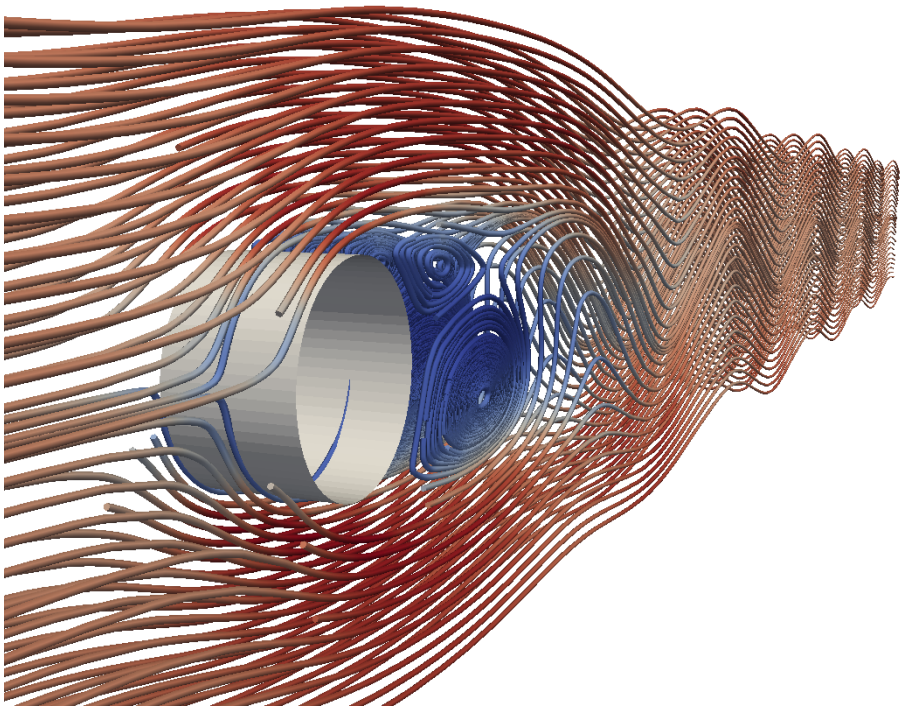
Final Report

Immersed Boundary pisoFoam Solver

Mahmoud Zidan, Mahmoud Ammar, Khaled Boulbrachene*
Group #2

Summer Semester 2019



* *Technische Universität München*

Contents

1 Introduction

To address the computations of solid particles in a fluid flow, one has to solve the transient Navier-Stokes (NS) equations in the fluid domain with the imposition of no-slip boundary condition at the interface between the fluid domain and the solid object. One way to achieve this is by the application of arbitrary Lagrangian-Eulerian method, which combines the advantages of both Lagrangian descripton following an individual parcel as it moves through space and Eulerian description focusing on specific locations in the space. However, this method requires adaptive meshing depending on solid particles displacements as time evolves and therefore leads to a substantial computational cost [1]. To avoid repeated re-meshing, the whole computational domain containing the fluid and the solid is meshed with a fixed structured cartesian grid. As a result, the grid will most probably not conform to the solids boundary. Here, the introduction of the solid phase to the governing flow equations is achieved by adequately formulating the source term marked in the NS equation (1).

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{1}{\rho}\frac{\partial p}{\partial \mathbf{x}} = \nu \frac{\partial^2 \mathbf{u}}{\partial \mathbf{x}^2} + \underbrace{f}_{\text{source term}} \tag{1}$$

Immersed boundary methods were first introduced by Peskin in 1972 [2] and used for flows around flexible membranes, specifically to simulate cardiac mechanics and associated blood flow. In this method, the embedded solid is defined via a number of points distributed over the fluid-solid interface. As already mentioned, as the cartesian finite volume grid is produced without considering the solid boundary, the solid cuts through the mesh and suitable incorporation of boundary conditions in the neighborhood of the boundary is necessary.

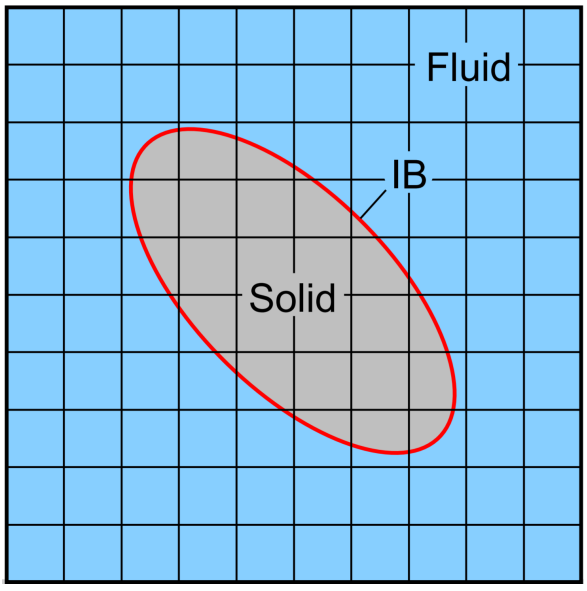


Figure 1: Complex Immersed Boundary in fluid flow

There exist two alternatives to accommodate the effects of the boundary condition on the immersed boundary, namely, the *continuous forcing* approach and the *discrete forcing* approach [3, 4]. In the former one, the forcing term is incorporated into the continuous NS equations through the entire computational domain before discretizing them and it is considered to be attractive for flows with an elastic immersed boundary. On the other hand, the later approach has the forcing applied (explicitly or implicitly) to the discretized NS equations and this was found to be preserving the sharpness of the immersed boundary, delivering best accuracy and

suitable for simulations with high Reynolds number values. The method chosen in this project is the *discrete forcing approach with direct imposition of boundary conditions*, which implies that modifications are directly introduced by imposing the boundary conditions on the cut-cells only, i.e. cells touching the immersed boundary.

