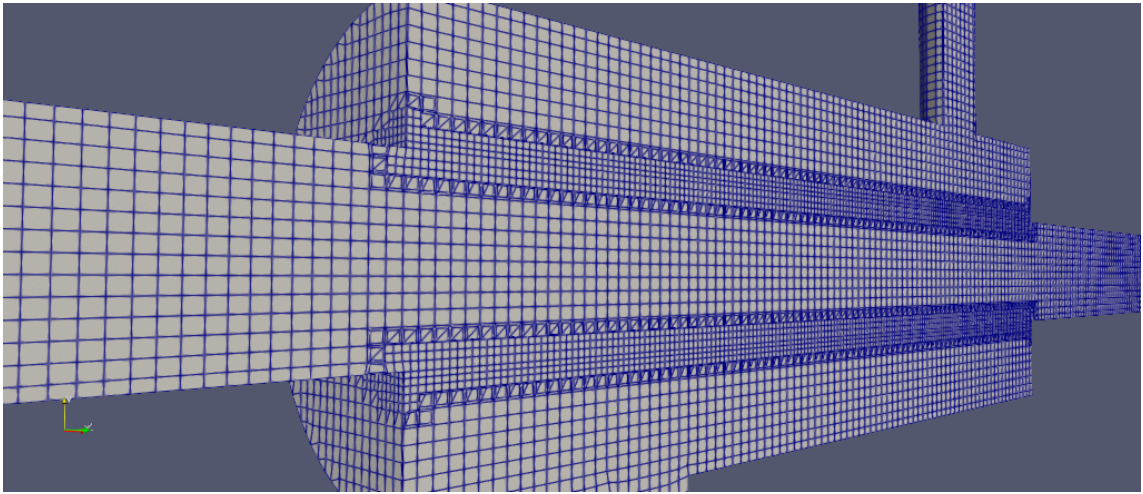
Example 3.



Example 3.



Example 3.



Example 3.

```
Checking geometry...
    Overall domain bounding box (0 -0.077500001 -0.015) (0.19000054 0.077500001 0.015)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (-2.6273236e-17 5.3265431e-17 -7.2852121e-16) OK.
    Max cell openness = 3.6676589e-16 OK.
    Max aspect ratio = 10.852132 OK.
    Minimum face area = 1.8361364e-08. Maximum face area = 1.6826547e-06.  Face area magnitudes OK.
    Min volume = 1.1537582e-11. Max volume = 1.4073213e-09.  Total volume = 8.0962096e-05.  Cell volume
s OK.
    Mesh non-orthogonality Max: 47.127806 average: 8.2143492
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 3.6355701 OK.
    Coupled point location match (average 0) OK.

Mesh OK.

End
```

Mesh in OpenFOAM

OpenFOAM is quite flexible regarding meshes. It can handle unstructured meshes composed of general convex polyhedra.

The mesh data used by OpenFOAM is stored in the *constant/polyMesh* directory (or in the time folder, when mesh changes during the simulation).

There are mainly 5 files that describe the mesh and boundary face conditions:

- *boundary*
- *faces*
- *neighbour*
- *owner*
- *points*

The *points* file lists all cell vertices.

The *faces* file lists all faces.

The *owner* file indicates which cell owns each face.

The *neighbour* file lists the neighboring cell for each face (in addition to the owner cell).

Finally, the *boundary* file contains information about each boundary type.

In some cases, **we need to configure the *boundary* file after mesh generation.**

This typically happens when using *gmsh* and the *gmshToFoam* command to generate the mesh.

In these cases, we must define the *type* for each boundary surface (the *patches*).

The main *types* are: *patch*, *wall*, *empty*, *symmetryPlane*, and *cyclic*.

Note: The *boundary* file is the only *polyMesh* file we'll occasionally need to edit. We won't modify the others.

Mesh Generation

There are many ways to generate mesh to use in an OpenFOAM simulation.

Commonly used mesh generation software in CFD:

- Ansys (Meshing, ICEM, CFX)
- COMSOL
- Pointwise
- SIMSCALE
- Salome
- cfMesh
- GAMBIT
- STAR-CCM+
- Gmsh
- OpenFOAM (blockMesh, snappyHexMesh)

We'll focus on 3 software tools: ***blockMesh***, ***Gmsh*** and ***snappyHexMesh***.

All three are *open source*. *blockMesh* and *snappyHexMesh* come with OpenFOAM.

blockMesh is suitable for very simple geometries.

To use *snappyHexMesh*, we need a background mesh (generated by *blockMesh*) and a file with the target geometry (which can be created in *Gmsh*).

Gmsh has a graphical interface and can generate both structured and unstructured meshes. *Gmsh* can be used to generaty the geometry and/or the mesh.

A good strategy to create the mesh is the following:

- For very simple geometries and meshes: ***blockMesh***.
- For moderately complex geometries: ***Gmsh***.
- For complex geometries: ***snappyHexMesh***.

For the **membrane module simulation,** we are going to use

- **blockMesh** to generate the background mesh,
- **Gmsh** to create the geometry and
- **snappyHexMesh** to create the final mesh using the geometry from Gmsh.

We've already covered some basics of *blockMesh*. I recommend consulting the *User Guide* and additional resources to further explore its capabilities.

Now we'll focus on the **fundamentals of *Gmsh*.**

snappyHexMesh will be addressed in our next lecture.

Sumário

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2009; **0**:1–24 *Prepared using* nmeauth.cls *[Version: 2002/09/18 v2.02]*

Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities

Christophe Geuzaine[1], Jean-François Remacle[2*]

[1] *Université de Liège, Department of Electrical Engineering And Computer Science, Montefiore Institute, Liège, Belgium.*

[2] *Université catholique de Louvain, Institute for Mechanical, Materials and Civil Engineering, Louvain-la-Neuve, Belgium*

SUMMARY

Gmsh is an open-source three-dimensional finite element grid generator with a build-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. This paper presents the overall philosophy, the main design choices and some of the original algorithms implemented in Gmsh. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS: Computer Aided Design, Mesh generation, Post-Processing, Finite Element Method, Open Source Software

Gmsh enables creation of both structured (*transfinite*) and unstructured 1D, 2D, and 3D meshes. Supported elements include triangles, quadrilaterals, tetrahedrons, hexahedrons, prisms, and pyramids.

Gmsh can be used in three ways:

- Through the graphical interface;
- Using its dedicated .geo scripting language;
- Via C++, C, Julia, and **Python** APIs.

The official Gmsh website is **https://gmsh.info/**. There you'll find installation instructions and comprehensive software documentation.

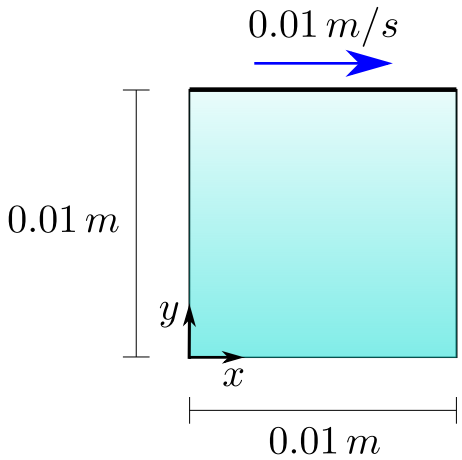
The source code is available at
https://gitlab.onelab.info/gmsh/gmsh.

An excellent way to learn Gmsh is by following the official tutorials at
https://gmsh.info/doc/texinfo/gmsh.html.

In the following slides, we'll examine simple examples. We'll explore geometry and mesh generation and how to use them in OpenFOAM.

Sumário

Let's solve again the **Lid Driven Cavity** problem, but now using Gmsh.



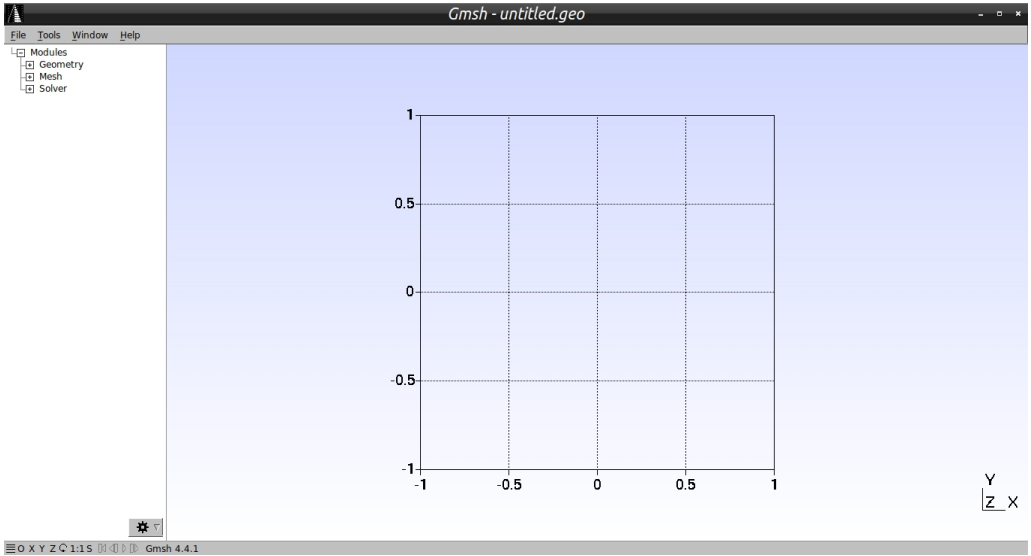
The simulation configuration is already defined. We only need to generate the new mesh. We used *blockMesh* in the first solution, now we will use *Gmsh*.

To open *Gmsh*, run the command:

```
$ gmsh
```

or launch the software directly.

You will see the image of the next slide.

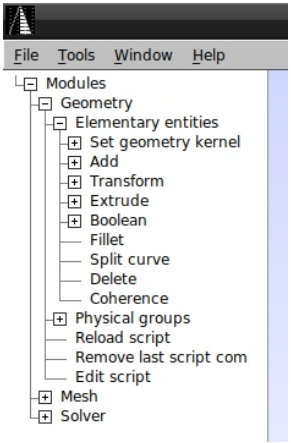

Gmsh - untitled.geo
File Tools Window Help
Modules
Geometry
Mesh
Solver
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
Y
Z X
O X Y Z 1:1 S
Gmsh 4.4.1

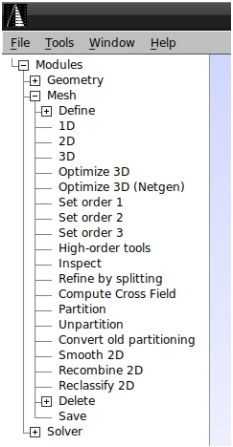
Go to *File -> New...* and create a file in .geo format. Name it as you prefer (e.g., cavity.geo).

Select the OpenCASCADE option.

This .geo file will save everything done in Gmsh through the graphical interface.

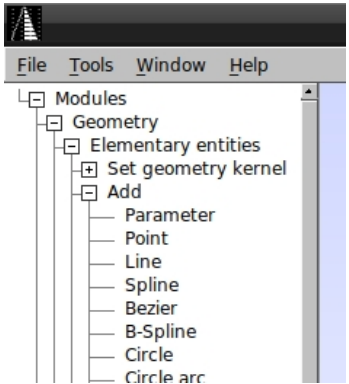
On the left side of the screen, we have the modules. We'll use the *Geometry* module to create the geometry and the *Mesh* module to generate the mesh.



