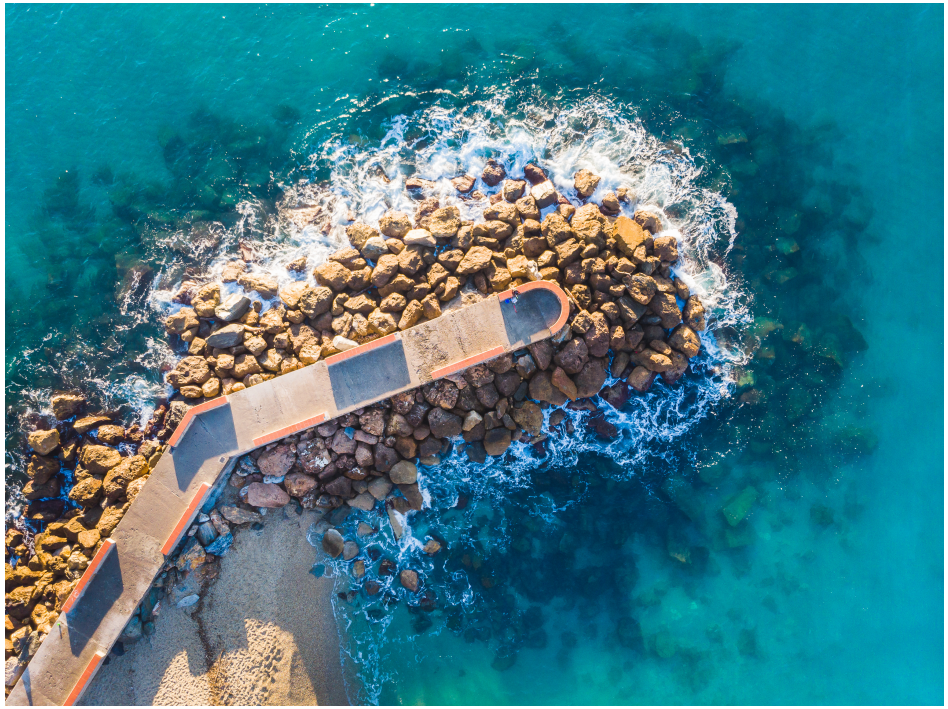
A continuous forcing immersed boundary approach to solve the VARANS equations in a volumetric porous region

Marco Vergassola

Offshore Engineering section/Hydraulic Engineering department,
Delft University of Technology,
Delft, Netherlands

January 20, 2022

Offshore and coastal porous structures



Breakwaters

Fish farming

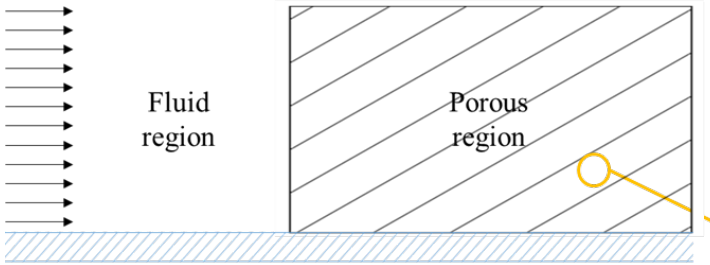
Offshore platform

Agenda

- Theory
  - Pressure drop across porous barrier
  - Immersed boundary method
- Existing solvers
  - The `olaFlow` toolbox
  - The IBM library of `foam-extend`
- Implementation of the `porousPimpleIbFoam` solver
- Verification of the `porousPimpleIbFoam` solver
- Hands-on tutorial

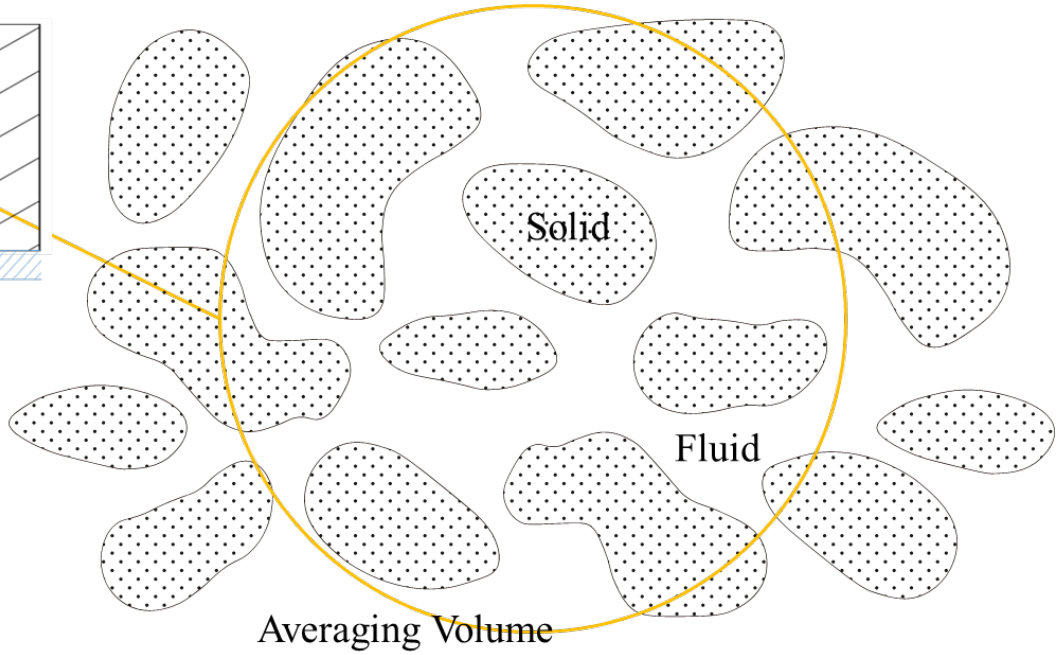
Porosity model

- Microscopic vs Macroscopic approach



$$\langle a \rangle^f = \frac{1}{V_f} \int_{V_f} a dV \,, \quad \text{where}$$

$$\langle a \rangle = n \langle a \rangle^f \quad \text{and} \quad n = V_f / V$$



The macroscopic porous model

- Darcy-Forchheimer model:

$$I = au - bu|u| - c\frac{\partial u}{\partial t}$$

where (van Gent, 1995)

$$a = \frac{\alpha}{\rho}\frac{(1-n)^3}{n^2}\frac{\mu}{D_{50}^2}, \qquad b = \beta\left(1+\frac{7.5}{\mathrm{KC}}\right)\frac{1-n}{n^2}\frac{1}{D_{50}}$$

- VARANS equations proposed by del Jesus et al., 2012

$$\frac{\partial}{\partial x_i}\frac{u_i}{n} = 0,$$

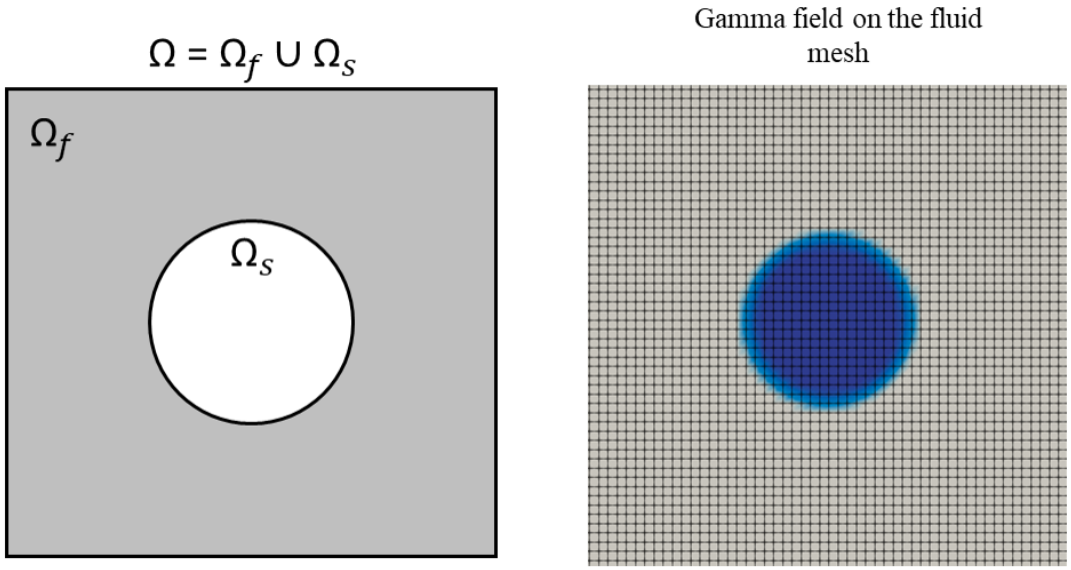
$$\frac{\partial}{\partial t}u_i + u_j\frac{\partial}{\partial x_j}\frac{u_i}{n} = -\frac{n}{\rho}\frac{\partial}{\partial x_i}p + ng_i + n\frac{\partial}{\partial x_j}\left(\nu\frac{\partial}{\partial x_i}\frac{u_i}{n}\right) - au_i - bu_i|u_i| - c\frac{\partial}{\partial t}u_i$$

The porous models in OpenFOAM

- Originally developed for breakwaters or similar structures:
  - Porous models developed for fixed objects
  - Dynamic porous regions not available
  - They require a conformal mesh
- Nowadays, porous (or perforated) elements are used in many offshore applications
  - These elements can be treated as thin porous regions
  - There is a need to simulate moving porous regions
  - A porous model based on a conformal mesh is not ideal

Immersed boundary method

- Good for:
  - Moving objects
  - Complex geometries
- Two main categories:
  - Continuous forcing methods
  - Discrete forcing methods (`foam-extend`)



General schematic of the immersed boundary method

Continuous forcing approach

- General forcing term formulation

$$F = \beta \gamma_s \left(u_s - u \right)$$

where $\gamma_s = V_s / V$ is the solid concentration field.