Currently, there exist immersed boundary applications in foam-extend, which are *icoIbFoam* and *icoDyMIbFoam* for tansient incompressible laminar flows, *potentialIbFoam* for potential flows, *simpleIbFoam* and *porousSimpleIbFoam* for steady-state turbulent flows. In this project, work has been extended to implement a new Immersed Boundary application, pisoIbFoam, capable of simulating transient turbulent flows.

This report is divided into four main sections. First, the description of boundary condition treatment and turbulence model chosen is presented in 2, followed by the implementation details and mesh refinements in 3. Then, two case studies, namely, flow around a cylinder 4.1 and Pitz Daily 4.2, are solved using the pisoIbFoam solver. These two cases made it possible to study the performance of the solver when the immersed boundary is suspended against a fluid flow and when the immersed boundary is the flow's boundary itself. Results are then discussed in 4 and compared against pisoFoam solver results with a body fitted mesh. Finally, findings were summarized in 5.

2 Theory

2.1 Immersed Bounday

It is worth to mention that it was referred to [5] extensively in this section. As a start, the value of a primary variable (e.g pressure) situated at the center of an immersed boundary cell (cut-cell) is computed by interpolating neighboring cells values and the boundary condition at the immersed boundary point. Figure 2 demonstrates the definitions used.

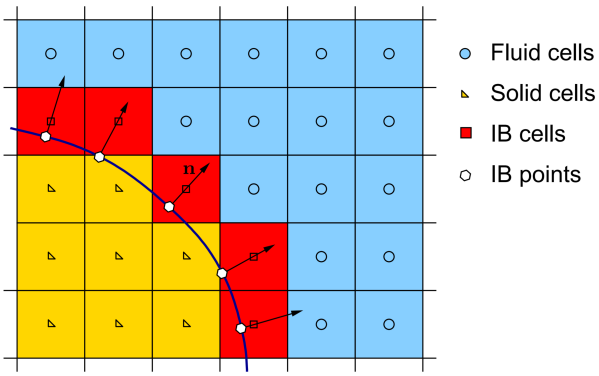


Figure 2: The definition of cells and points in the IB method.

Dirichlet boundary condition interpolation to the Immersed Boundary cell is defined as:

$$\begin{aligned}\phi_P =& \phi_{ib} + C_0(x_P - x_{ib}) + C_1(y_P - y_{ib}) + C_2(x_P - x_{ib})(y_P - y_{ib}) \\ &+ C_3(x_P - x_{ib})^2 + C_4(y_P - y_{ib})^2,\end{aligned} \tag{2}$$

where the coefficients of the quadratic polynomial above are found by the weighted least squares method on the extended stencil depicted below:

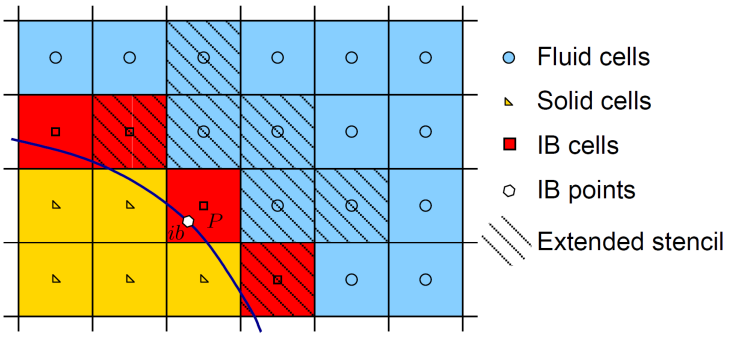


Figure 3: Dirichlet Boundary Condition extended stencil.

On the other hand, Neumann boundary condition is interpolated to the Immersed boundary cell by:

$$\phi_P = C_0 + [\mathbf{n}_{ib} \cdot (\nabla\phi)_{ib}]\, x'_P + C_1 {y'}_P + C_2 x'_P y'_P + C_3 (x'_P)^2 + C_4 (y'_P)^2, \tag{3}$$

where the primed local coordinates are taken such that the x is always perpendicular to the immersed boundary as shown below:

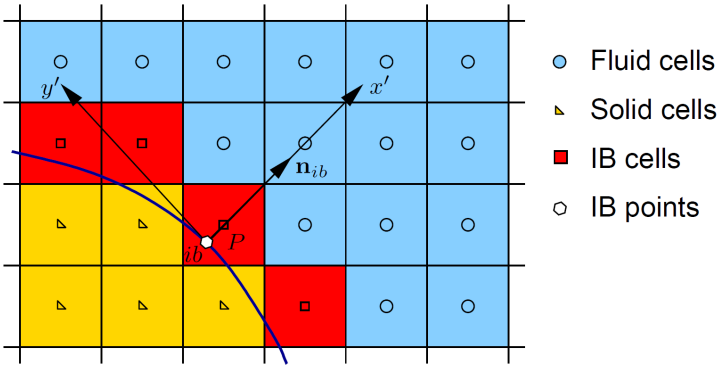


Figure 4: Local coordinate system used for Neumann boundary condition quadratic interpolation

The choice of such local coordinate system reduces the quadratic equations coefficients to be solved for as y'_P always vanish.
The last point to have a closed definition is the pressure equation boundary condition of cells neighboring an Immersed boundary cell. An extra term to the conventional definition is added:

$$\sum_f \left(\frac{1}{a_P}\right) \mathbf{n}_f \cdot (\nabla p)_f S_f = \sum_f \mathbf{n}_f \cdot \mathbf{v}_{f_{ib}} S_f + \sum_{f_{ib}} \mathbf{n}_f \cdot \mathbf{v}_{f_{ib}} S_f, \tag{4}$$

with f_{ib} to be the cell face coinciding to the immersed boundary cage, and $\mathbf{v}_{f_{ib}}$ to be the interpolated velocity at the immersed boundary face.