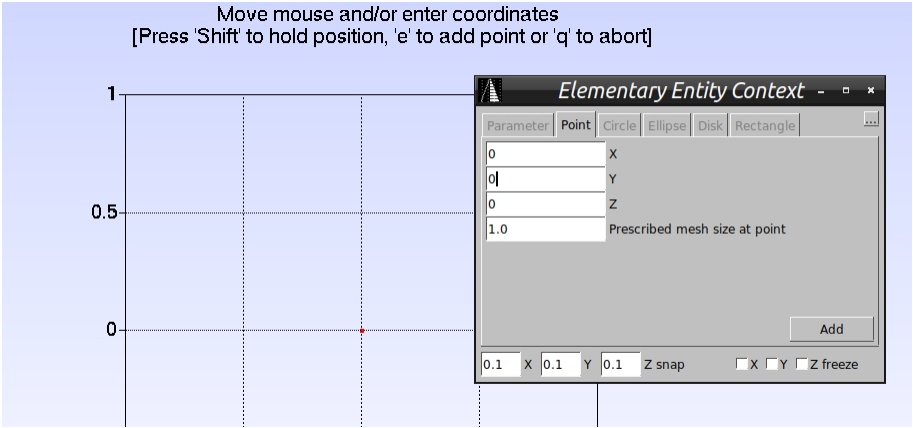


We'll create the square's 4 vertices first, then the 4 edges, and finally the surface.

To add a point, go to *Geometry / Elementary entities / Add / Point*.

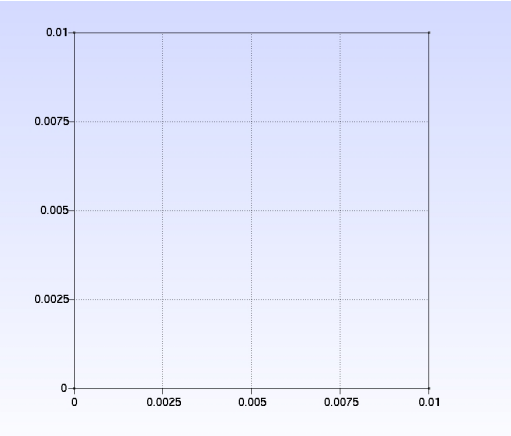


Enter the coordinates for each point and click *Add*.



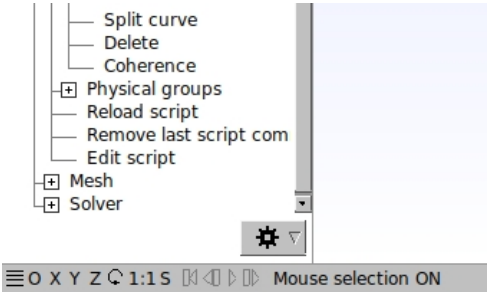
There are 4 points total: $(0, 0)$, $(0.01, 0)$, $(0.01, 0.01)$, and $(0, 0.01)$. Leave the *"Prescribed mesh size at point"* option as 1.0.

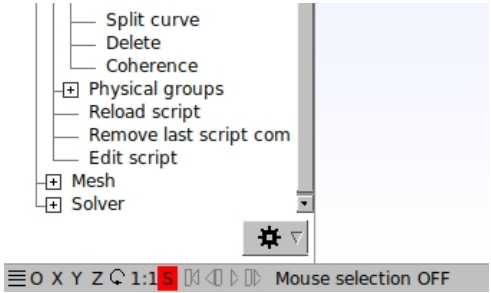
After adding all points, close the coordinates window and press 'q'. We now have all 4 vertices.



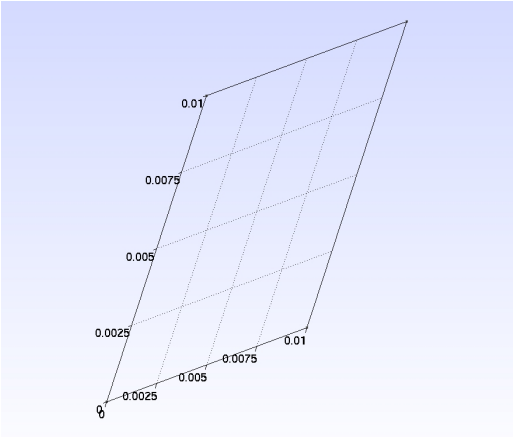
Note: Type the letter ‘q’ to finish a command. When you use ‘Esc’, Gmsh toggles mouse selection on/off.

Observe the bottom left corner of the screen. If it is red, you won’t be able to select anything with the mouse. Just type ‘Esc’ again.





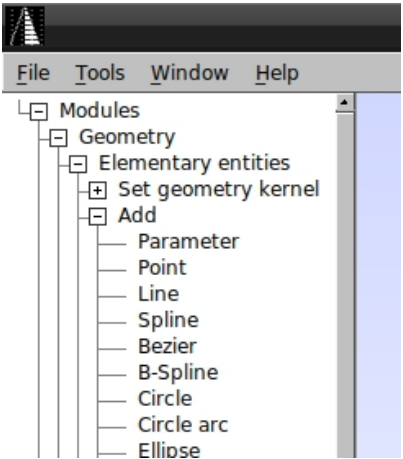
Use mouse buttons to rotate and zoom the drawing.



To return to the initial view, double-click with the left mouse button and select ***Reset viewport***.

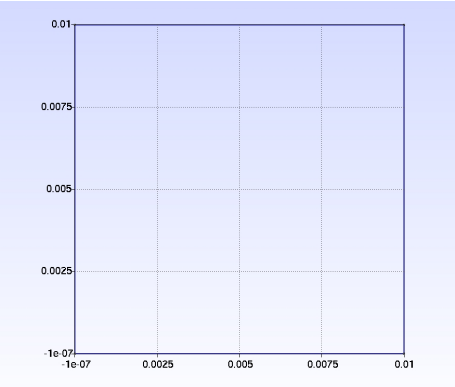
Continuing, let's create the edges (or lines).

Click on *Geometry / Elementary entities / Add / Line*.



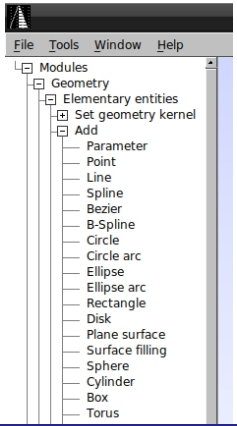
Select the start and end points for each *line*.

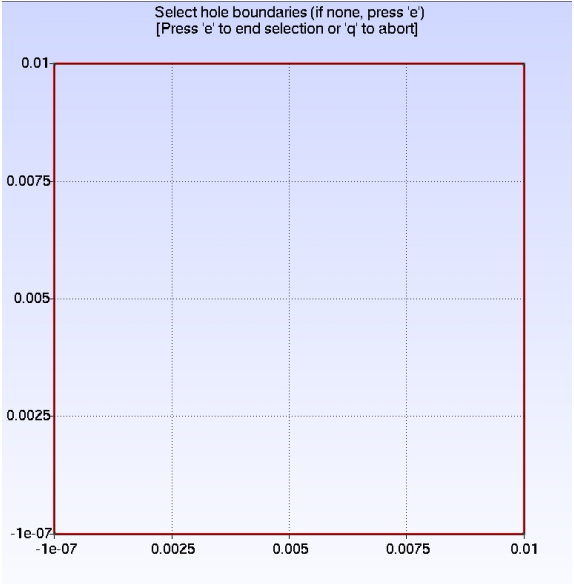
Type ‘q’ when finished.



Now we need to specify that these 4 edges form a plane surface.

For this, click on *geometry/elementary entities/Add/Plane surface*, select the square (more specifically, the dashed line at the center of the square) and type the letter 'e' on the keyboard.




Select hole boundaries (if none, press 'e')
[Press 'e' to end selection or 'q' to abort]
0.01
0.0075
0.005
0.0025
-1e-07
-1e-07
0.0025
0.005
0.0075
0.01

Hover your mouse over the vertices, edges, and center of the square. You'll see information about these geometric entities.

One **very interesting** detail about Gmsh: everything we're doing is recorded in the .geo file.

This file is saved in the directory where Gmsh was opened. It contains the code to generate this geometry.

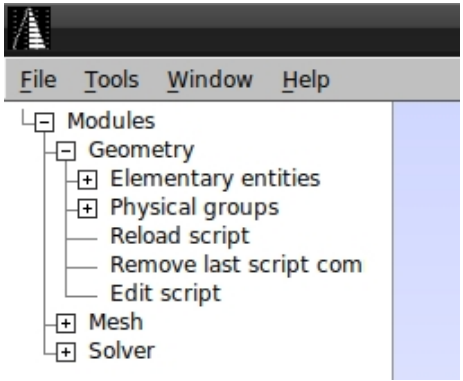
You can modify the code through text lines and reload the new version into the software.

This is my .geo file:

```
SetFactory("OpenCASCADE");
//+
Point(1) = {0, 0, 0, 1.0};
//+
Point(2) = {0.01, 0, 0, 1.0};
//+
Point(3) = {0.01, 0.01, 0, 1.0};
//+
Point(4) = {0.0, 0.01, 0, 1.0};
//+
Line(1) = {1, 2};
//+
Line(2) = {2, 3};
//+
Line(3) = {1, 4};
//+
Line(4) = {4, 3};
//+
Curve Loop(1) = {3, 4, -2, -1};
//+
Plane Surface(1) = {1};
```

You can modify point positions, create variables, and include sections of other geometries in the code.

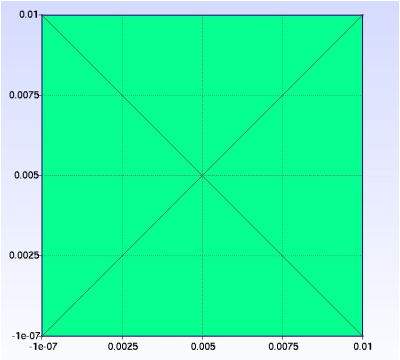
To open the code file, go to *Geometry / Edit script*. After making changes, click *Geometry / Reload script* to update the drawing.



Continuing, with the plane surface we can now generate the mesh.

On the left side of the screen, click *Mesh* / *2D*.

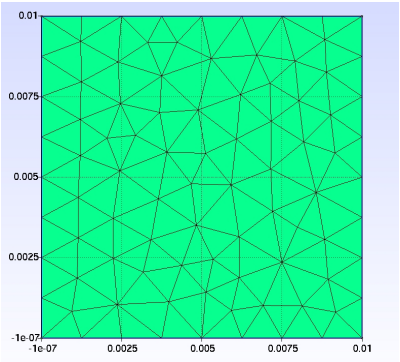
You should see something like this:



(If you don't see the mesh, double-click with the left mouse button and adjust the *Mesh visibility* settings, case 2D.)

This mesh only has 4 cells (this was defined when creating points in the *Prescribed mesh size at point* option). To refine the mesh further, click *Mesh / Refine by spliting*.

I clicked 3 times on *Refine by spliting* and then *2D* again. This was the result:



You can click *Smooth 2D* to make the mesh generation algorithm try to improve cell quality.