- The relaxation factor β can take many forms. For instance (Viré et al., 2012):

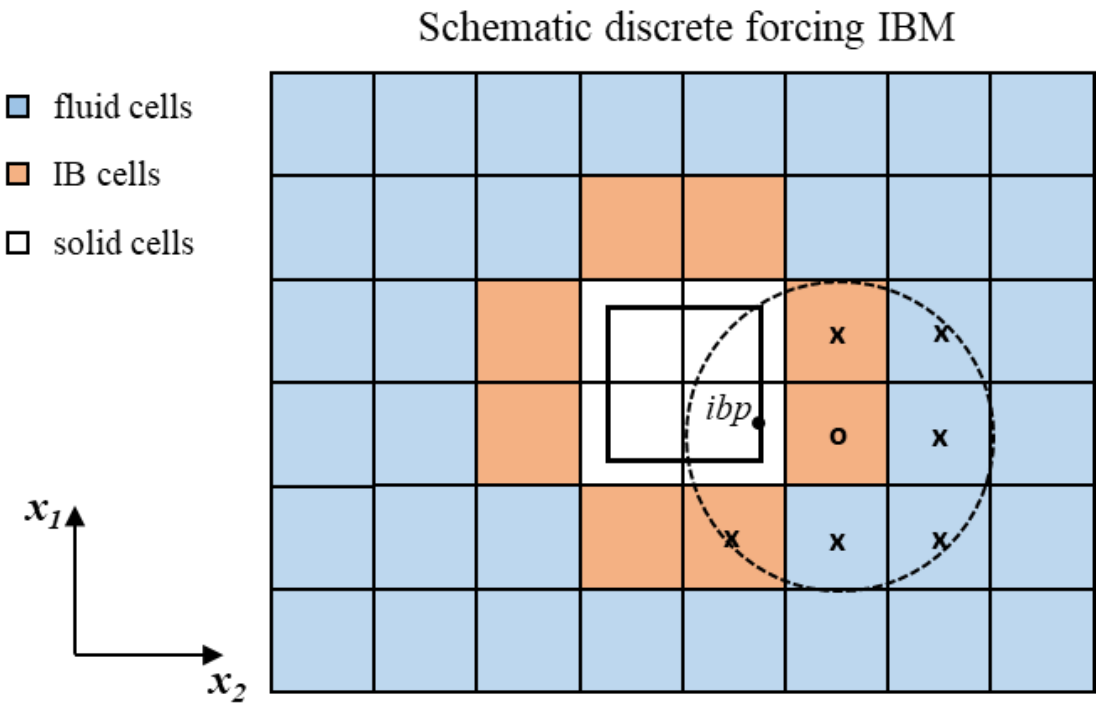
$$\beta = \max \left(\frac{\rho_f}{\Delta t}, \frac{\nu}{L^2} \right)$$

Discrete forcing approach

- Ghost-cells method: fluid not resolved inside IB
- BC for ϕ comes from minimization of

$$\sum_{i=1}^{m} w_i \left(\phi - \tilde{\phi}^2\right)$$

- Formulation of $\tilde{\phi}$ depends on interpolation
- w_i are weighting factors



Schematic IBM `foam-extend 3.2` or newer

Immersed boundary method

- Continuous forcing method
  - Forcing term similar to porous model
  - No mass conservation problems
  - Low accuracy, first-order
- Discrete forcing method (Ghost cell)
  - Flow inside IB not solved
  - Assumes presence of a boundary condition
  - Higher accuracy, second-order

The continuous forcing method is preferred for its simplicity despite the lower accuracy

olaFlow toolbox

- Many options available, `olaFlow` is one of them
- Open source project (previously `IHFOAM` and `olaFoam`)
  - Simulation of wave dynamics
  - Interaction with porous, floating and static structures
  - `olaFlow` solver is based on `interFoam`. Main difference in UEqn.H file
- The momentum equation proposed by del Jesus et al., 2012 is implemented:

$$\frac{\partial}{\partial t}u_i + u_j\frac{\partial}{\partial x_j}\frac{u_i}{n} = -\frac{n}{\rho}\frac{\partial}{\partial x_i}p + ng_i + n\frac{\partial}{\partial x_j}\left(\nu\frac{\partial}{\partial x_i}\frac{u_i}{n}\right) - au_i - bu_i|u_i| - c\frac{\partial}{\partial t}u_i$$

- After multiplying each term by ρ/n and rearranging

$$(1+c)\frac{\partial}{\partial t}\frac{\rho u_i}{n} + \frac{u_j}{n}\frac{\partial}{\partial x_j}\frac{\rho u_i}{n} = -\frac{\partial}{\partial x_i}p + \rho g_i + \frac{\partial}{\partial x_j}\left(\mu\frac{\partial}{\partial x_i}\frac{u_i}{n}\right) - a\frac{u_i}{n} - b\frac{u_i}{n}\left|\frac{u_i}{n}\right|$$

olaFlow toolbox

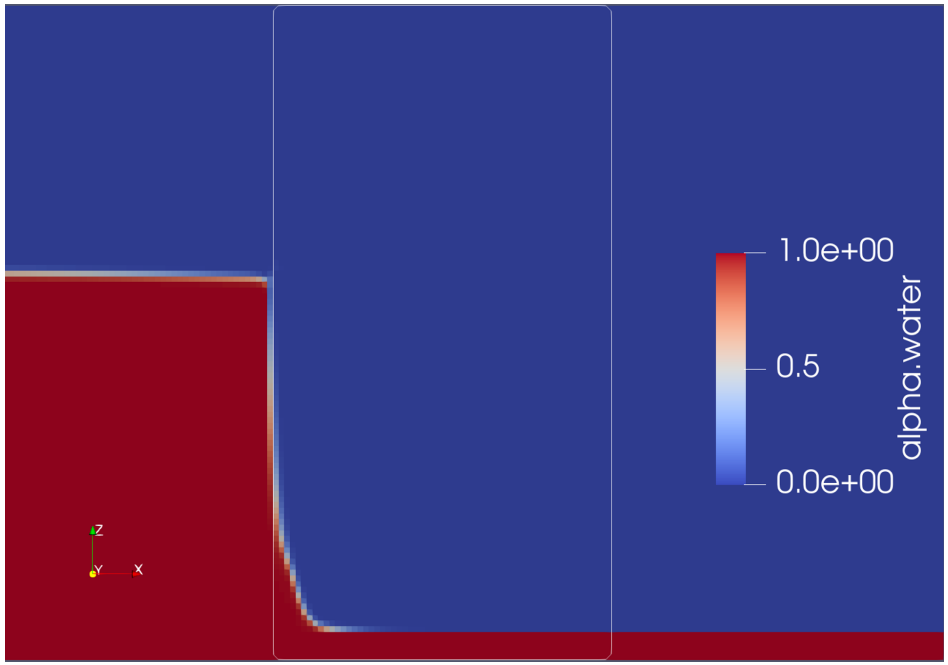
- The momentum equation solved by `olaFlow`

$$(1+c)\frac{\partial}{\partial t}\frac{\rho u_i}{n} + \frac{u_j}{n}\frac{\partial}{\partial x_j}\frac{\rho u_i}{n} = -\frac{\partial}{\partial x_i}p + \rho g_i + \frac{\partial}{\partial x_j}\left(\mu\frac{\partial}{\partial x_i}\frac{u_i}{n}\right) - a\frac{u_i}{n} - b\frac{u_i}{n}\left|\frac{u_i}{n}\right|$$