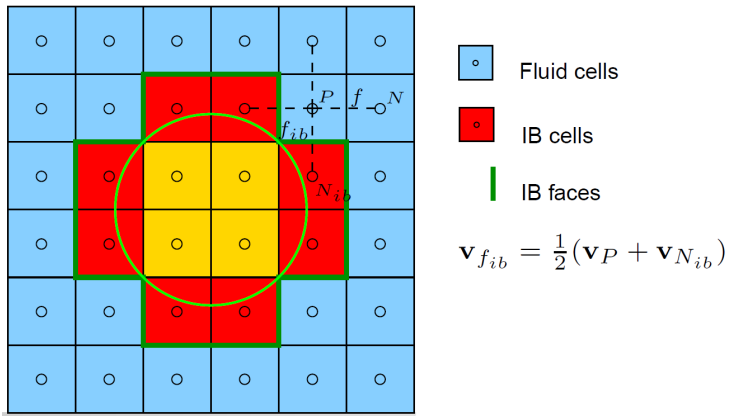


Figure 5: Pressure cell neighbouring to IB cells

The solution of the pressure is said to be independent of the boundary condition since the velocity at the IB face is found as specified. But pressure at IB faces is necessary for momentum equation solution. Therefore, (3) is used to find the pressure solution at the IB cell and eventually at the IB face.

2.2 Turbulence Modeling

In turbulence modelling there are three main models that are used in computational fluid dynamics (CFD), namely, Direct Numerical Simulation (DNS), Reynolds-Averaged Navier-Stokes (RANS), and Large Eddy Simulation (LES). DNS model requires a considerable amount of computational power where all length scales are resolved exactly [6]. Therefore, only RANS and LES models are a reasonable choice to use.

What differentiates RANS from LES is that the equations are derived by taking the time average of the NS equations and defining the velocity to be a summation of average and fluctuation parts. On the other hand, LES method resolves the large scales of the flow and models the small scales by means of a spatial filter. As shown in Table 1, RANS is more suitable for the current cases as it is computationally cheaper and applicable for the 2-dimensional problems [6].

Table 1: Some differences between RANS and LES.

	RANS	LES
Type of solution	Steady and unsteady	Always unsteady
Dimensions	1D, 2D or 3D	Only 3D
Computational effort	Comparably small	Comparably large
Grid required	Relatively coarse	Fine

There are many models in RANS and the focus here will be on the two-equations models. For the cylinder case, the k-ω SST model is used due to the fact that it combines the advantages of both k-ω and k-ϵ models which are capable of predicting the flow in the viscous sub-layer, and away from the wall, respectively. On the other hand, for the pitzDaily case, the k-ϵ model is used because the flow behaviour near the wall will not affect reaching a steady state, therefore, the k-ω model is abandoned here [7].

3 Setup

In this section, the code of the pisoIbFoam is explained in 3.1. Also, two case studies are shown in 3.3. The results are discussed further in section 4.

3.1 Code

As mentioned in the foam-extend wiki, one could create an application that uses the immersed boundary toolkit with certain steps to follow. The steps followed in this work to make a pisoFoam Immersed Boundary application, namely pisoIbFoam, are:

1. The dependacies are added by appending `EXE_INC` in "Make/options" with:

   ```
   -I$(LIB_SRC)/finiteVolume/lnInclude \
   -I$(LIB_SRC)/meshTools/lnInclude \
   -I$(LIB_SRC)/immersedBoundary/immersedBoundary/lnInclude
   ```

   and appending `EXE_LIBS` in "Make/options" with:

   ```
   -lsurfMesh \
   -lsampling \
   -limmersedBoundary \
   -limmersedBoundaryTurbulence
   ```

2. The support of immersed boundary patches is added by adding the following to pisoFoam.C:

   ```
   #include "createIbMasks.H"
   #include "immersedBoundaryFvPatch.H"
   #include "immersedBoundaryAdjustPhi.H"
   ```

3. Before the `adjustPhi` command, `immersedBoundaryAdjustPhi(U, phi)` is added in pisoFoam.C.

4. After each explicit update of the velocity (`U = ...`), the boundary conditions must be corrected by adding `U.correctBoundaryConditions();`.

5. Printing the continuity error in pisoFoam.C is changed to be immersed boundary sensetive by changing `#include "continuityErrs.H"` to `#include "immersedBoundaryContinuityErrs.H"`

The final code for pisoIbFoam.C can be seen in section A.2 in the appendix.

3.2 Immersed Boundary Meshing and Boundary Conditions

As mentioned in 1, the background mesh is a cartesian mesh. The mesh, however, needs to be fine around the immersed boundary to be able to capture it. Therefore, the *refineMesh* OpenFOAM application was used to refine the mesh around the boundary, where a *cellSet* with a *surfaceToCell* option given the STL immersed bonundary. The mesh is shown in Figures 8 and 10b.
The method followed to create an *immersedBoundary* patch is to create a boundary in "blockMeshDict" with no faces, and to manually change the `faces ();` in "constant/polyMesh/boundary" to `internalFlow no;` or `internalFlow yes;`, after all the refinements are finished. For example, the immersed boundary patch is defined in the file "boundary" as:

```
Cylinder
{
    type            immersedBoundary;
    nFaces          0;
    startFace       29820;
    internalFlow    no;
}
```

The boundary condition on the immersed boundary patch can be applied given a type, reference value, reference gradient, *fixesValue* tag, and a value. For example, a velocity wall boundary condition can be applied using:

```
Cylinder
{
    type                immersedBoundaryWallFunction;
    patchType           immersedBoundary;
    refValue            uniform (0 0 0);
    refGradient         uniform (0 0 0);
    fixesValue          yes;
    setDeadCellValue    yes;
    deadCellValue       (0 0 0);
    value               uniform (0 0 0);
}
```

and a pressure zero gradient boundary condition would be applied using:

```
Cylinder
{
    type                immersedBoundary;
    refValue            uniform 0;
    refGradient         uniform 0;
    fixesValue          no;
    setDeadCellValue    yes;
    deadCellValue       0;
    value               uniform 0;
}
```

Notice the `fixesValue` tag set to "yes" in case of a *fixedValue* boundary condition, and "no" in case of the *zeroGradient* boundary condition.