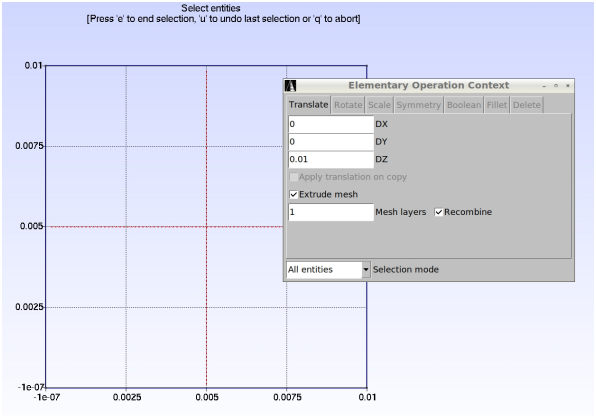
You can also click *Recombine* to convert to quadrilaterals.

Done! These are the steps to generate a 2D mesh in Gmsh: create a surface and then run the mesh command.

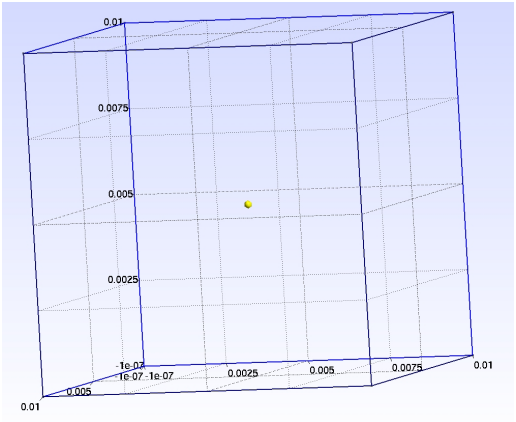
The problem is that OpenFOAM is 3D. Even for 2D simulations in OpenFOAM, we need a 3D mesh.

Therefore, we'll need to create a 3D geometry by extruding this square in the z direction.

To do this, click *Geometry / Extrude / Translate*. Select the square, configure the extrusion as shown below (0.01 meter extrusion in the z-direction, with mesh extrusion using just one layer), click anywhere on the drawing, type ‘e’ and then ‘q’.



Rotate the image.



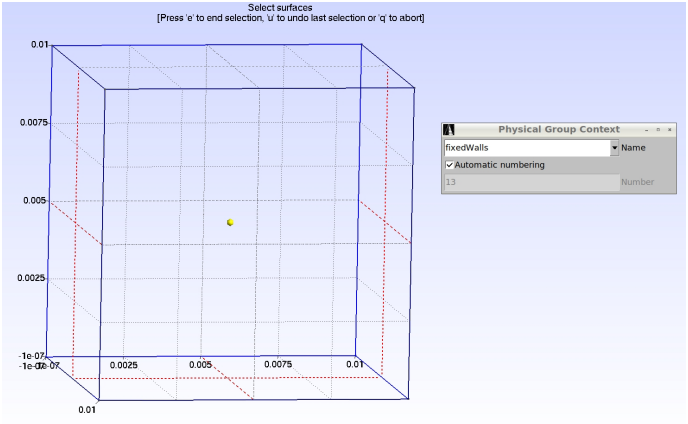
We now have a 3D geometry.

Before generating the mesh, let's define surface and volume names. This will be crucial for OpenFOAM, to specify boundary conditions.

We'll use the *Physical groups* option. Note that these names we're defining don't affect the geometry or mesh. They only help the CFD software to recognize each surface.

Click *Geometry / Physical groups / Add / Surface*.

Select the left, right and bottom surfaces, name them *fixedWalls*, click anywhere on the drawing and type ‘e’.

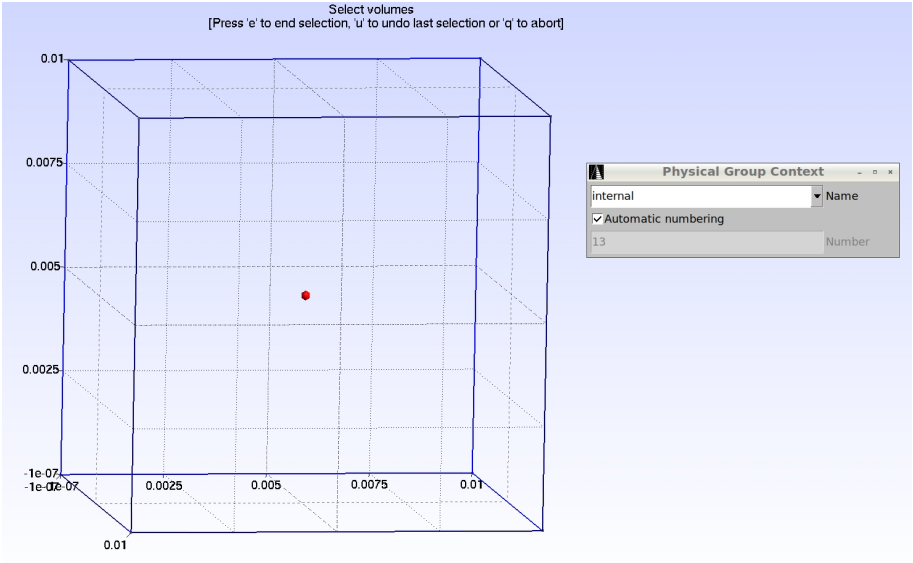


Repeat the procedure for the top wall (*movingWall*) and the front/back walls (*frontAndBack*).

Hover your mouse over the dashed line at the center of each surface. If everything is correct, you'll see the name of the *Physical group* you just defined.

Finally, click *Geometry / Physical groups / Add / Volume*, select the yellow sphere (it will turn red), name it *internal*, click anywhere on the drawing and type 'e'.

OpenFOAM will need to recognize this internal volume of cells.


Select volumes
[Press 'e' to end selection, 'u' to undo last selection or 'q' to abort]
0.01
0.0075
0.005
0.0025
-1e-07
-1e-07
0.0025
0.005
0.0075
0.01
0.01
Physical Group Context
internal
Name
Automatic numbering
13
Number

Follow all these modifications in the text file. See how it has been modified.

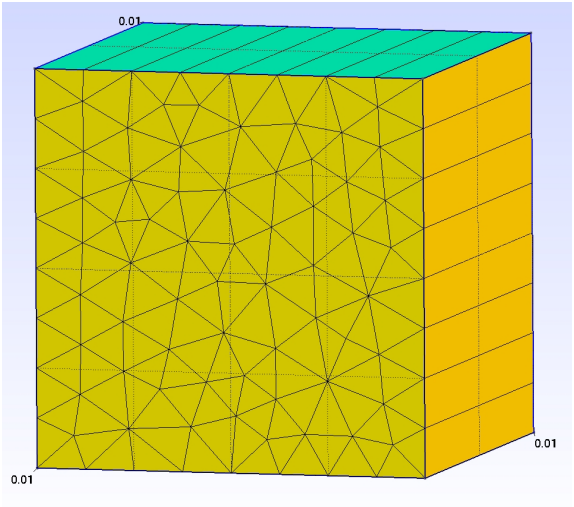
Now we can generate the mesh to export it to OpenFOAM.

Click *Mesh/2D*. Click 3 times on *Mesh/Refine by splitting* and then *Mesh/2D* again.

Finally, click *3D*, since the final mesh must be 3D.

Check if the message *'Done optimizing 3D mesh'* appears in the bottom left corner of the screen.

Done! We now have a 3D mesh ready for OpenFOAM.



Note that our mesh has only cell in the z direction.

To export the mesh, go to *File -> Export*, select the *Mesh* format (*.msh*), click *Save* and choose *Version 2 ASCII* as the format option.

Open the generated file and see how it was configured.

We're finished with Gmsh!

In OpenFOAM...

Copy the .msh file to the main case directory. Here, use the 2D shear cavity case we saw in Lecture 00 (the mesh was originally generated with *blockMesh*).

Run the command (this command converts the *.msh* mesh from *Gmsh* to an *OpenFOAM* style mesh (it creates the *constant/polymesh* folder)):

```
$ gmshToFoam <filename>
```

The mesh created can be found in *constant/polymesh* folder.

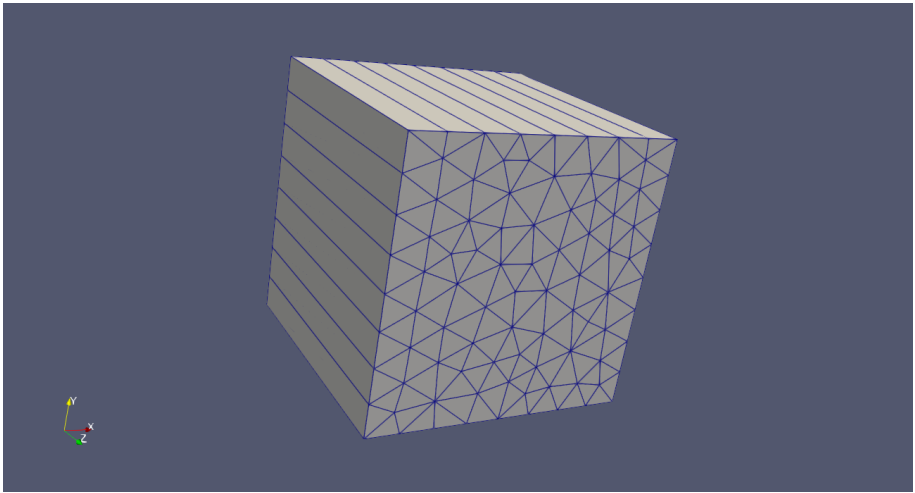
The command *gmshToFoam*, by default, gives the type *patch* for every boundary. We need to correct it.

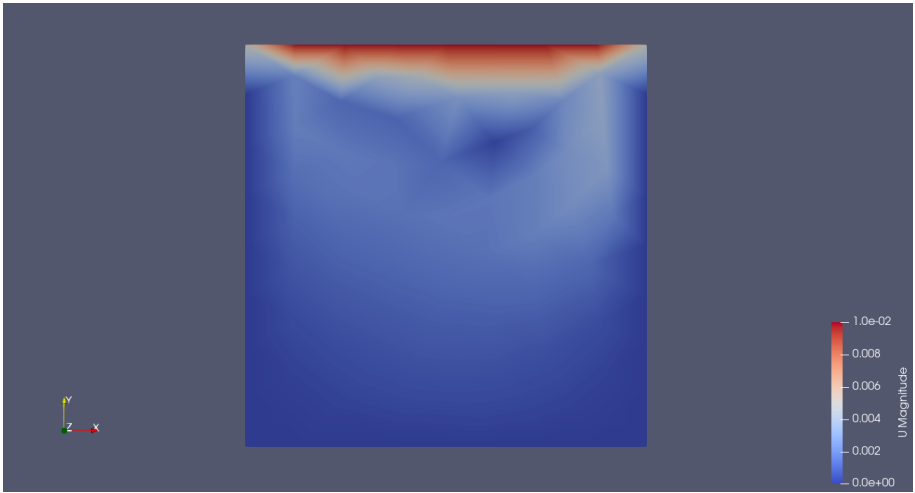
The *frontAndBack* boundary must has type 'empty'. Change it in *constant/polyMesh/boundary*.