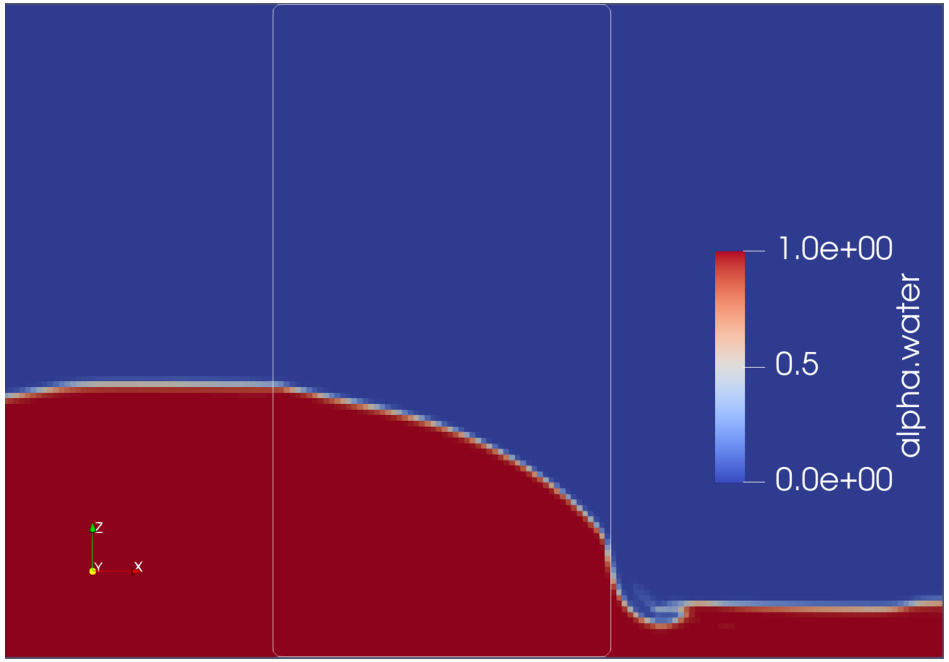
```
fvVectorMatrix UEqn
(
    (1.0 + cPorField) / porosity * fvm::ddt(rho, U)
    + 1.0/porosity * fvm::div(rhoPhi/porosityF, U)
    - fvm::laplacian(muEff/porosityF , U)
    - 1.0/porosity * ( fvc::grad(U) & fvc::grad(muEff) )
    // Closure Terms
    + aPorField * pow(1.0 - porosity, 3) / pow(porosity,3)
    * mixture.mu() / pow(D50Field,2) * U
    + bPorField * rho * (1.0 - porosity) / pow(porosity,3) / D50Field
    * mag(U) * U * (1.0 + useTransMask * 7.5 / KCPorField)
==
    fvOptions(rho, U)
);
```

olaFlow toolbox

- The `olaFlow` toolbox can be used to simulated a dam break event including a volumetric porous region



`alpha.water` field at t=0.05s



`alpha.water` field at t=1.0s

Results from the tutorial `CR35_damBreak`

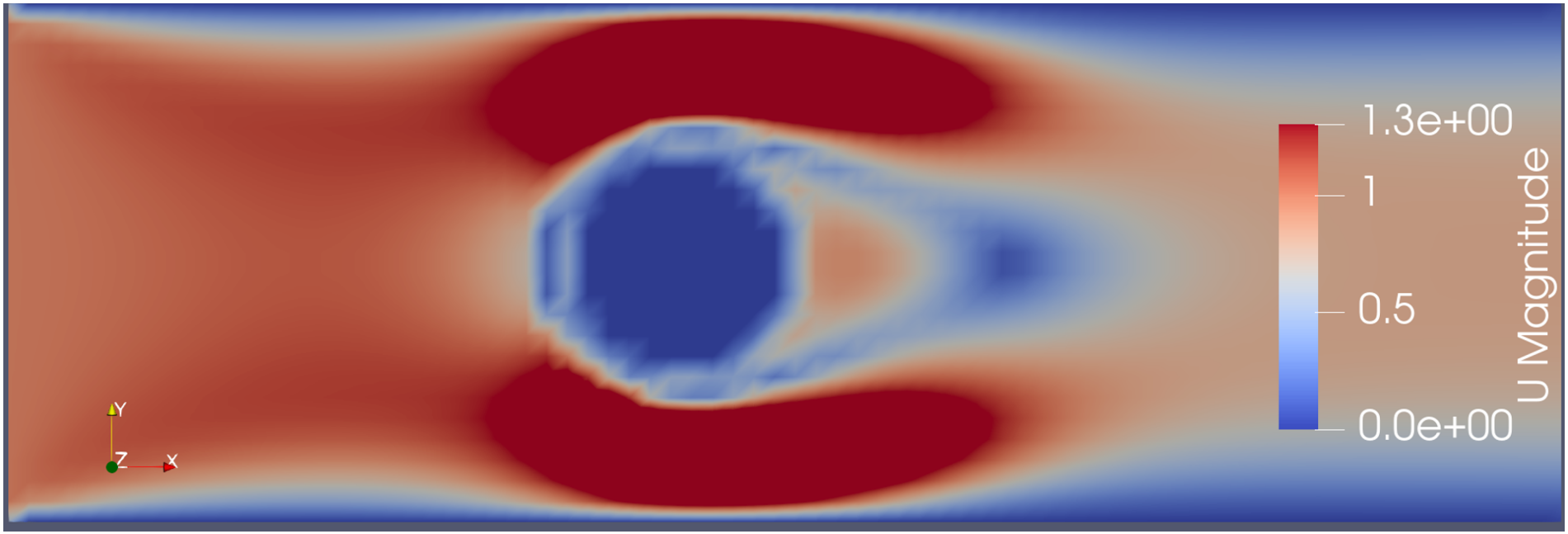
IBM foam-extend

- Discrete forcing ghost-cell method
- Quite extensive library
- Many BC for both internal and external flow
- Two transient solvers available
  - `pimpleDyMIbFoam` for incompressible flows
  - `sonicDyMIbFoam` for transonic/supersonic compressible flows

- The IBM of `foam-extend` consist of three main classes
  - `class immersedBoundaryPolyPatch` - basic mesh support for the IB mesh
  - `class immersedBoundaryFvPatch` - takes care of defining the IB mask and constructs the interpolation matrices
  - `class immersedBoundaryFvPatchFields` - is responsible for the evaluation of the boundary conditions

IBM foam-extend

- The IBM of `foam-extend` can be used to simulated a IB cylinder oscillating inside a channel



Velocity field at t=4.0s

Results from the tutorial `movingCylinderInChannelIco`

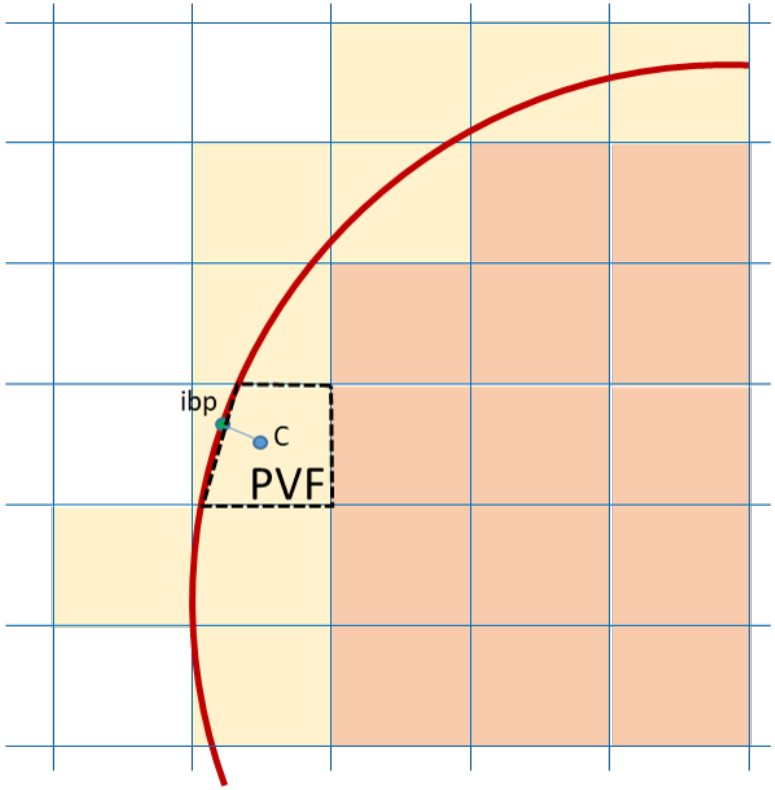
The porousPimpleIbFoam

- The new solver is based on:
  - The `OpenFOAM`'s `pimpleFoam` solver
  - The porous model implemented in `olaFlow`
  - A continuous forcing IBM where the porous drag force is the forcing term