3.3 Case Studies

Two cases are studied with the new application, pisoIbFoam. The first is a flow around a cylinder and the second is the Pitz Daily case [8]. A body-fitted mesh as well as an immersed boundary simulations are applied.

3.3.1 Flow Around a Cylinder

The geometry as well as the boundaries of the case are shown in Figure 6, where the diameter of the cylinder is 2.

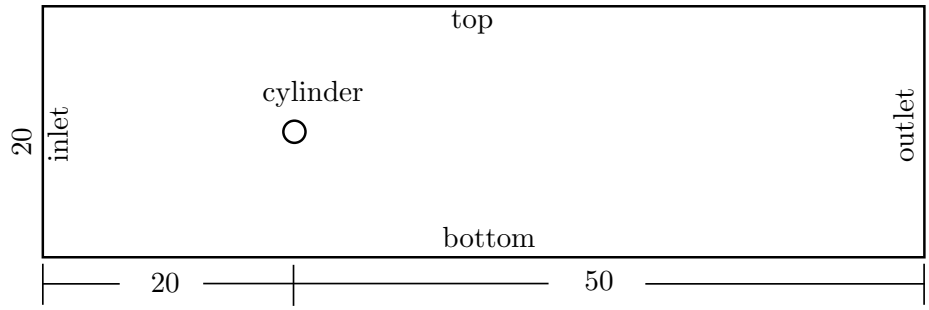


Figure 6: Boundaries of the flow around a cylinder

The boundary conditions applied on this case can be depicted in Table 2, where the k, ϵ and ω values are estimated as follows:

$$\begin{aligned} I &= 0.16 \cdot \mathrm{Re}^{-\frac{1}{8}}, \\ k &= \frac{3}{2} \cdot (U_{\mathrm{ref}} \cdot I)^2, \\ \varepsilon &= \frac{0.164 \cdot k^{1.5}}{0.07 \cdot L}, \\ \omega &= \frac{\varepsilon}{k}, \end{aligned} \tag{5}$$

where I is the turbulence intensity, Re is Reynold's number, U_{ref} is the inlet velocity (0.130626) and L is the diameter of the cylinder. The dynamic viscosity of the fluid is ν =1.7894 $\times$ 10^{-6}.

Table 2: Boundary conditions of the body-fitted flow around a cylinder.

Boundary	inlet / outlet / top / bottom	Cylinder	front / back
U	freestream (0.130626 0 0)	fixedValue (0 0 0)	empty
p	freestreamPressure uniform 0	zeroGradient	empty
nut	calculated uniform 0	nutWallFunction uniform 0	empty
k	freestream uniform 3.352×10^{-5}	wallFunction uniform 3.352×10^{-5}	empty
ε	freestream uniform 2.2734×10^{-7}	wallFunction uniform 2.2734×10^{-7}	empty
ω	freestream uniform 0.006782	wallFunction uniform 0.006782	empty

The mesh in case of the body-fitted simulation is divided into regions, where the the mesh is finer around the cylinder leading to a maximum y+ of 31, and in the wake region where the vortices are shed. The mesh is depicted in 7.

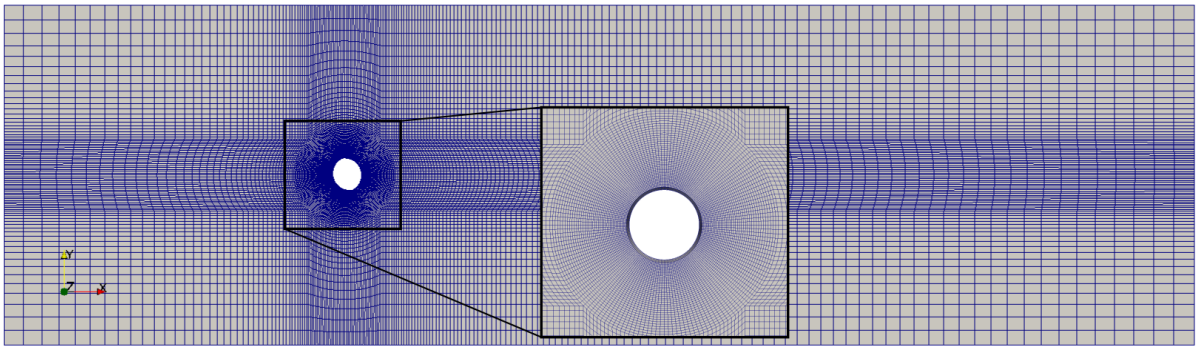
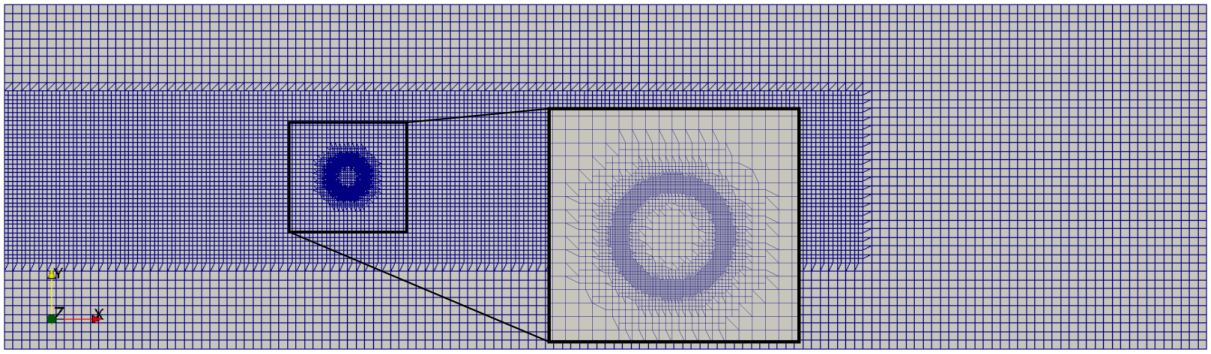
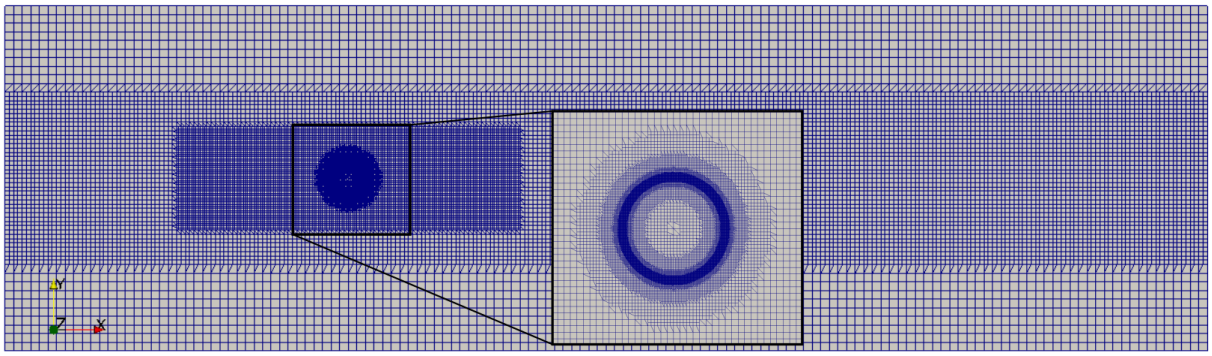