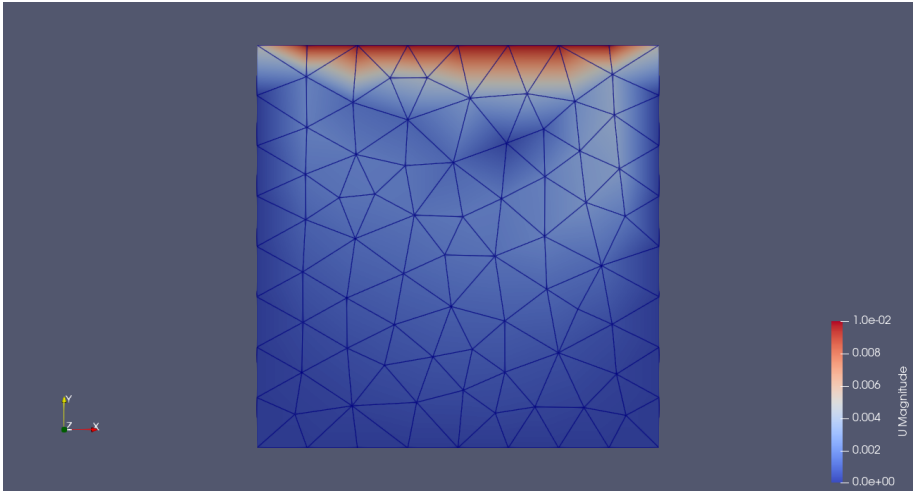
The *fixedWalls* and *movingWall* boundaries must have type 'empty'. Change it in *constant/polyMesh/boundary*.

Check the boundary conditions in the 0 directory and run the case.

(Compare the results of this mesh with the original. Which one is better?)


Y
X
Z


Y
Z
X
1.0e-02
0.008
0.006
0.004
0.002
0.0e+00
U Magnitude


Y
Z
X
1.0e-02
0.008
0.006
0.004
0.002
0.0e+00
U Magnitude

Sumário

Now we are going to generate a structure mesh for the cavity.

You should use structured mesh when possible. In genereal, the results are better in OpenFOAM for structured and hexahedral meshes.

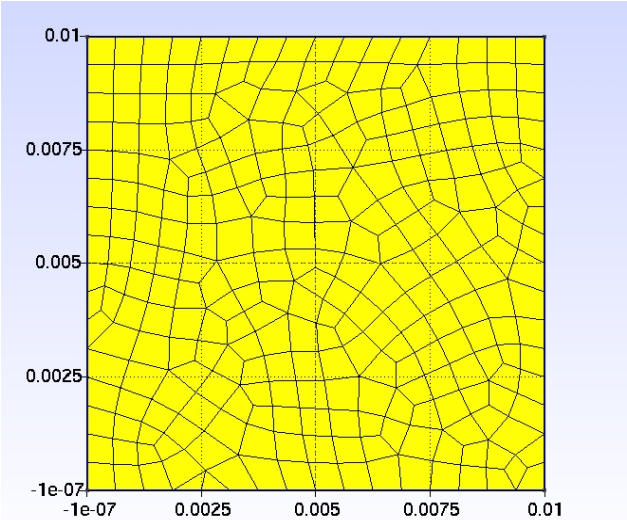
To generate a square cavity with structured mesh, we'll use the file we already created.

First, we'll specify we want quadrilaterals instead of triangles using the *Recombine* command.

In the left-side tree, click *Mesh/Recombine*. Select the original back surface (*Plane 1*). Type 'e' then 'q'.

It's important to select *Plane 1* because it's the original surface (others were created by extrusion).

Click *Mesh/2D* then *Refine by splitting* (three times). Then click in *Mesh/2D* again. Note the elements are now quadrilaterals.


0.01
0.0075
0.005
0.0025
-1e-07
-1e-07
0.0025
0.005
0.0075
0.01

The mesh consists of quadrilaterals but is still unstructured.

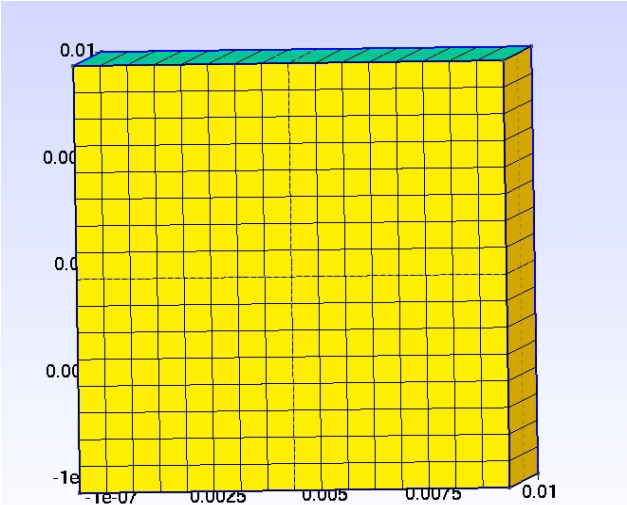
The command to make it structured is ***Transfinite***.

Click *Mesh/Define/Transfinite/Surface* and select surface *Plane 1*. Click anywhere and type 'e' then 'q'.

Click *Mesh/2D*, *Mesh/Refine by splitting* (three times), and *Mesh/2D* again.

The result is shown on the next slide.

(To clean the mesh and restart its generation, you just need to click in *Mesh/1D*.)

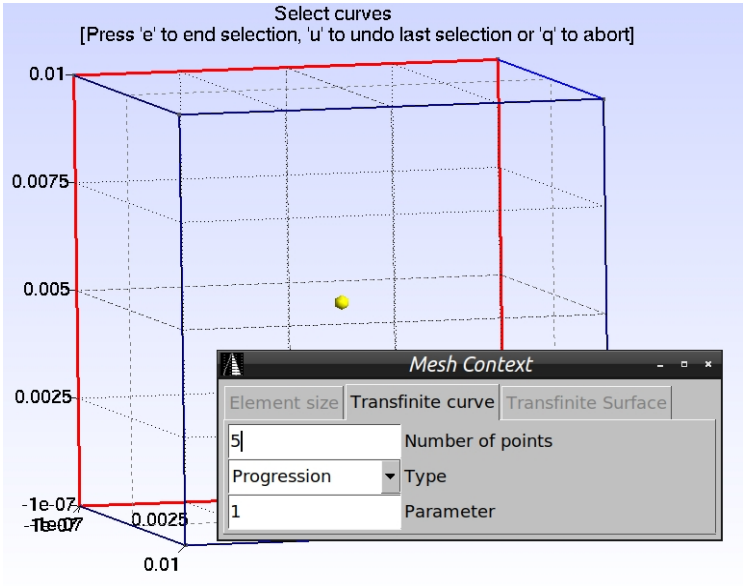

0.01
0.00
0.0
0.00
-1e
-1e-07
0.0025
0.005
0.0075
0.01

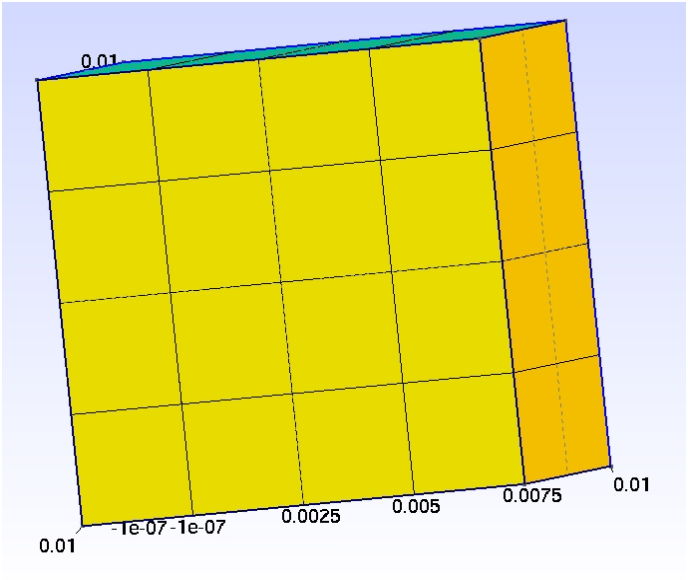
We could further improve by defining the exact number of points per edge. Clicking *Refine by splitting* multiple times is tedious and imprecise.

For this, click *Mesh/Define/Transfinite/Curve*, select *Lines* 1, 2, 3 and 4, and set 5 points per line.

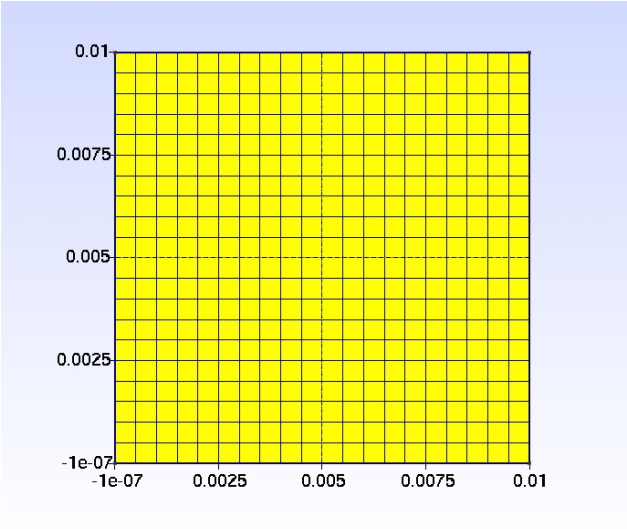
5 points correspond to 4 cells per line.

Generate the mesh with *Mesh/2D*. The click in *Mesh/3D* and export in *msh* format.


Select curves
[Press 'e' to end selection, 'u' to undo last selection or 'q' to abort]
0.01
0.0075
0.005
0.0025
-1e-07
0.0025
0.01
Mesh Context
Element size
Transfinite curve
Transfinite Surface
5
Number of points
Progression
Type
1
Parameter


0.01
0.01
-1e-07
-1e-07
0.0025
0.005
0.0075
0.01

Edit the text file to change from 5 to 21 points. In Gmsh, click *Geometry/-Reload script* then click *Mesh/2D* and ***Mesh/3D***.