- A porosity field n is defined
- Special treatment at the IB surface required

$$n_{\text{corr}} = \gamma + n\,(1 - \gamma)$$

- γ is the PVF (Porous Volume Fraction) field



The porousPimpleIbFoam

The structure of the solver looks as follows

```
porousPimpleIbFoam/
|-- correctPhi.H
|-- createFields.H
|-- createPorosity.H
|-- createPorousIbMask.H
|-- Make
|   |-- files
|   |-- options
|-- pEqn.H
|-- porousPimpleIbFoam.C
|-- UEqn.H
```

- PVF evaluation routine is implemented in the `createPorousIbMask.H` file
- The porosity field and its correction are found in the `createPorosity.H` file

createPorousIbMask.H

Firstly, a dictionary is created from which the IB surface is read

```
// Define dictionary
IOdictionary porousIbMaskDict
(
    IOobject
    (
        "porousIbMaskDict",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
);

// Read dictionary
fileName surfName(porousIbMaskDict.get<fileName>("surface"));
surfName.expand();
pointField outsidePts(porousIbMaskDict.lookup("outsidePoints"));
const label nCutLayers(porousIbMaskDict.getOrDefault<label>("nCutLayers", 0));
```

createPorousIbMask.H

Then the cells are classified

```
// Surface.
const triSurface surf(surfName);

// Search engine on surface
const triSurfaceSearch querySurf(surf);

// Divide the cells according to status compared to surface. Constructs sets:
// - cutCells : all cells cut by surface
// - inside   : all cells inside surface
// - outside  :   ,,      outside ,,

cellSet inside(mesh, "inside", mesh.nCells()/10);
cellSet outside(mesh, "outside", mesh.nCells()/10);
cellSet cutCells(mesh, "cutCells", mesh.nCells()/10);
```

createPorousIbMask.H

The cells sets are obtained with the function `getSurfaceSets`

```
surfaceSets::getSurfaceSets
(
    mesh,
    surfName,
    querySurf.surface(),
    querySurf,
    outsidePts,

    nCutLayers,

    inside,
    outside,
    cutCells
);
```

createPorousIbMask.H

Before computing the PVF, the γ field is defined

```
Info<< "Create immersed boundary cell mask" << endl;

volScalarField gamma
(
    IOobject
    (
        "gamma",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar( "gamma", dimless, 1.0 )
);
```

createPorousIbMask.H

Then the PVF is evaluated

```
forAll(cutCells, cellI)//loop over the cells intersecting the object surface
{
    const point& c = ctrs[cutCells.sortedToc()[cellI]];

    const pointIndexHit inter = querySurf.nearest(c, span);

    const label trii = inter.index();
    const vector normal = normals[trii]/mag(normals[trii]); //compute surface normal
    const scalar cutValue = (inter.hitPoint() - c) & (normal); //compute distance to surface
    cutCell.calcSubCell
    (
        cellI,
        cutValue,
        normal
    );
    gamma[cutCells.sortedToc()[cellI]] = cutCell.VolumeOfFluid(); //assigned field value
}
```

createPorosity.H

The `createPorousIbMask.H` file in then included in `createPorosity.H`. The fields related to the porosity model are defined and initialized

```
#include "createPorousIbMask.H"
volScalarField porosityIndex
(
    IOobject
    (
        "porosityIndex",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar( "porosityIndex", dimless, 0.0 )
);
volScalarField porosity
(
    IOobject
    (
        "porosity",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar( "porosity", dimless, 1.0 )
);
```

```
// Porosity variables
scalarList aPor, bPor, cPor, D50Por, phiPor;
// Define fields related with porosity
volScalarField aPorField = volScalarField("aPorField",
    0.0 * porosityIndex);
volScalarField bPorField = volScalarField("bPorField",
    aPorField);
volScalarField cPorField = volScalarField("cPorField",
    aPorField);
volScalarField useTransMask = volScalarField("useTrans
    ", aPorField);
volScalarField KCPorField = volScalarField("KCPorField
    ", aPorField + 1.0);

volScalarField D50Field
(
    IOobject
    (
        "D50Field",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar( "D50", dimLength, 1.0 )
);
```

createPorosity.H

Then the defined non-uniform coefficient fields are allocated

```
        forAll(porosityIndex, item)
        {
            if( porosityIndex[item] > 0.0 )
            {
                aPorField[item] = aPor[porosityIndex[item]];
                bPorField[item] = bPor[porosityIndex[item]];
                cPorField[item] = cPor[porosityIndex[item]];
                KCPorField[item] = KC;
                D50Field[item]  = D50Por[porosityIndex[item]];
                porosity[item]  = phiPor[porosityIndex[item]];
                if ( useTransient )
                {
                    useTransMask[item] = 1.0;
                }
            }
        }
```

createPorosity.H

Finally, the porosity field is corrected and written

```
        //Correct porosity field to account for porous volume fraction at interface
        Info << "\nCorrecting porosity field\n" << endl;
        porosity = gamma + (1-gamma)*porosity;

        porosityF = fvc::interpolate(porosity);

        // Write out porosity
        porosity.write();
```

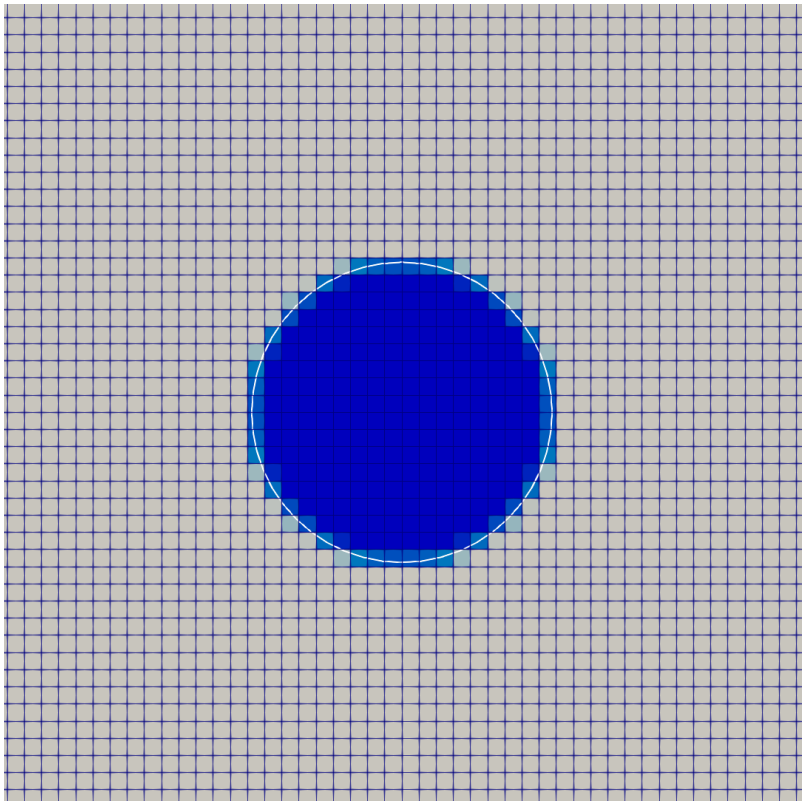
The porousPimpleIbFoam

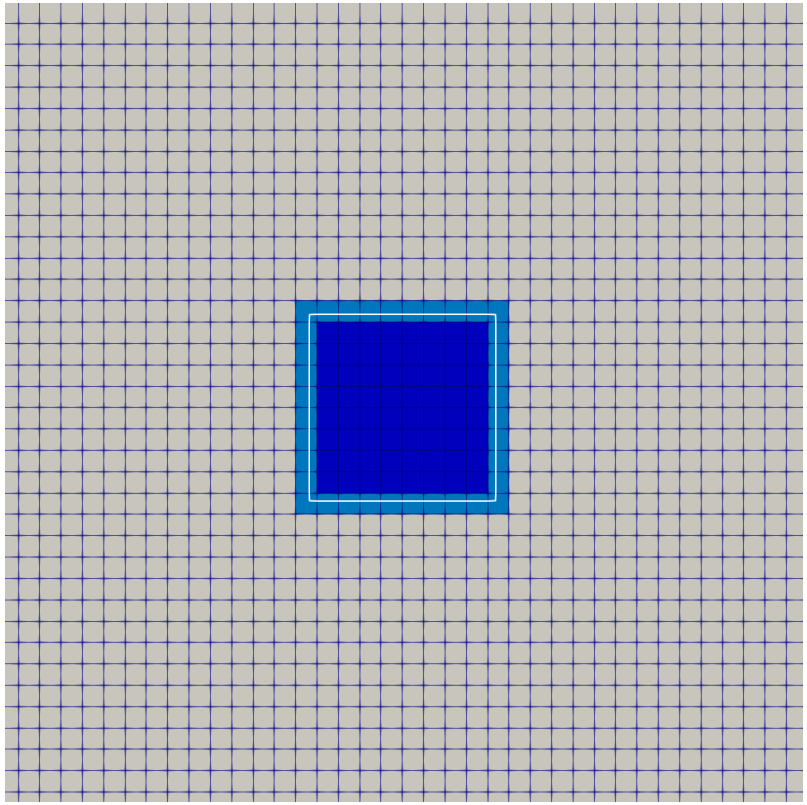
- Apart from `UEqn.H` the remaining files are almost identical to `interFoam`
- The UEqn.H file looks like the one of `olaFlow`

```
tmp<fvVectorMatrix> tUEqn
(
  (1.0+cPorField)/porosity*fvm::ddt(U)
  + 1.0/porosity*fvm::div(phi/porosityF, U)
  - fvm::laplacian(nuEff/porosityF,U)
  - 1.0/porosity*(fvc::grad(U) & fvc::grad(nuEff))
  // Closure Terms
  + aPorField*pow(1.0-porosity,3)/pow(porosity,3)*turbulence->nu()/pow(D50Field
     ,2)*U
  + bPorField*(1.0-porosity)/pow(porosity,3)/D50Field*mag(U)*U*
  // Transient formulation
  (1.0 + useTransMask * 7.5 / KCPorField)
 ==
    fvOptions(U)
);
```