Figure 7: Body-fitted mesh of flow around a cylinder.

In case of the immersed boundary simulation, the background is meshed, and steps of cummulative refinements are performed, as mentioned briefly in 3.2. First, the wake region is refined using a box *cellSet*. Then, three refinements using the *surfaceToCell* are performed. Two meshes are studied using one and two box sets, as shown in Figures 8a and 8b, respectively.



(a) Coarser mesh.



(b) Finer mesh.

Figure 8: Immersed boundary mesh of flow around a cylinder.

3.3.2 Pitz Daily

The geometry as well as the boundaries of the case are shown in Figure 9. The boundary conditions applied are summarized in Table 3.

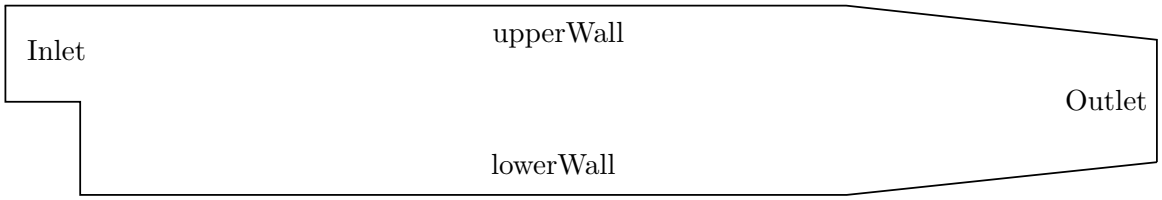
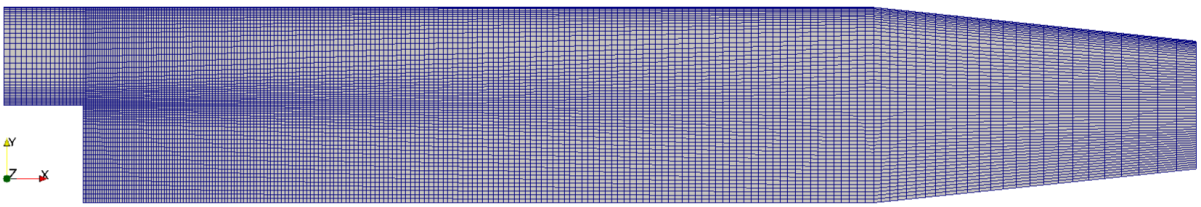


Figure 9: Boundaries of the pitzDaily case

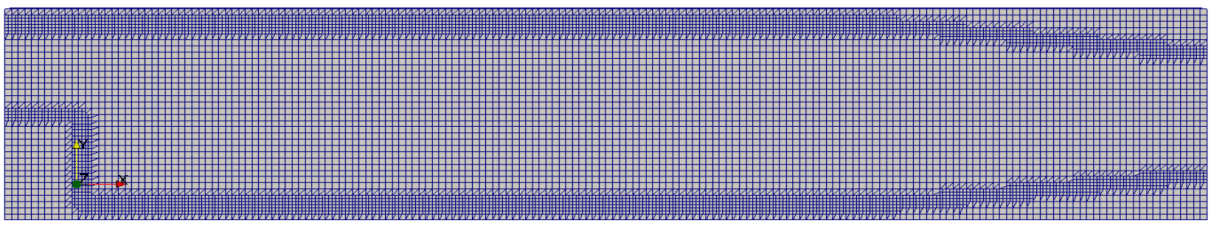
Table 3: Boundary conditions of the body-fitted flow around a cylinder.

Boundary	inlet	outlet	top / bottom	frontAndBack
U	fixedValue (10 0 0)	zeroGradient	fixedValue (0 0 0)	empty
p	zeroGradient	fixedValue uniform 0	zeroGradient	empty
nut	calculated uniform 0	calculated uniform 0	nutWallFunction uniform 0	empty
k	fixedValue uniform 0.375	zeroGradient	kqRWallFunction uniform 0.375	empty
ε	fixedValue uniform 14.855	zeroGradient	epsilonWallFunction uniform 14.855	empty

The body-fitted mesh shown in 10a is finer around the walls to capture the high velocity gradient due to the no-slip boundary condition. The immersed boundary mesh shown in 10b is done by a background mesh and a refinement of a *surfaceToCell* region of the STL.