The complete code:

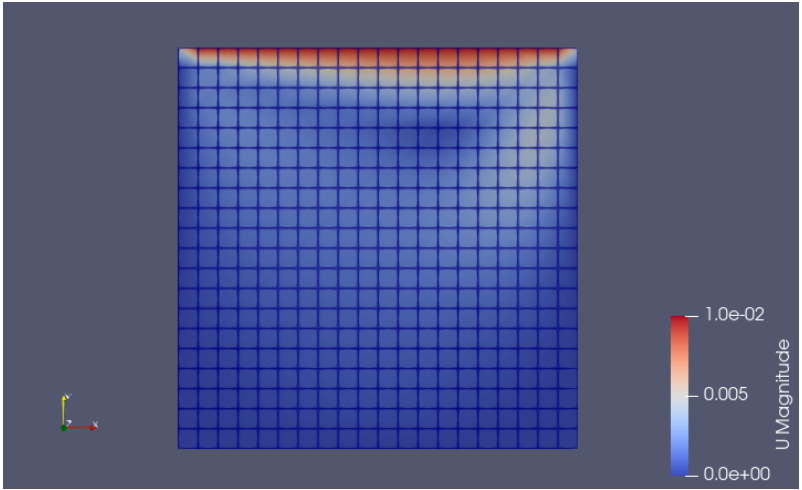
```
SetFactory("OpenCASCADE");
//+
Point(1) = {0, 0, 0, 1.0};
//+
Point(2) = {0.01, 0, 0, 1.0};
//+
Point(3) = {0.01, 0.01, 0, 1.0};
//+
Point(4) = {0.0, 0.01, 0, 1.0};
//+
Line(1) = {1, 2};
//+
Line(2) = {2, 3};
//+
Line(3) = {1, 4};
//+
Line(4) = {4, 3};
```

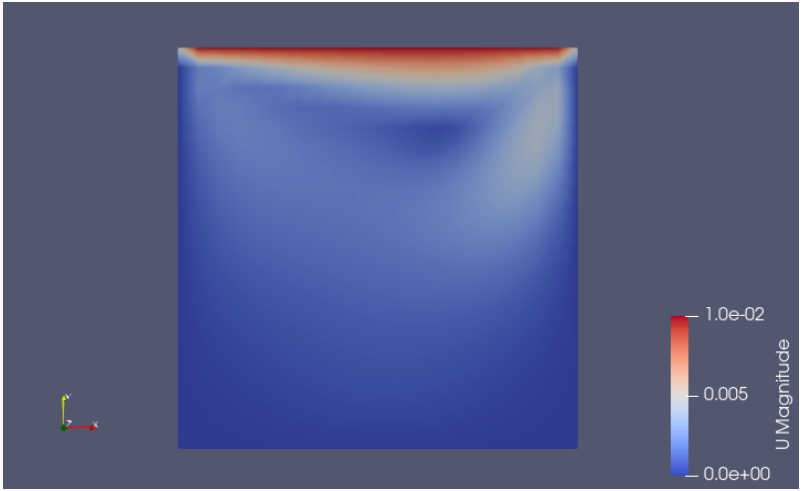
```
//+
Curve Loop(1) = {3, 4, -2, -1};
//+
Plane Surface(1) = {1};
//+
Extrude {0, 0, 0.01} {
  Surface{1}; Layers{1}; Recombine;
}
//+
Physical Surface("fixedWalls") = {2, 4, 5};
//+
Physical Surface("movingWall") = {3};
//+
Physical Surface("frontAndBack") = {6, 1};
//+
Physical Volume("internal") = {1};
//+
Recombine Surface {1};
//+
Transfinite Surface {1};
//+
Transfinite Curve {3, 2, 1, 4} = 21 Using Progression 1;
```

Next you can export the *.msh* file and generate the OpenFOAM mesh with the command *gmshToFoam*.

Change the types in the *constant/polymesh/boundary* file.

And now you can run the simulation: *foamRun* .


1.0e-02
0.005
0.0e+00
U Magnitude



Is this result better than the one for unstructured mesh? What do you think?

Important Notes.

For a ***Transfinite* surface**, it must be a quadrilateral and opposite sides must have the same number of points.

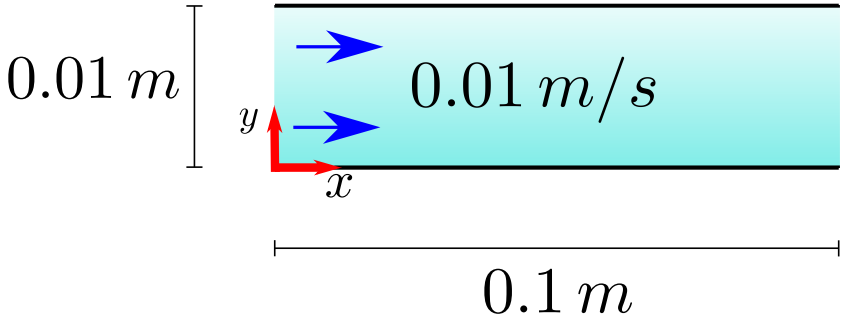
It's better to use *Transfinite/Curve* for one *Line* at a time. This makes it easier to control progression (try setting progression to 1.1 and see what happens). Then individualize the *Transfinite* and try again.

The final mesh (*.msh* file) must **always be 3D**. Click in *Mesh/3D*.

To generate a 3D mesh with multiple divisions along the z-direction, simply change the *Layers* value during extrusion (look for the extrusion command in the geo file and change the value from 1 to 10; then *Reload* in Gmsh and regenerate the mesh).

Sumário

Let's study again the flow between parallel plates. But now, we are going to use *Gmsh* to generate the mesh.



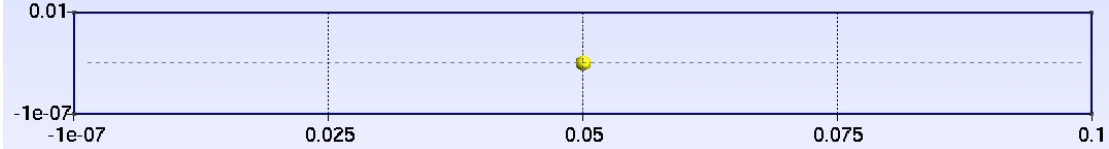
Steps:

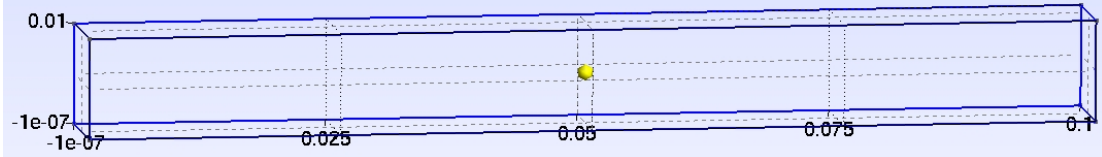
- Create the geometry using *Gmsh*
- Create the mesh using *Gmsh*
- Export the mesh (*.msh* file)
- Convert the mesh to *OpenFOAM* style (*gmshToFoam* command)
- Edit boundary types in *constant/polyMesh/boundary*
- Check the simulation configuration (*0*, *system* and *constant* folders)
- Run the simulation (*foamRun* command)

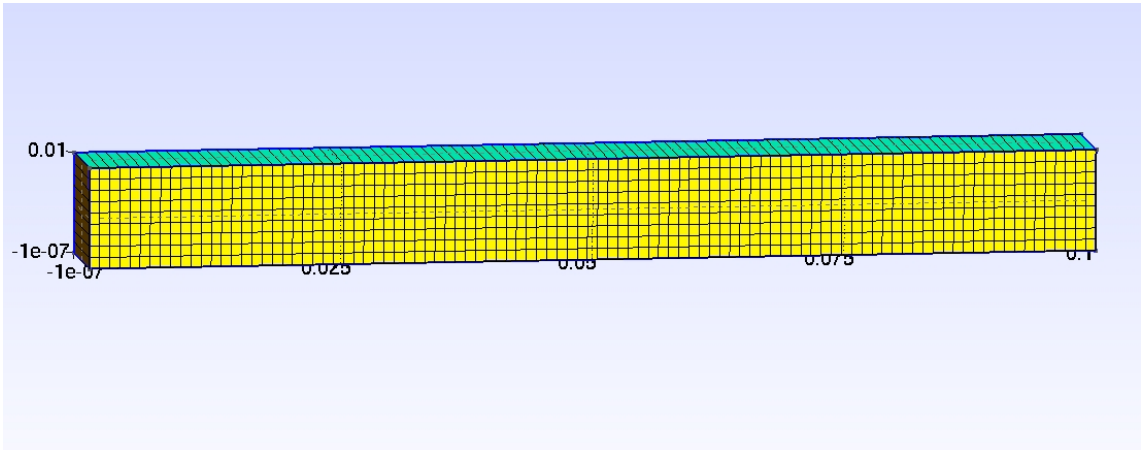
The simulation configuration is already correct (we are going to use the previous parallel plate simulation). We need to generate the mesh (now using *Gmsh*).

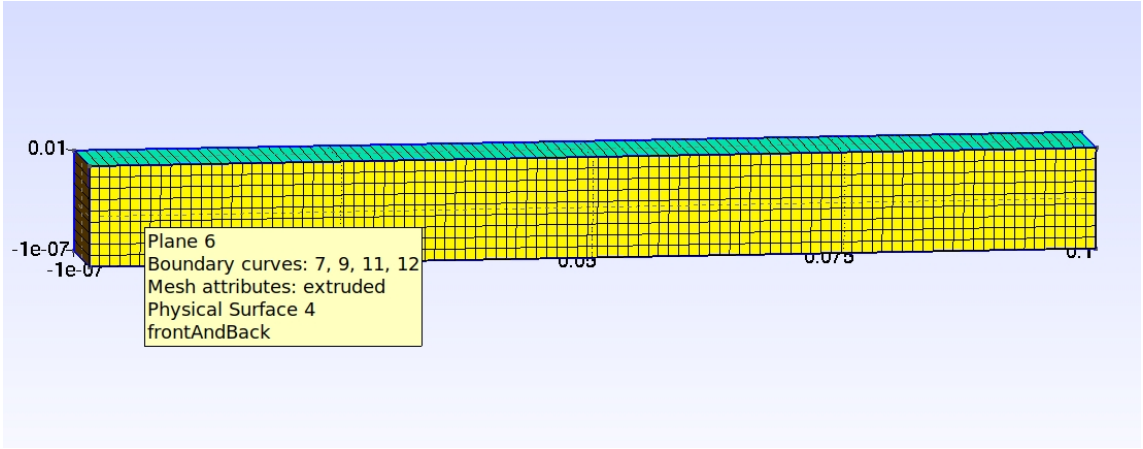
In the next slides I show some *Gmsh* images of the mesh creation and the final code.

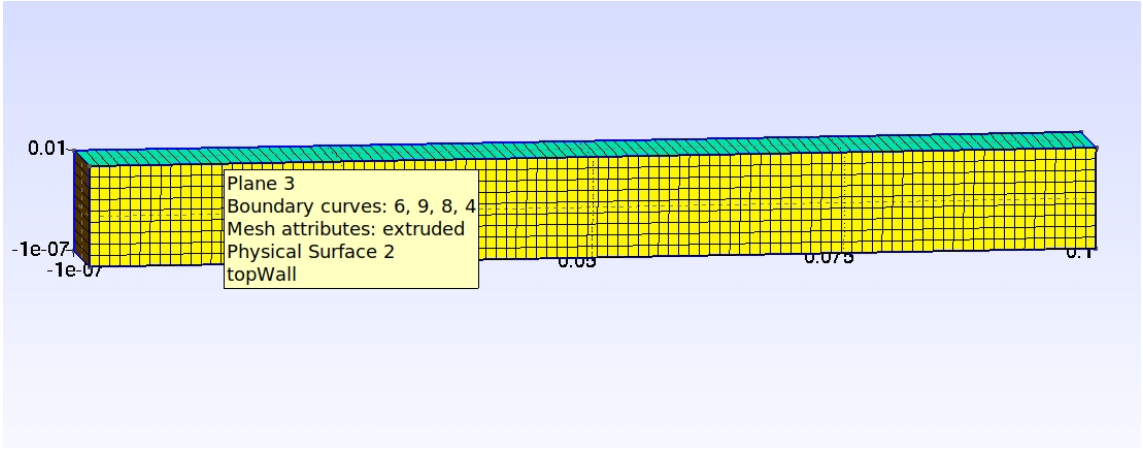
This simulation case can be found in the 'simulations' folder.