Verification PVF algorithm

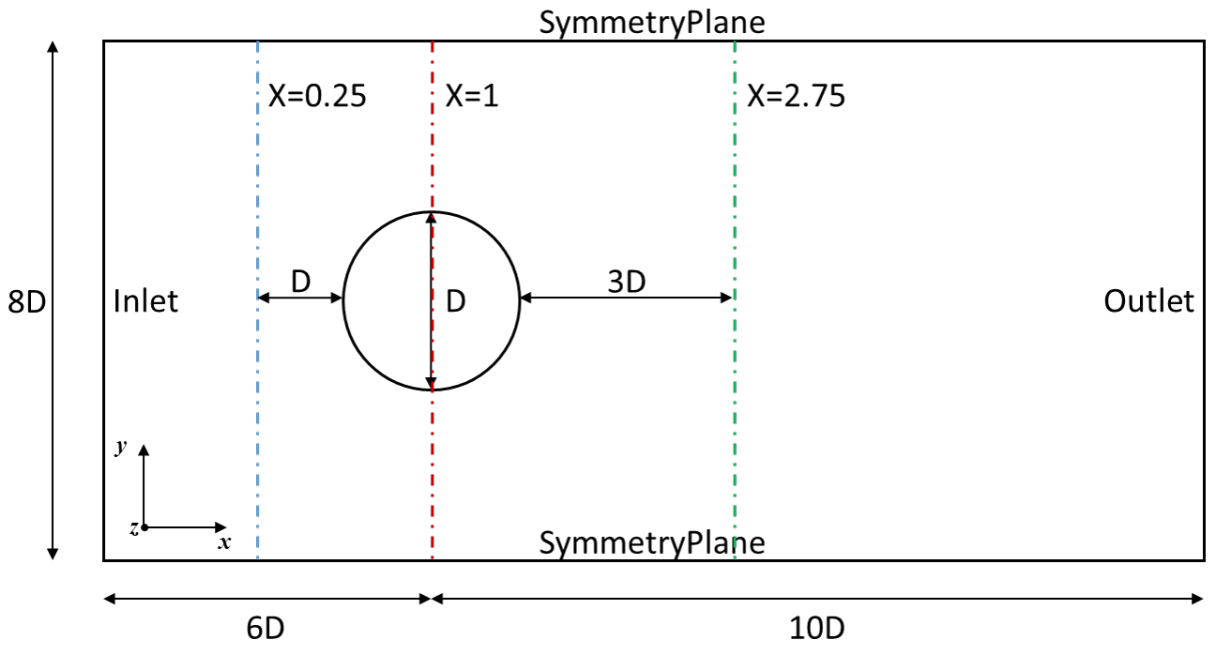
- The PVF field is computed for different shapes. Two are presented here
- The algorithm cannot handle sharp features





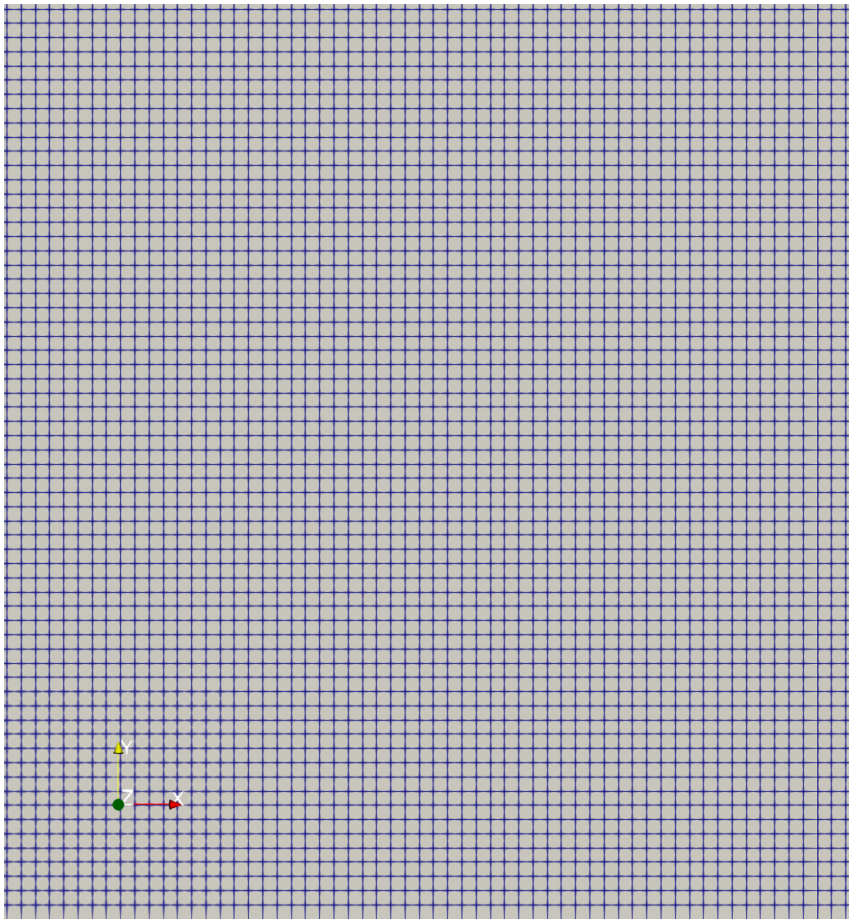
Verification setup

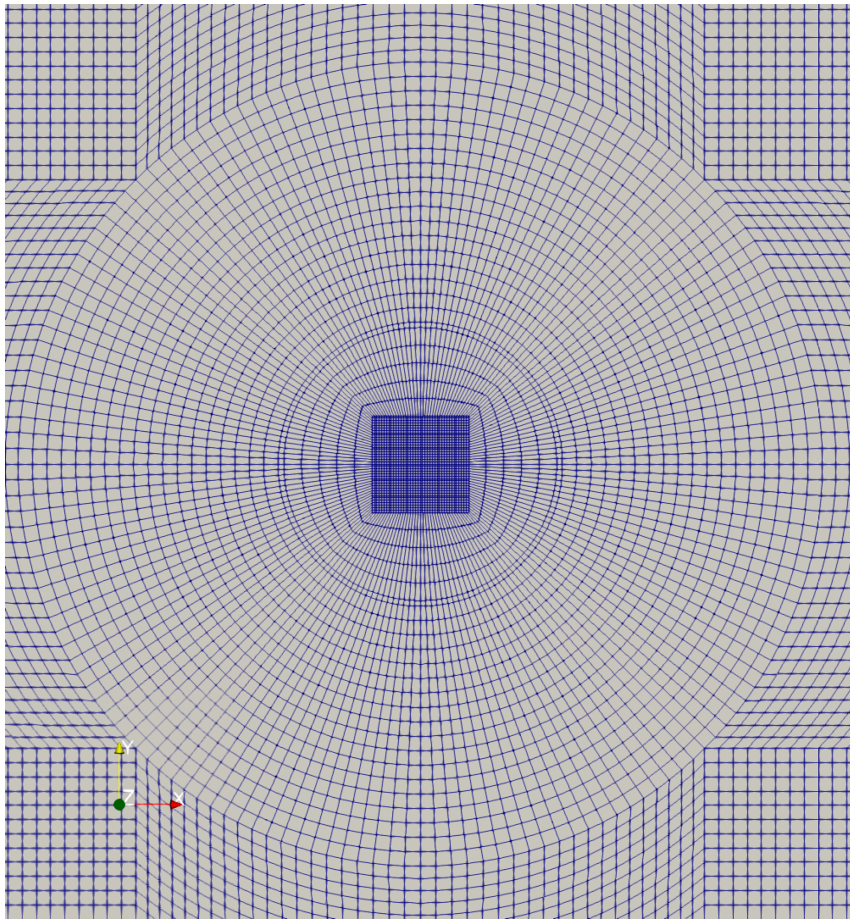
- The `porousPimpleIbFoam` solver is compared against a conformal mesh solver
- The verification case consist of a porous cylinder in a constant 2D flow at Re=100
- Comparison at both quantitative and qualitative level