(a) Body-fitted



(b) Immersed boundary

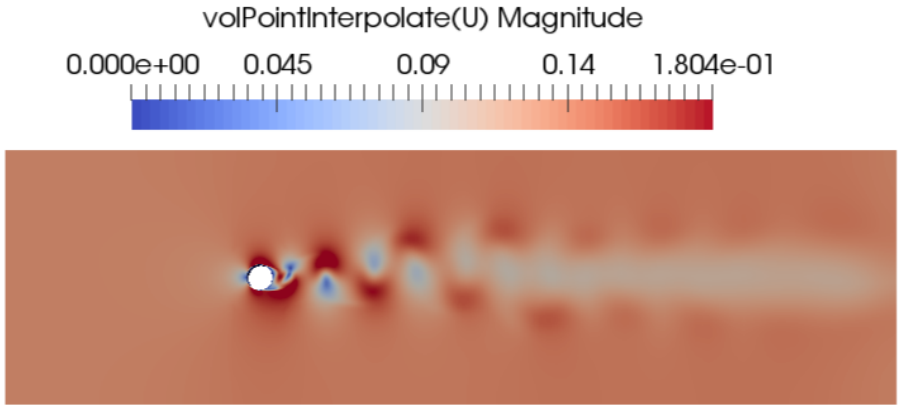
Figure 10: Mesh of the Pitz Daily case.

4 Results

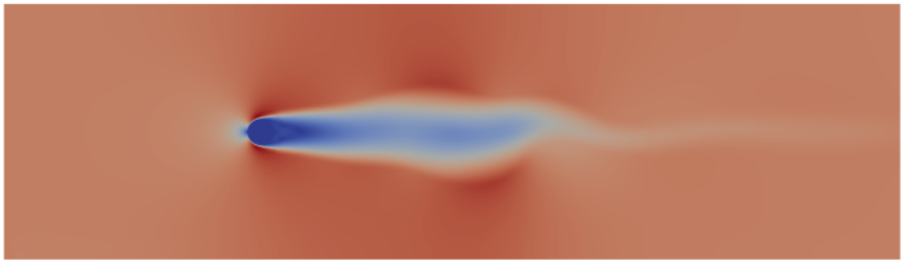
In this section, the results of the cases introduced in 3.3.1 and 3.3.2 are shown and discussed.

4.1 Flow Around a Cylinder

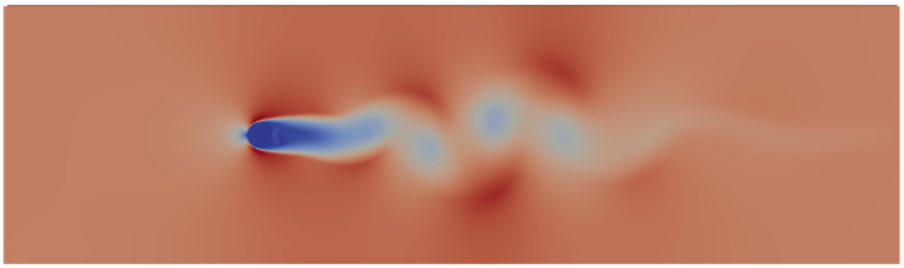
Due to the aforementioned reasons in 2.2, the turbulence model chosen for this case is K-ω-SST. The simulation's end time is 1500s. In the body-fitted mesh, a Δt of 0.25s is stable. In the immersed boundary coarser mesh (Figure 8a), Δt of 0.1s is stable for the whole simulation. In the finer immersed boundary mesh (Figure 8b), Δt of 0.05s is stable until $t = 560$s. The result differs with the refinement around the immersed boundary. The finer mesh is computationally expensive, and because the immersed boundary tool in foam-extend does not work in parallel based on our trials, the mesh needs a $\Delta t < 0.05$s, which would take more than 24 hours to simulate. At $t = 550$s, the velocity field is shown in Figure 11. As shown, the body-fitted and the fine immersed have already started vortex shedding, but that is not the case in the coarser immersed mesh.



(a) Body-fitted mesh.



(b) Coarser immersed mesh.

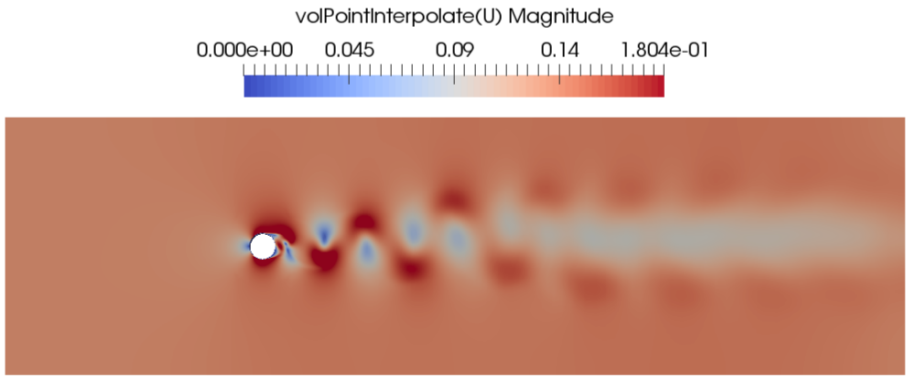


(c) Finer immersed mesh.