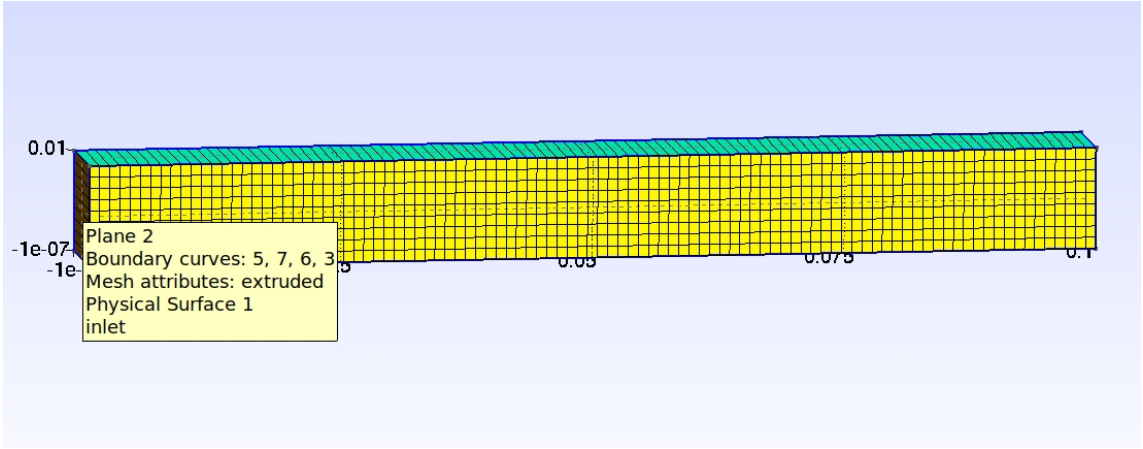

0.01
-1e-07
-1e-07
0.025
0.05
0.075
0.1


0.01
-1e-07
-1e-07
0.025
0.05
0.075
0.1


0.01
-1e-07
-1e-07
0.025
0.05
0.075
0.1


0.01
-1e-07
-1e-07
0.05
0.075
0.1
Plane 6
Boundary curves: 7, 9, 11, 12
Mesh attributes: extruded
Physical Surface 4
frontAndBack


0.01
-1e-07
-1e-07
Plane 3
Boundary curves: 6, 9, 8, 4
Mesh attributes: extruded
Physical Surface 2
topWall
0.05
0.075
0.1

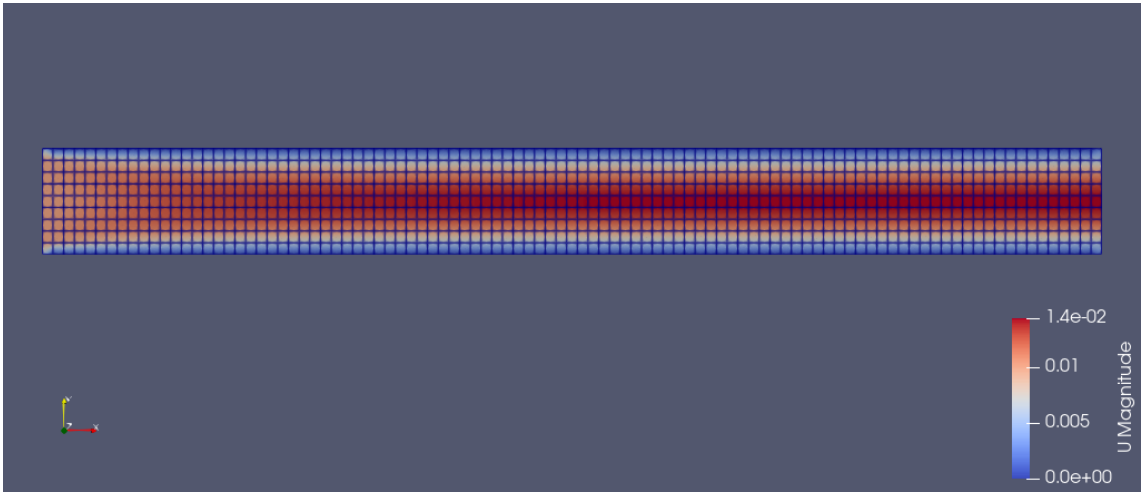

0.01
-1e-07
-1e
Plane 2
Boundary curves: 5, 7, 6, 3
Mesh attributes: extruded
Physical Surface 1
inlet
0.05
0.075
0.1

The complete *Gmsh* code (*parallelPlate.geo*):

```
SetFactory("OpenCASCADE");
//+
Point(1) = {0, 0, 0, 1.0};
//+
Point(2) = {0.1, 0, 0, 1.0};
//+
Point(3) = {0.1, 0.01, 0, 1.0};
//+
Point(4) = {0, 0.01, 0, 1.0};
//+
Line(1) = {1, 2};
//+
Line(2) = {2, 3};
//+
Line(3) = {1, 4};
//+
Line(4) = {4, 3};
```

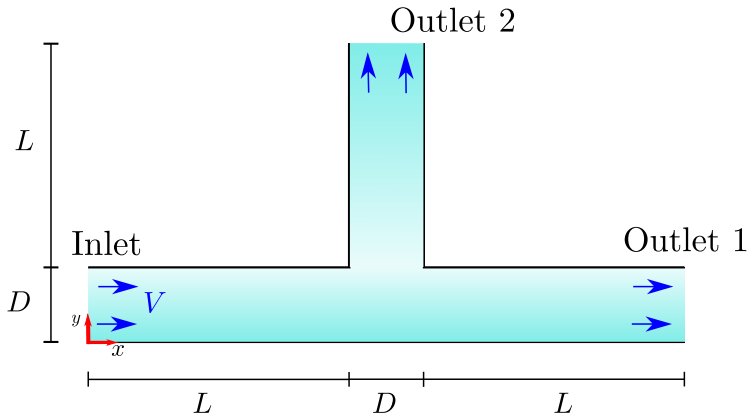
```
18 //+
19 Curve Loop(1) = {3, 4, -2, -1};
20 //+
21 Plane Surface(1) = {1};
22 //+
23 Recombine Surface {1};
24 //+
25 Transfinite Curve {3, 2} = 10 Using Progression 1;
26 //+
27 Transfinite Curve {4, 1} = 100 Using Progression 1;
28 //+
29 Transfinite Surface {1};
30 //+
31 Extrude {0, 0, 0.01} {
32   Surface{1}; Layers{1}; Recombine;
33 }
34 //+
35 Physical Surface("inlet") = {2};
```

```
36 //+
37 Physical Surface("topWall") = {3};
38 //+
39 Physical Surface("bottomWall") = {5};
40 //+
41 Physical Surface("frontAndBack") = {6, 1};
42 //+
43 Physical Surface("outlet") = {4};
44 //+
45 Physical Volume("internal") = {1};
```


1.4e-02
0.01
0.005
0.0e+00
U Magnitude

Sumário

Now we are going to investigate the T-Junction problem.



$D = 0.01\,m$, $L = 0.1\,m$, $V = 0.01\,m/s$, $\nu = 10^{-6}\,m^2/s$.
$Re = VD/\nu = 100$

The fluid enters the domain through the inlet and we have now two outlets: outlet 1 and outlet 2.

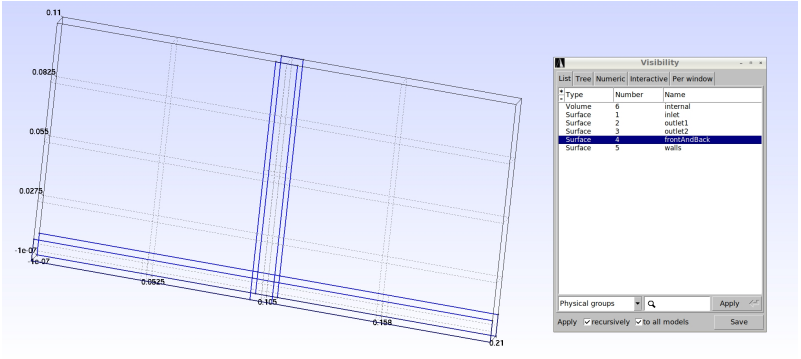
We want to investigate how the flow divides. The flow rate in each outlet will depend on the Re number.

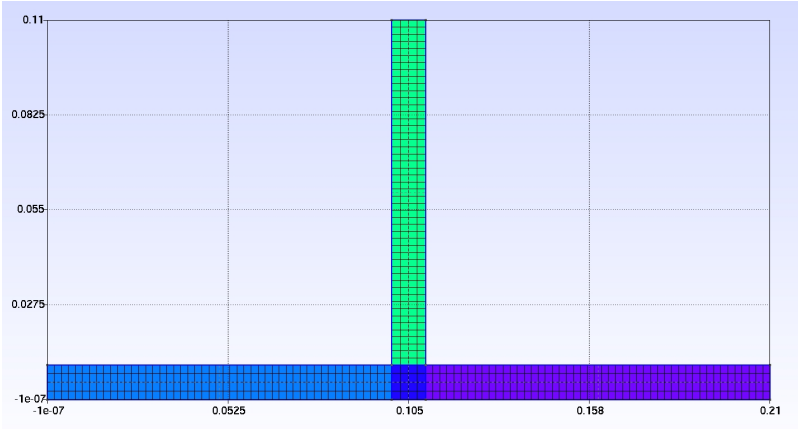
We will use the parallel plate simulation as an initial configuration for this case.

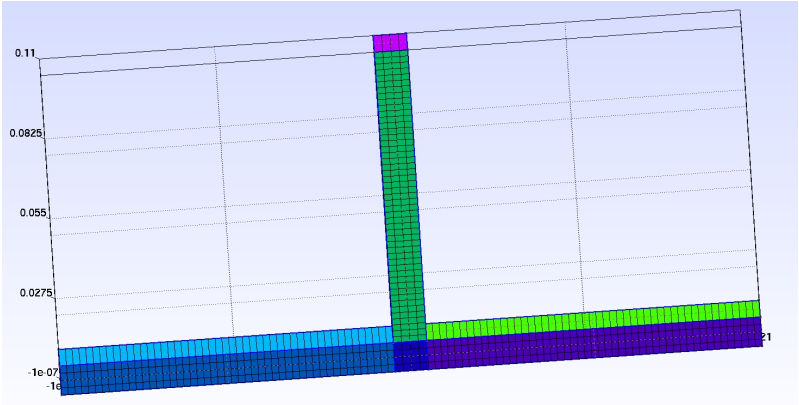
The *Gmsh* code to generate the geometry and the mesh are presented in the next slides, and also the simulation configuration.

In Gmsh, one of the most challenging steps is assigning names to surfaces (using Physical Groups > Add > Surface).

To ensure that you have selected the correct surfaces, you can use the Visibility tool in the Tools menu.