Verification setup

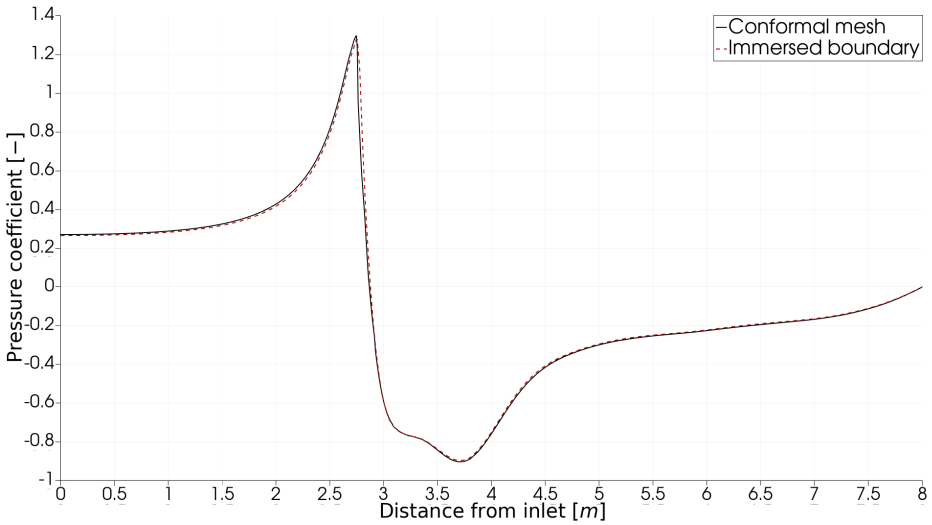
- The `porousPimpleIbFoam` solver uses a simple Cartesian mesh
- The conformal and IB meshes are compared closed to the cylinder location

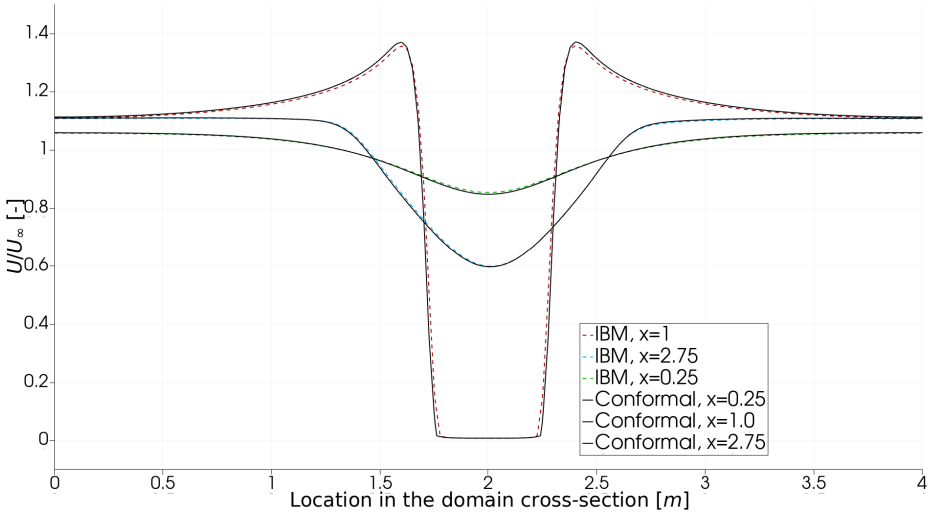




Quantitative results

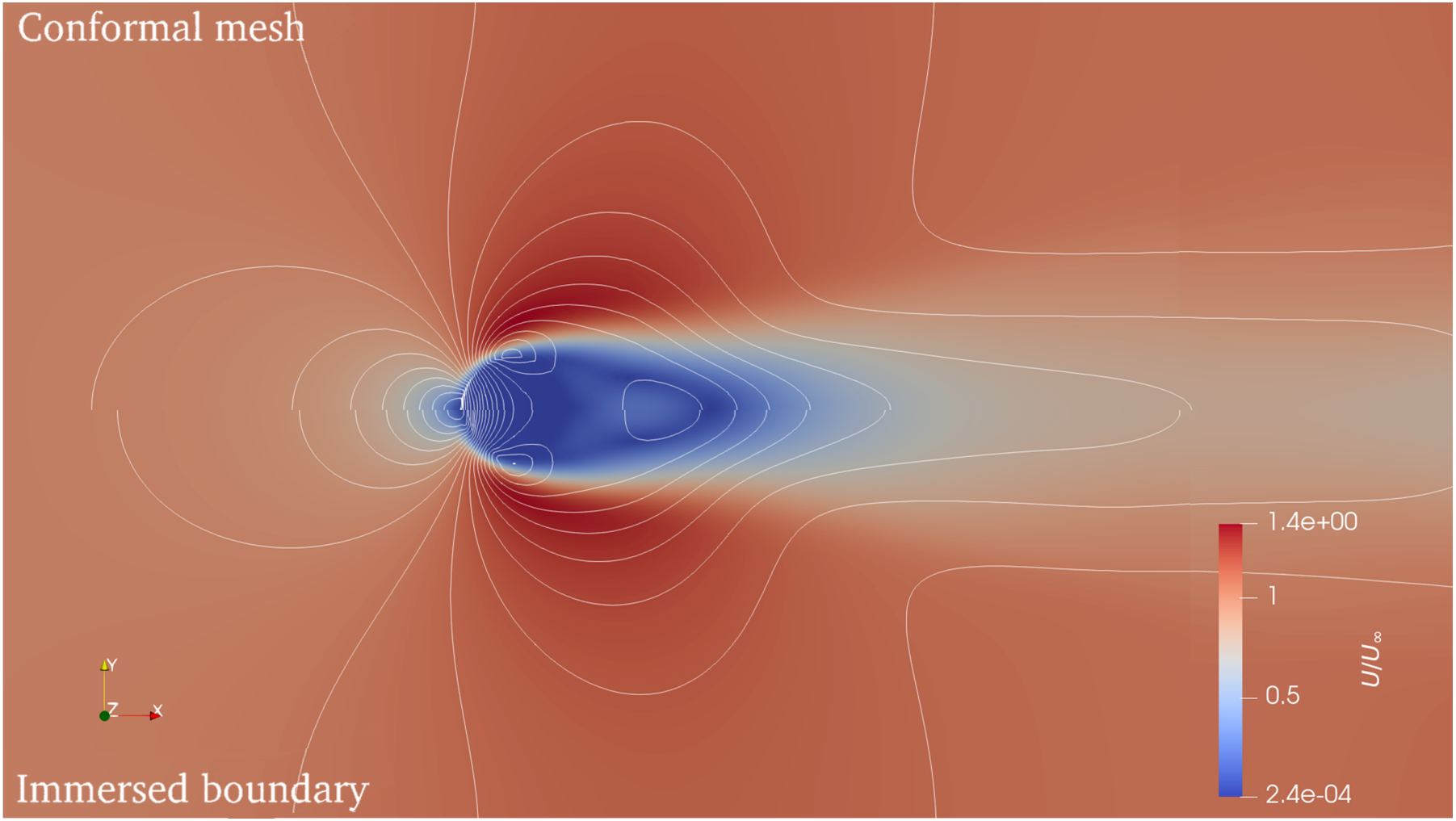
- The two solvers are compared in terms of:
  - Pressure coefficient along the central streamline
  - Velocity profiles at different locations