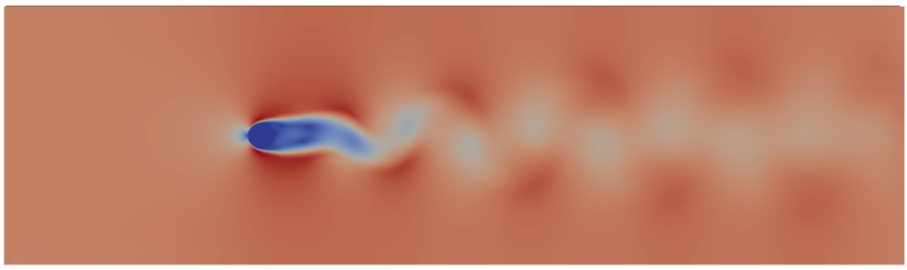
Figure 11: Velocity magnitude at $t = 550$s

The velocity field at $t = 1500$s is shown in Figure 12. In both the body-fitted and the immersed cases, the vortices are already shedding periodically. The velocity field is different due to the error in the immersed boundary simulation. The errors may be due to the insufficient mesh, since the finer mesh shows a different result as shown in Figure 11, however, computationally

infeasible.
In Figure 13, the pressure forces are shown. The drag forces (i.e f_x) are of a similar shape and a close magnitude and the twist force (i.e f_z) is nearly zero in both cases. However, the lift force (i.e f_y) is of different magnitude and frequency. The frequency of the lift forces can be calculated using a Fast Fourier Transform (FFT) and is 0.0041 Hz in the body-fitted case after $t = 400$s and 0.000294 Hz in the coarser immersed boundary mesh after $t = 1200$s. The amplitude is also significantly different.



(a) Body-fitted mesh.



(b) Coarser immersed mesh.

Figure 12: Velocity magnitude at $t = 1500$s

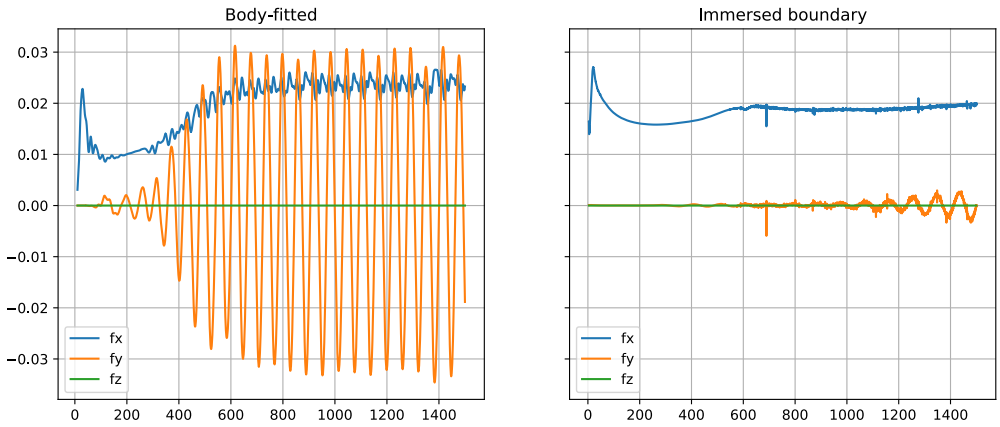
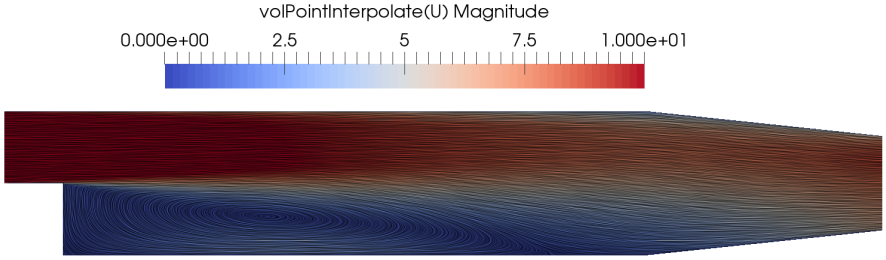


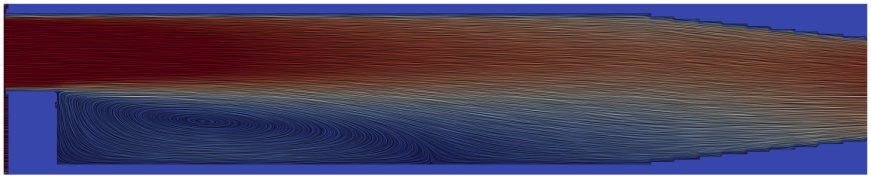
Figure 13: Pressure forces on the cylinder

4.2 Pitz Daily

Due to the fact that an extensive computational effort is needed and parallelizing in immersed boundary method solver in OpenFOAM is not possible, the cylinder case could not be validated. The Pitz Daily benchmark case was chosen because a steady state solution can be reached to compare the results between pisoFoam and pisoIBFoam.



(a) Body-fitted mesh.



(b) Immersed mesh.

Figure 14: Velocity magnitude at $t = 0.15$s

In Figure 14, the results look similar when it reached a steady-state with similar velocity magnitude. A slight difference can be observed in Figure 14a has a slightly smaller and wider recirculation region than Figure 14b.

5 Conclusion

In this work, a new application to solve transient turbulent flows with immersed boundary support, pisoIbFoam, was developed and tested with two cases, a transient vortex shedding phenomenon of a flow around a cylinder and the flow reaching steady-state Pitz Daily benchmark case. The application is advantageous due to its simplicity in meshing, however, could not be fully validated due to the extended computational effort as the mesh has to be much finer around the immersed boundary.

References

[1] Markus Uhlmann.
"An Immersed Boundary Method with Direct Forcing for the Simulation of Particulate Flows".
In: *J. Comput. Phys., 209(2):448-476, 2005)* (Nov. 2005).
URL: https://www.sciencedirect.com/science/article/pii/S0021999105001385.

[2] Charles S Peskin.
"Flow patterns around heart valves: A numerical method".
In: *Journal of Computational Physics* 10.2 (1972), pp. 252 –271.
ISSN: 0021-9991.
DOI: https://doi.org/10.1016/0021-9991(72)90065-4.
URL: http://www.sciencedirect.com/science/article/pii/0021999172900654.