0.11
0.0825
0.055
0.0275
-1e-07
-1e-07
0.0525
0.105
0.158
0.21


0.11
0.0825
0.055
0.0275
-1e-07

The complete *Gmsh* code (*tJunction.geo*):

```
SetFactory("OpenCASCADE");

L = 0.1;
D = 0.01;
dx = 0.002;   // Size of the mesh cell.

//+
Point(1) = {0, 0, 0, 1.0};
//+
Point(2) = {L, 0, 0, 1.0};
//+
Point(3) = {L+D, 0, 0, 1.0};
//+
Point(4) = {L+D+L, 0, 0, 1.0};
//+
Point(5) = {0, D, 0, 1.0};
//+
Point(6) = {L, D, 0, 1.0};
```

```
19 //+
20 Point(7) = {L+D, D, 0, 1.0};
21 //+
22 Point(8) = {L+D+L, D, 0, 1.0};
23 //+
24 Point(9) = {L, D+L, 0, 1.0};
25 //+
26 Point(10) = {L+D, D+L, 0, 1.0};
27 //+
28 Line(1) = {1, 2};
29 //+
30 Line(2) = {2, 3};
31 //+
32 Line(3) = {3, 4};
33 //+
34 Line(4) = {5, 6};
35 //+
36 Line(5) = {6, 7};
37 //+
38 Line(6) = {7, 8};
39 //+
40 Line(7) = {9, 10};
```

```
41 //+
42 Line(8) = {1, 5};
43 //+
44 Line(9) = {2, 6};
45 //+
46 Line(10) = {6, 9};
47 //+
48 Line(11) = {3, 7};
49 //+
50 Line(12) = {7, 10};
51 //+
52 Line(13) = {4, 8};
53 //+
54 Curve Loop(1) = {1, 9, -4, -8};
55 //+
56 Plane Surface(1) = {1};
57 //+
58 Curve Loop(2) = {9, 5, -11, -2};
59 //+
60 Plane Surface(2) = {2};
```

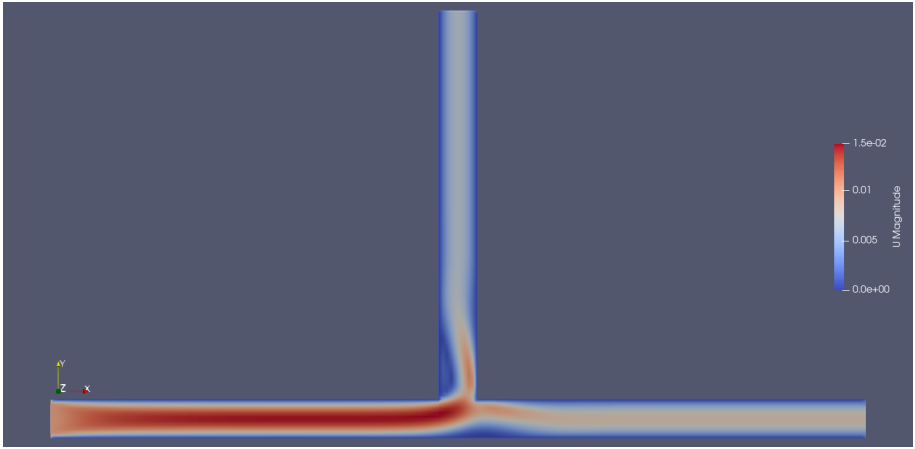
```
61 //+
62 Curve Loop(3) = {3, 13, -6, -11};
63 //+
64 Plane Surface(3) = {3};
65 //+
66 Curve Loop(4) = {5, 12, -7, -10};
67 //+
68 Plane Surface(4) = {4};
69 //+
70 Recombine Surface {1, 2, 3, 4};
71 //+
72 Transfinite Surface {1};
73 //+
74 Transfinite Surface {2};
75 //+
76 Transfinite Surface {3};
77 //+
78 Transfinite Surface {4};
79 //+
80 Transfinite Curve {1, 4} = L/dx Using Progression 1;
81 //+
82 Transfinite Curve {6, 3} = L/dx Using Progression 1;
```

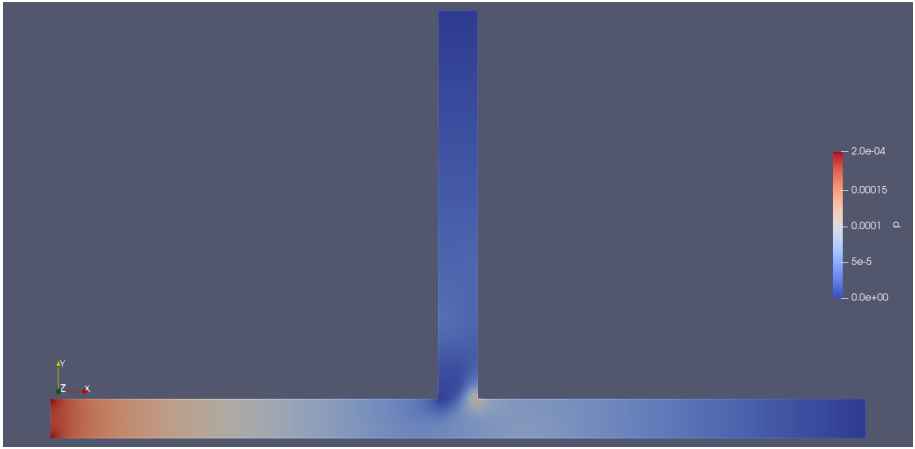
```
//+
Transfinite Curve {12, 10} = L/dx Using Progression 1;
//+
Transfinite Curve {8, 9, 11, 13} = D/dx Using Progression 1;
//+
Transfinite Curve {2, 5, 7} = D/dx Using Progression 1;
//+
Extrude {0, 0, D} {
  Surface{1}; Surface{2}; Surface{3}; Surface{4}; Layers{1}; Recombine;
}
//+
Physical Surface("inlet") = {8};
//+
Physical Surface("outlet1") = {15};
//+
Physical Surface("outlet2") = {19};
//+
Physical Surface("frontAndBack") = {17, 3, 13, 2, 9, 1, 21, 4};
//+
Physical Surface("walls") = {16, 14, 12, 5, 7, 20, 18};
//+
Physical Volume("internal") = {1, 2, 3, 4};
```

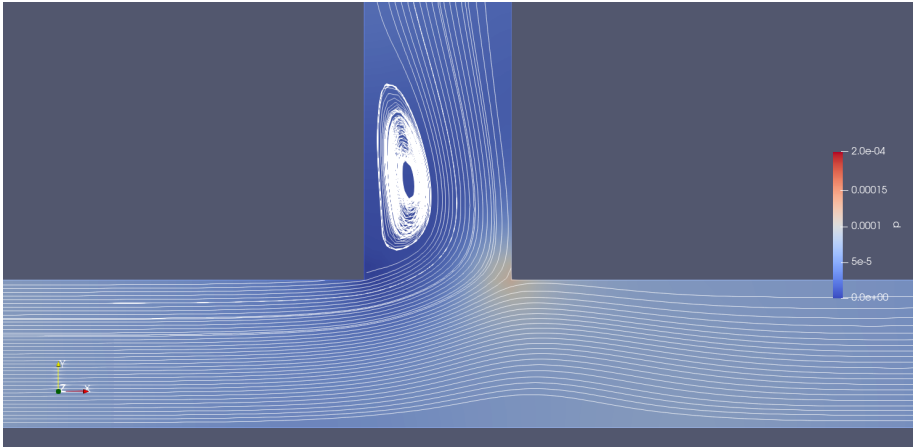
In the case configuration (OpenFOAM), we need to include *outlet1* and *outlet2* in the p and U files, and also include *walls*.

You can find the complete case configuration in the *simulations* folder.

Some results can be seen in the next slides.


1.5e-02
0.01
0.005
0.0e+00
U Magnitude
Y
Z
X


2.0e-04
0.00015
0.0001
5e-5
0.0e+00
p
Y
Z
X


2.0e-04
0.00015
0.0001
5e-5
0.0e+00
a
Y
Z
X

You can find the flow rate for each outlet in the *postProcessing* folder, created during the simulation run.

There are many postProcess functions that are available to use in OpenFOAM. Execute the command

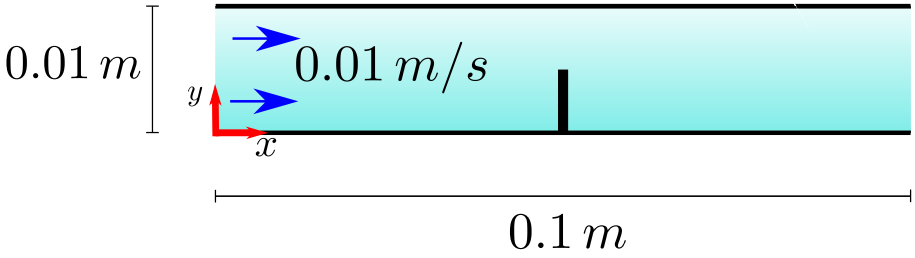
```
$: foamPostProcess -list
```

To know more details about a function:

```
$: foamInfo <function name>
```

Sumário

Last problem of this lecture: flow between parallel plates with internal obstacle (we will call the obstacle a baffle).



Simply creating an internal surface in *Gmsh* isn't enough for OpenFOAM.

Key considerations:

- OpenFOAM won't automatically recognize internal surfaces as boundaries
- Special treatment is required for internal surfaces

In OpenFOAM terminology, these internal surfaces are called ***baffles***.

Implementation Steps:

1. Create the mesh in Gmsh with the internal surface properly named/labeled

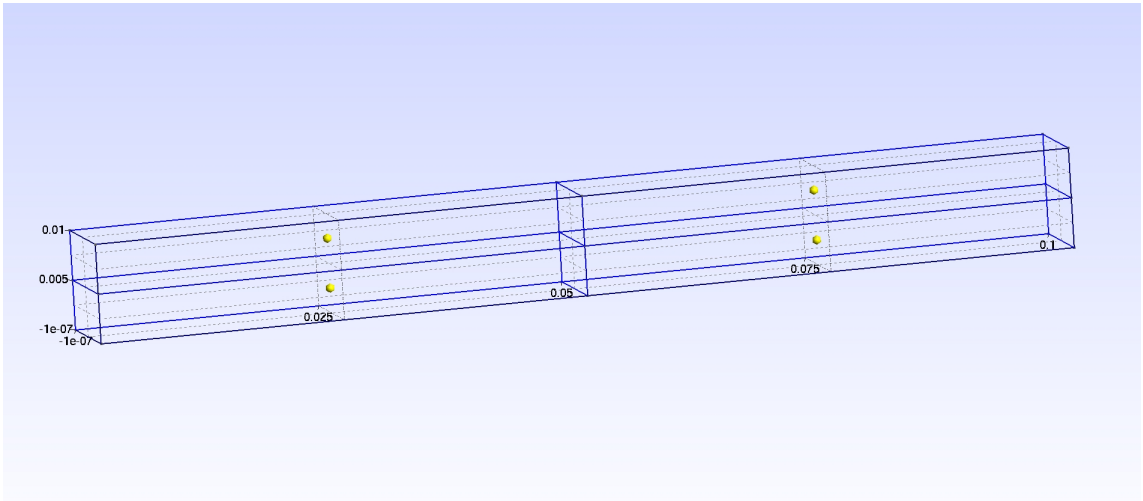
2. Convert the mesh using *gmshToFoam*

3. Execute the *createBaffles* utility:

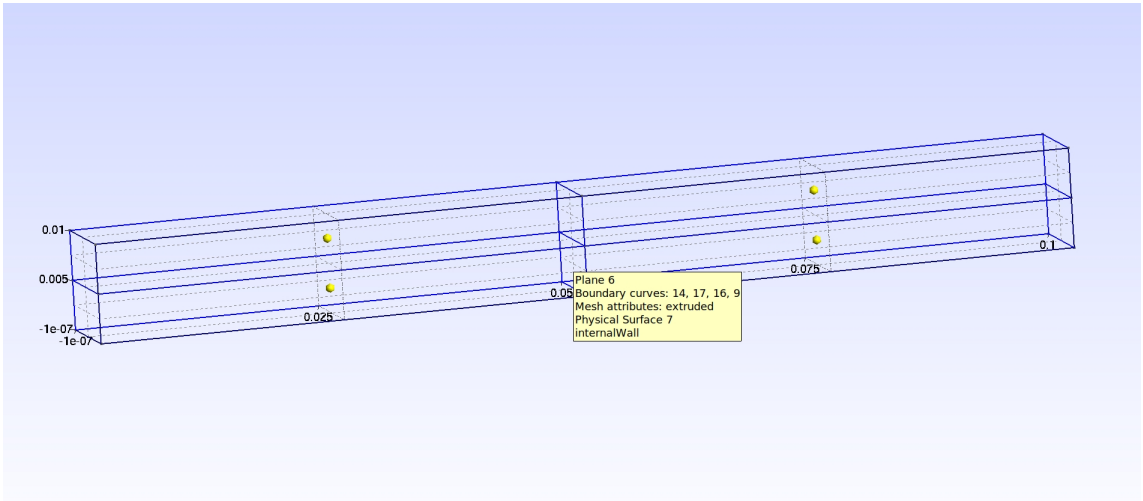
   $ createBaffles

Configuration file: *system/createBafflesDict*

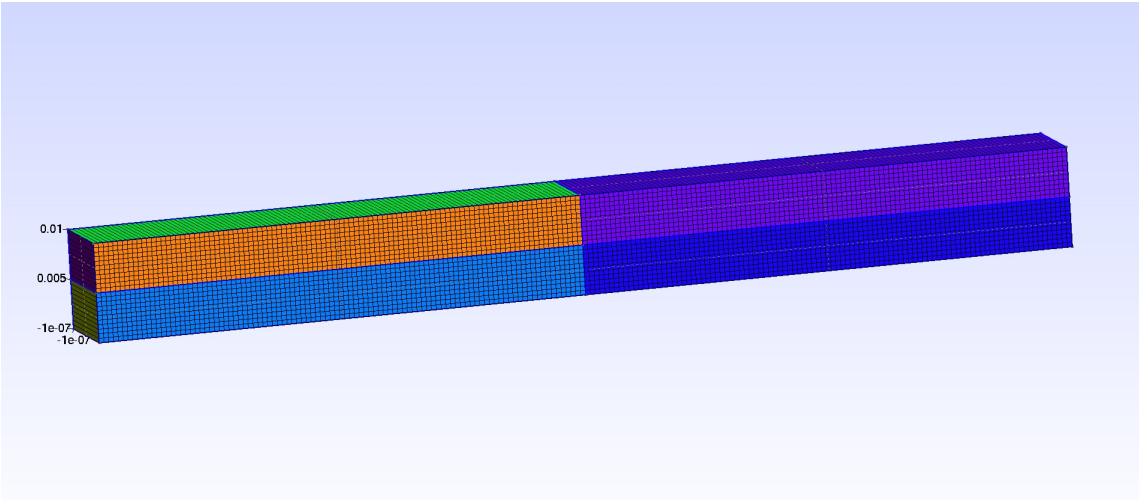
Gmsh.



Gmsh.


0.01
0.005
-1e-07
-1e-07
0.025
0.05
0.075
0.1
Plane 6
Boundary curves: 14, 17, 16, 9
Mesh attributes: extruded
Physical Surface 7
internalWall

Gmsh.

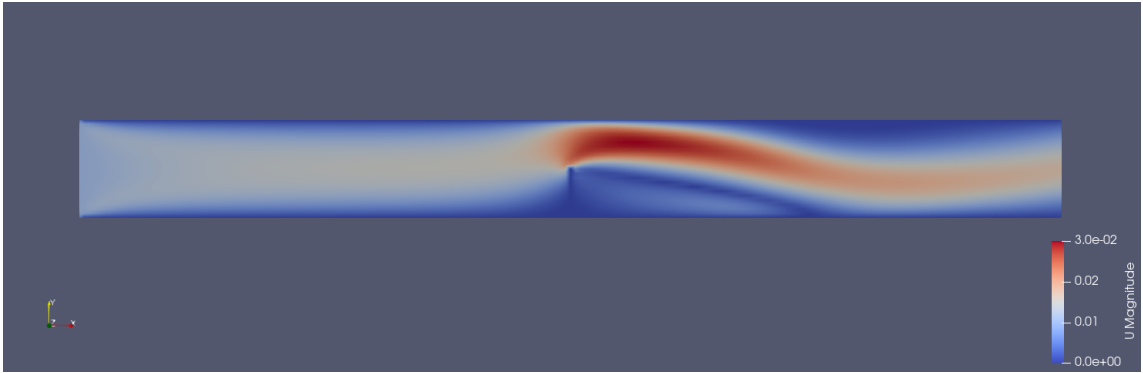


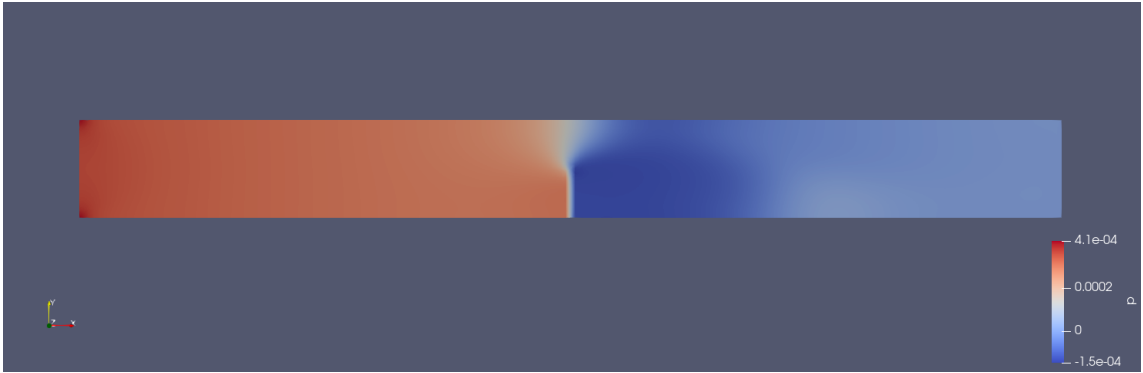
system / createBafflesDict

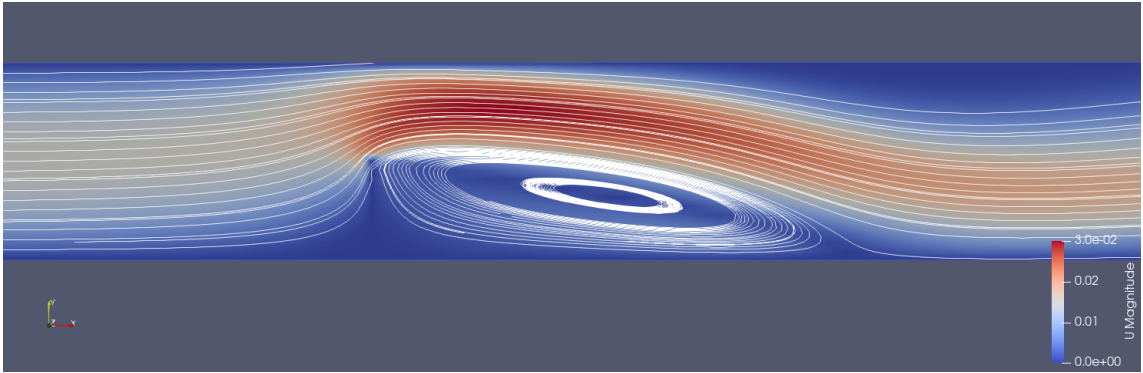
```
internalFacesOnly true;
baffles
{
    internalWall
    {
        type        faceZone;
        zoneName    internalWall;

        owner
        {
            name        internallWallLeft;
            type        mappedWall;
            neighbourPatch internallWallRight;
        }

        neighbour
        {
            name        internallWallRight;
            type        mappedWall;
            neighbourPatch internallWallLeft;
        } } }
```

Commands:

- `$: gmshToFoam` (convert the mesh to OpenFOAM format)
- `$: createBaffles` (then you need to edit the boundary file)
- `$: foamRun` (run the simulation)
- `$: paraFoam` (visualize results in ParaView)


3.0e-02
0.02
0.01
0.0e+00
U Magnitude
Y
Z
X


4.1e-04
0.0002
0
-1.5e-04
p


3.0e-02
0.02
0.01
0.0e+00
U Magnitude
Y
Z
X

Note: Baffles are particularly important for our application as **the membrane will be implemented as a baffle.**

However, unlike a typical solid wall baffle, our membrane configuration will allow controlled fluid flux across it.

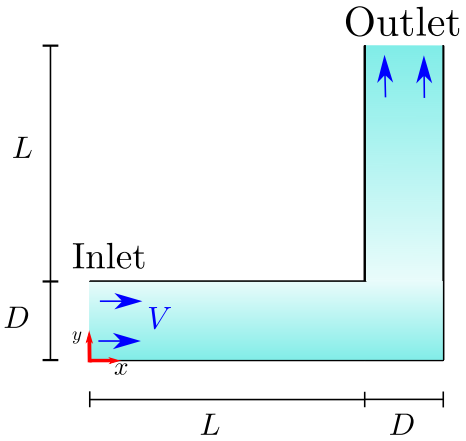
We will explore the detailed implementation in the next lectures.

Sumário

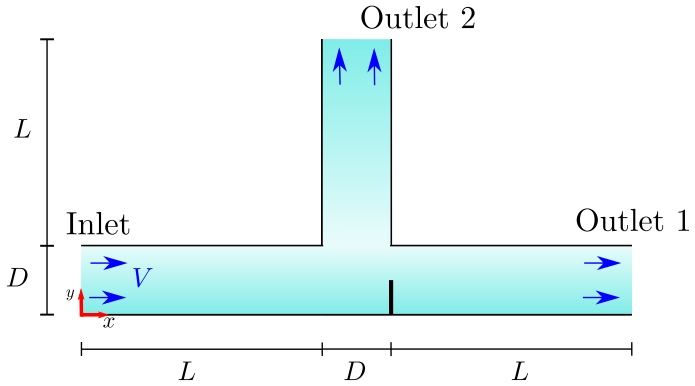
Problem 1. In the T-Junction simulation, investigate the ratio between the volumetric flow rate in outlets 1 and 2 (Q_1/Q_2) as a function of Re. Use $Re = 1, 10, 50, 100$ and 200. Get the results in the permanent regime.

Problem 2. Run the simulation of the flow between parallel plates with two obstacles (baffles). Make some plots of velocity, pressure and streamlines.

Problem 3. Simulate the flow of water in the following geometry. Use $L = 0.1\,cm$, $D = 0.01\,cm$ and $V = 0.01\,m/s$. Make plots of the velocity profile in different positions.



Problem 4. T-Junction with baffle. Simulate the flow in a T-Junction with an obstacle at the entrance of outlet 1. Use $Re = 100$. Compare the flow rates in both outlets.



Sumário

Key topics from **Lecture 01**.

- Mesh basics:
  - Properties
  - Mesh generation
- Introduction to Gmsh
- Simulation setup
  - Set fields
  - Baffles
- Simulations
  - Parallel plates
  - T-Junction

Next Lecture:

- Mesh generation:
  - snappyHexMesh
- Simulations:
  - T-Junction with membrane