Qualitative results

- The velocity and pressure fields are qualitatively compared

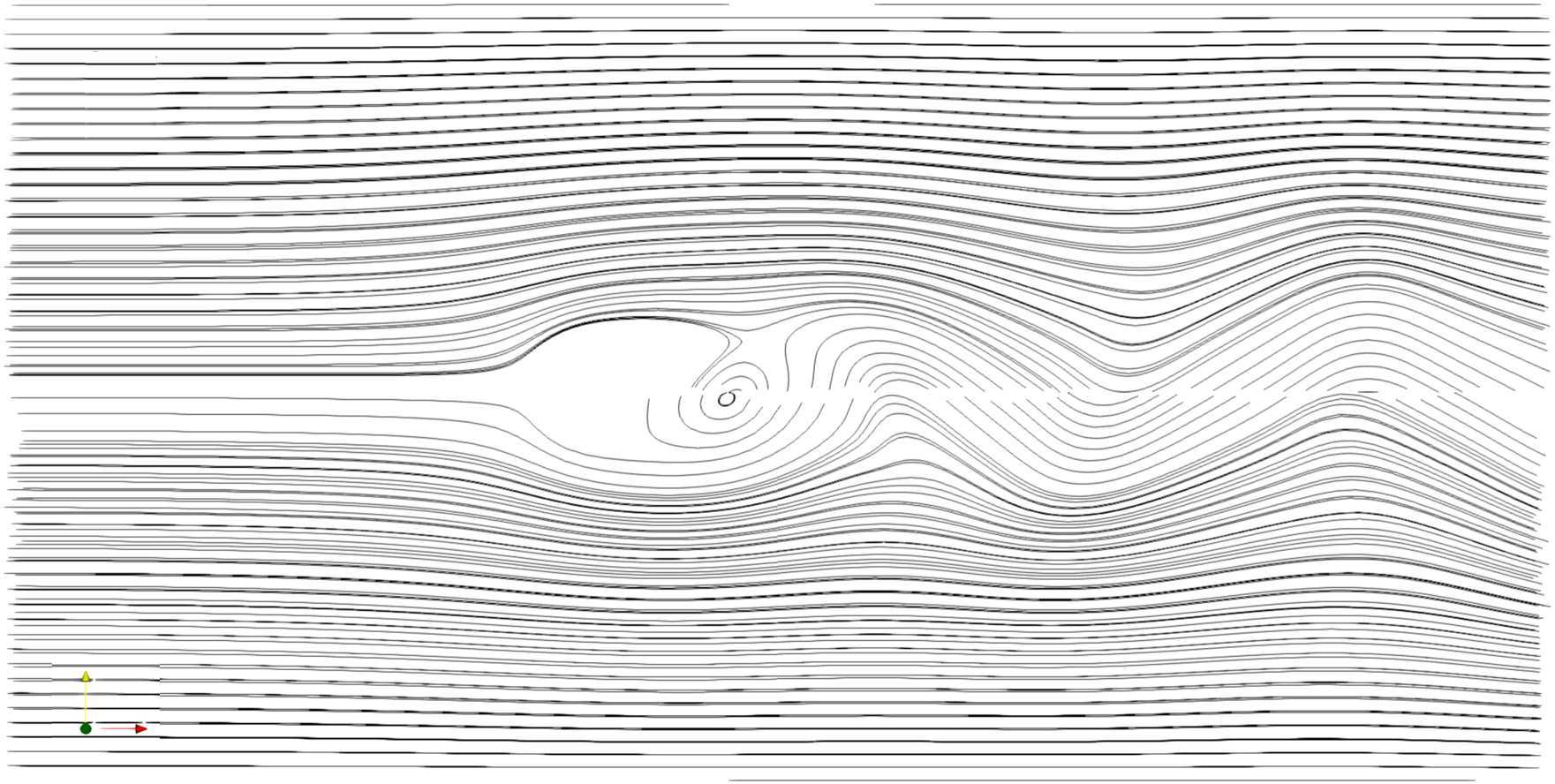


Qualitative results

- The vortex shedding is also qualitatively compared

Conformal mesh

Time: 2500 s



Immersed boundary

Hands-on tutorial

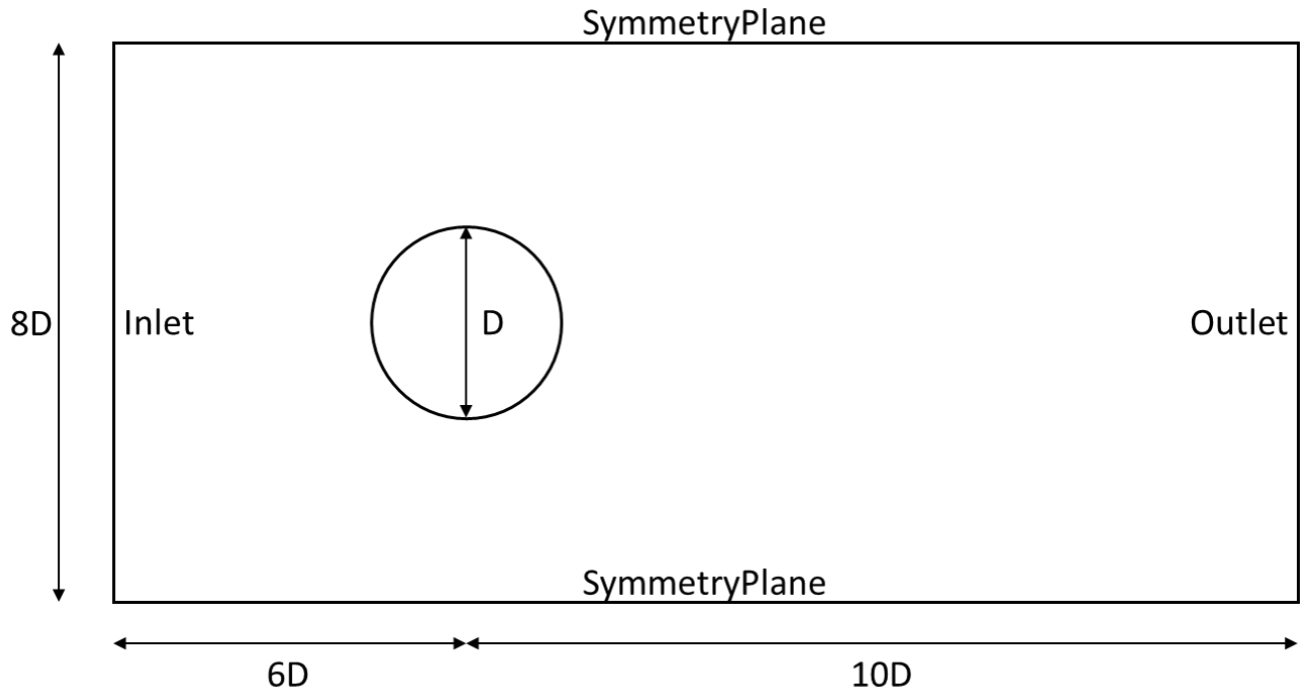
Important information:

- The next set of slides presents an hands-on tutorial
- To follow along you need `OpenFOAM v20.06` and the tutorial files (including the solver)
- All the files can be downloaded here[1]

[1] If the hyperlink doesn't work you can copy this in a web browser https://drive.google.com/drive/folders/1MXOMVoIxbO5I0McBQRUSwpaJApGFL-WC?usp=sharing.

Hands-on tutorial

- The tutorial simulates an immersed porous cylinder in a constant 2D flow at Re=40
- A porosity of 10% is chosen for the cylinder



Tutorial structure

```
porousPimpleIbFoamCase/
|--  0.org
|   |-- p
|   |-- porosityIndex
|   |__ U
|-- Allclean
|-- Allrun
|-- constant
|   |-- polyMesh
|   |-- porosityDict
|   |-- porousIbMaskDict
|   |-- transportProperties
|   |-- triSurface
|       |__ ibCylinder.stl
|-- system
    |-- blockMeshDict
    |-- controlDict
    |-- fvSchemes
    |-- fvSolution
    |__ setFieldsDict
```

Mesh generation

Firstly, we start from generating the mesh (`blockMeshDict` file)

```
scale   1;

vertices
(
    (-2 -2 -0.1)
    (6  -2 -0.1)
    (6   2 -0.1)
    (-2  2 -0.1)
    (-2 -2  0.1)
    (6  -2  0.1)
    (6   2  0.1)
    (-2  2  0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (320 160 1) simpleGrading (1 1 1)
);
```