[3] Science Direct Topics.
"Immersed Boundary Method - an overview".
In: ().
URL: https://www.sciencedirect.com/topics/engineering/immersed-boundary-method.

[4] Jiyuan Tu, Guan-Heng Yeoh, and Chaoqun Liu.
"Chapter 9 - Some Advanced Topics in CFD".
In: *Computational Fluid Dynamics (Third Edition)*.
Ed. by Jiyuan Tu, Guan-Heng Yeoh, and Chaoqun Liu.
Third Edition.
Butterworth-Heinemann, 2018,
Pp. 369 –417.
ISBN: 978-0-08-101127-0.
DOI: https://doi.org/10.1016/B978-0-08-101127-0.00009-X.
URL: http://www.sciencedirect.com/science/article/pii/B978008101127000009X.

[5] H. Jasak and Z. Tukovic.
"Immersed boundary method in FOAM, theory, implementation, and use".
In: (2015).
URL: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2015/HrvojeJasak/ImmersedBoundary.pdf.

[6] J H. Ferziger and Milovan Perić.
Computational Methods for Fluid Dynamics.
3rd.
Springer, 2002.

[7] F. R. Menter.
"Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications".
In: *AIAA Journal* 32.8 (1994), pp. 1598 –1605.
DOI: https://doi.org/10.2514/3.12149.
URL: https://arc.aiaa.org/doi/abs/10.2514/3.12149.

[8] R. PITZ and J. DAILY.
"Experimental study of combustion in a turbulent free shear layer formed at a rearward facing step".
In:
19th Aerospace Sciences Meeting.
DOI: 10.2514/6.1981-106.
URL: https://arc.aiaa.org/doi/abs/10.2514/6.1981-106.

A Appendix

A.1 pisoIbFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | foam-extend: Open Source CFD
   \\    /   O peration     | Version:     4.0
    \\  /    A nd           | Web:         http://www.foam-extend.org
     \\/     M anipulation  | For copyright notice see file Copyright
-------------------------------------------------------------------------------
License
    This file is part of foam-extend.

    foam-extend is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation, either version 3 of the License, or (at your
    option) any later version.

    foam-extend is distributed in the hope that it will be useful, but
    WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with foam-extend.  If not, see <http://www.gnu.org/licenses/>.

Application
    pisoIbFoam

Description
    Transient solver for incompressible flow with immersed boundary.

    Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
#include "pisoControl.H"

#include "immersedBoundaryFvPatch.H"
#include "immersedBoundaryAdjustPhi.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"

    pisoControl piso(mesh);

#   include "createIbMasks.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
```

```
59
60      while (runTime.loop())
61      {
62          Info<< "Time = " << runTime.timeName() << nl << endl;
63
64  #       include "immersedBoundaryCourantNo.H"
65
66          // Pressure-velocity PISO corrector
67          {
68              // Momentum predictor
69
70              fvVectorMatrix UEqn
71              (
72                  fvm::ddt(U)
73                + fvm::div(phi, U)
74                + turbulence->divDevReff()
75              );
76
77              UEqn.relax();
78
79              if (piso.momentumPredictor())
80              {
81                  solve(UEqn == -fvc::grad(p));
82              }
83
84              // --- PISO loop
85
86              while (piso.correct())
87              {
88                  volScalarField rUA = 1.0/UEqn.A();
89
90                  U = rUA*UEqn.H();
91                  U.correctBoundaryConditions();
92                  phi = (fvc::interpolate(U) & mesh.Sf()) + fvc::ddtPhiCorr(rUA, U
      , phi);
93
94                  immersedBoundaryAdjustPhi(phi, U);
95                  adjustPhi(phi, U, p);
96
97                  // Non-orthogonal pressure corrector loop
98                  while (piso.correctNonOrthogonal())
99                  {
100                     // Pressure corrector
101
102                     fvScalarMatrix pEqn
103                     (
104                         fvm::laplacian(rUA, p) == fvc::div(phi)
105                     );
106
107                     pEqn.setReference(pRefCell, pRefValue);
108                     pEqn.solve
109                     (
110                         mesh.solutionDict().solver
111                         (
112                             p.select(piso.finalInnerIter())
113                         )
114                     );
115
116                     if (piso.finalNonOrthogonalIter())
117                     {
118                         phi -= pEqn.flux();
119                     }
```

```
120                 }
121
122 #               include "immersedBoundaryContinuityErrs.H"
123
124                 U -= rUA*fvc::grad(p);
125                 U.correctBoundaryConditions();
126             }
127         }
128
129         turbulence->correct();
130
131         runTime.write();
132
133         Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
134             << "  ClockTime = " << runTime.elapsedClockTime() << " s"
135             << nl << endl;
136     }
137
138     Info<< "End\n" << endl;
139
140     return 0;
141 }
142
143
144 // ************************************************************************* //
```

A.2 Allrun

```
#!/bin/sh
# Source tutorial run functions
. \$WM_PROJECT_DIR/bin/tools/RunFunctions

# Get application name
application="pisoIbFoam"

refineMeshByCellSet()
{
    echo "creating cell set for primary zone - $1"
    cp system/cellSetDict.$1 system/cellSetDict
    cellSet > log.cellSet.$1 2>&1
    echo "refining primary zone - $1"
    refineMesh -dict -overwrite > log.refineMesh.$1 2>&1
}

runApplication blockMesh -dict system/blockMeshDict
refineMeshByCellSet 0
refineMeshByCellSet 1
refineMeshByCellSet 2
refineMeshByCellSet 3

# Define the immersed boundary patch (section 3.2)
sed -i "s/faces           ( );/internalFlow no;/g" constant/polyMesh/boundary

# Renumbers the cell list in order to reduce the bandwidth
runApplication renumberMesh -overwrite

cp 0/polyMesh/* constant/polyMesh/
rm -rf 0
mkdir 0
cp -f 0_org/* 0/
runApplication $application
```