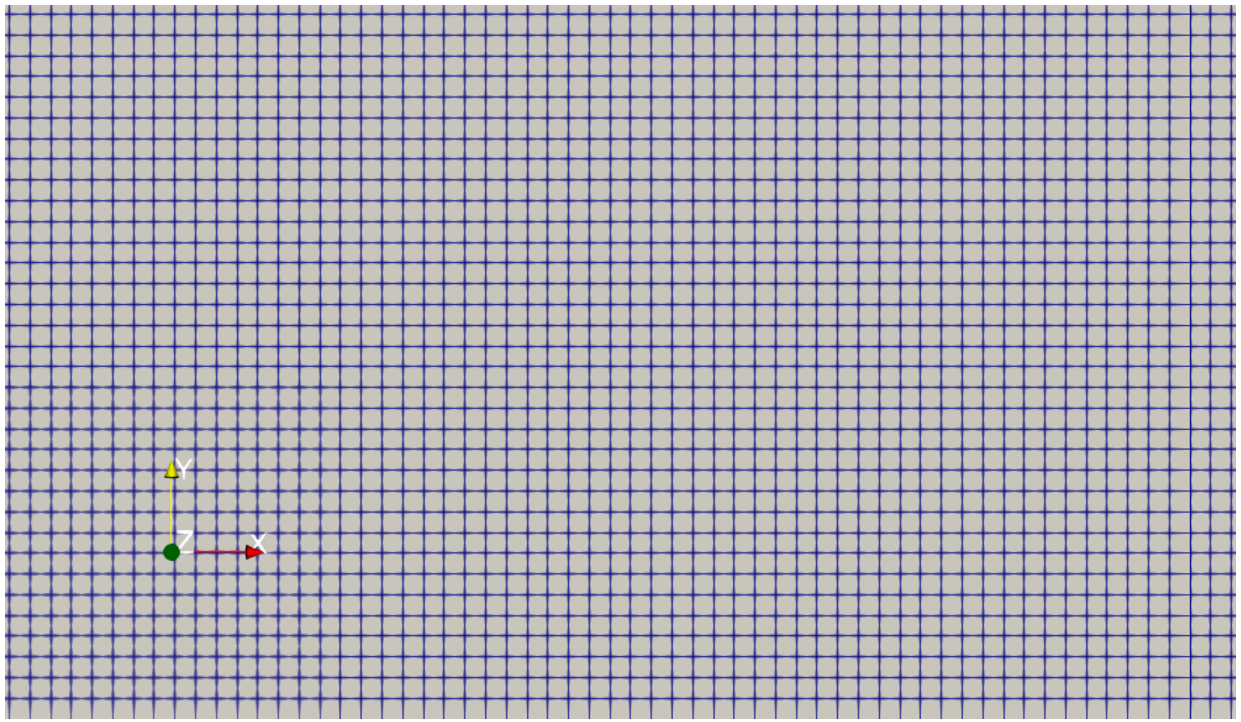
Mesh generation

Then we define the boundary patches. Only some are presented here

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

Mesh generations

- At this point it is possible to generate and visualize the mesh.
- Open a terminal window, source the `OpenFOAM` installation and type `blockMesh`
- Once the operation is done, in the same terminal type `paraFoam`
- Click apply and change the visualization from `surface` to `surface with edges`. Your mesh should look like this:



Boundary and initial conditions

Lets have a look at the pressure file (velocity has inlet/outlet inverted)

```
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           $internalField;
    }
    top
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            symmetryPlane;
    }
    frontAndBack
    {
        type empty;
    }
```

Boundary and initial conditions

The porosityIndex field is a special field used for this solver

```
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    top
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            symmetryPlane;
    }
    frontAndBack
    {
        type            empty;
    }
```

The porosity dictionary

Now we look at the porosity dictionary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      porosityDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

a               2(0 0);
b               2(0 2.35);
c               2(0 0);

D50             2(1 1);
porosity        2(1 0.1);

useTransient    true;
debugPor        false;
// ************************************************************************* //
```

The porousIbMask dictionary

Now we take care of the porousIbMask dictionary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      porousIbMaskDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

surface         "<constant>/triSurface/ibCylinder.stl";
outsidePoints   ((0 -1 0));

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

The setFieldDict dictionary

Next we move to the setFieldDict file

```
defaultFieldValues
(
    volScalarFieldValue porosityIndex 0
);
regions
(
    surfaceToCell
    {
        file            "<constant>/triSurface/ibCylinder.stl";
        outsidePoints   ((0 -1 0));
        includeCut      false;
        includeInside   true;
        includeOutside  false;
        nearDistance    0.01;
        scale           1.0;
        curvature       -100;
        useSurfaceOrientation true;
        fieldValues
        (
            volScalarFieldValue porosityIndex 1
        );
```

Hands-on tutorial

A few more details:

- The `transportProperties` file includes the properties of the fluid
- The `fvSchemes` and `fvSolution` files provides the settings for the discretization and solution of the equations
- The last `divSchemes` entry in the `fvSchemes` file is specific of this solver

```
divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div((phi|interpolate(porosity)),U) Gauss limitedLinearV 1;
}
```

The controlDict file

Before we run the tutorial. Let's have a look at the control dictionary

```
application     porousPimpleIbFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         2000;
deltaT          0.01;
writeControl    adjustableRunTime;
writeInterval   100;
purgeWrite      0;
writeFormat     binary;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
adjustTimeStep  true;
maxCo           0.1;
```

Hands-on tutorial

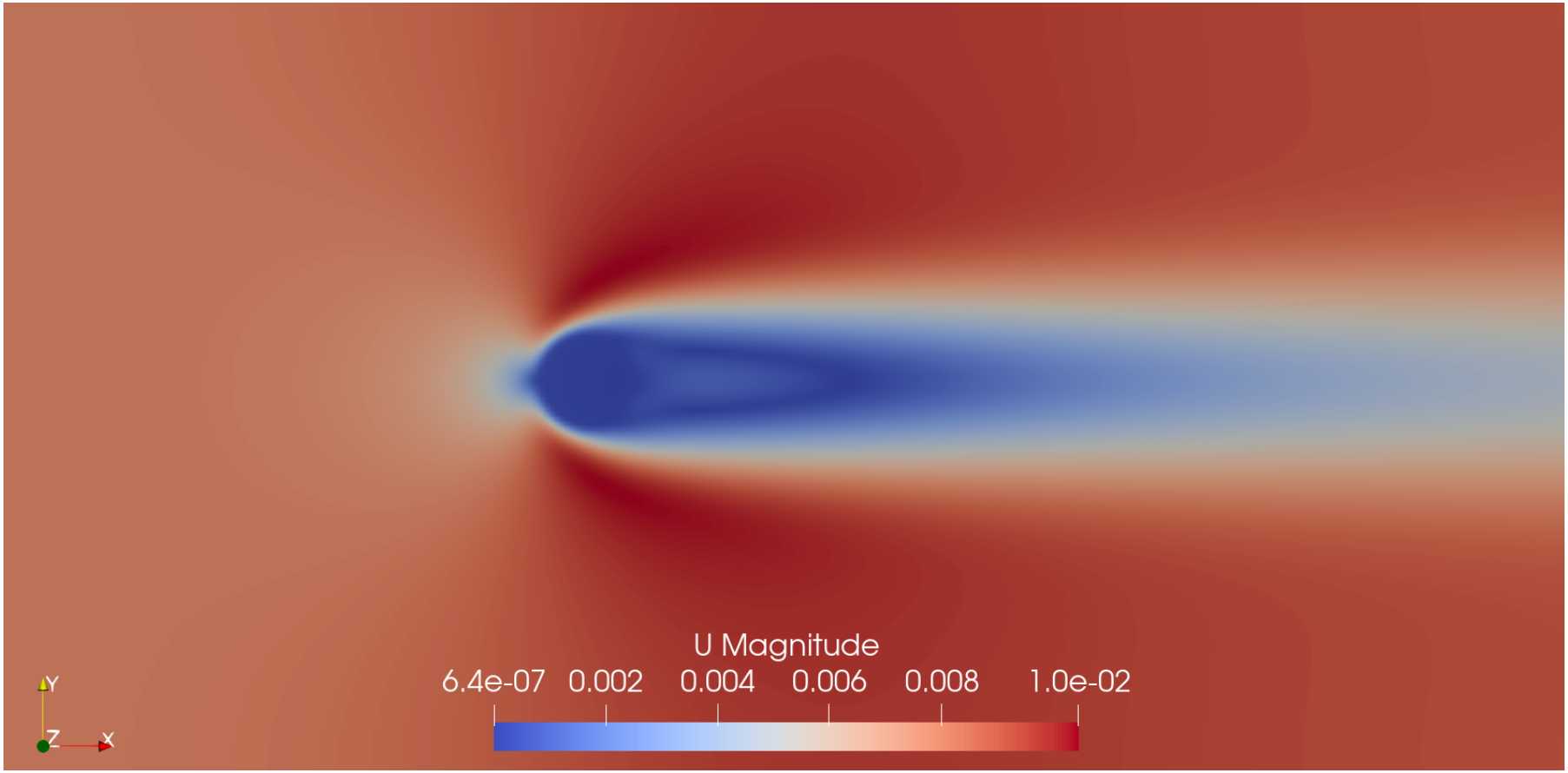
- Now, in a terminal window you can type the following lines. Remember to source the OpenFOAM installation first!

```
$ cd directory_solver
$ wmake
$ cd directory_case
$ cp -r 0.org 0
$ blockMesh > log.preProcessing
$ setFields >> log.preProcessing
$ decomposePar >> log.preProcessing
$ mpirun -np 4 porousPimpleIbFoam -parallel > log.porousPimpleIbFoam
```

- To visualize the results, type paraFoam in the terminal

Hands-on tutorial

- At the end of the simulation, you should obtain something like this:



Thank you for your attention!

Any question?