Integration of CFD Modelling Framework IC-FERST for Industrial Application in BP: Automation of pre- and post-processing using Python

Dongzhen Li[a], Asiri Obeysekara[b], Andre Nicolle[c], Chris Pain[b]

Module Name: ACSE9 - Independent Research Project
Github alias: anitali555888
Email: dongzhen.li18@imperial.ac.uk

[a]*Applied Computational Science and Engineering (ACSE), Imperial College London*
[b]*AMCG, Imperial College London*
[c]*BP*

Abstract

From the engineeing motivation to a review of flow past a cylinder, and finally to the development of a practical post-processing tool for cases relevant to the classic ideal case flow past a cylinder for integration purposes with IC-FERST, this project introduce the implementation details and the physical background. Some generalisation implementation from flow past cylinder is explained in this report, and the future modification possibility is discussed as well. IC-FERST is a CFD solver based on Fluidity for solving Navier-Stokes equations in 2 and 3 dimensions developed in AMCG Imerial College London. It is a general-purpose code for simulating multiphase flow and transport in complex geological reservoirs. This project is related to a research project between Imperial College and BP to evaluate IC-FERST.

Keywords: flow past a cylinder, CFD, turbulent flow, data visualization, simulation diagnosing, IC-FERST

1. Introduction

1.1. Motivation and Background

1.1.1. Turbulent Flow Around a Body

Computational Fluid Dynamics (CFD), which refers to use numerical analysis with the computer power to analyze and solve problems that involve fluid flows, is widely utilized in academic research as well as the study of fluids[1]. When it comes to engineering problems, CFD with its excellent efficiency and the low cost of physical systems characters becomes the popular choice of engineers[2][3].

As the development of computer technologies, simulations to evaluate the effects of operating parameters on the system performance with CFD was widely utilized[3]. One example is focusing on the flow around bluff bodies[1]. A bluff body has separated flow over a substantial part of its surface, and kept some part of its boundary out of touch with the fluid in a fluid flow[4]. The blowout preventer (B.O.P) underwater is a extended case of applying CFD to study impact of turbulent flow around structures.

A blowout preventer (B.O.P) is a large, specialized valve or similar mechanical device, used to seal, control and monitor oil and gas wells to prevent blowouts, the uncontrolled release of crude oil and/or natural gas from a well[5].

Because B.O.P is important for the safety of the crew and natural environment, and it is required to remain submerged for a long time in extreme conditions in deep-water, increasing safe operating capacity of them is a key focus in the technological development[6]. Extreme conditions, for examples, the erosion due to the sea water, the vibration due to the water flow and the possibility of low-quality structural integrity of B.O.P. Therefore, studies about the impact of external environmental conditions on the sub-sea structures like B.O.P become of significant importance for mitigation of risk and costs during design, operation and repair of such engineering applications.

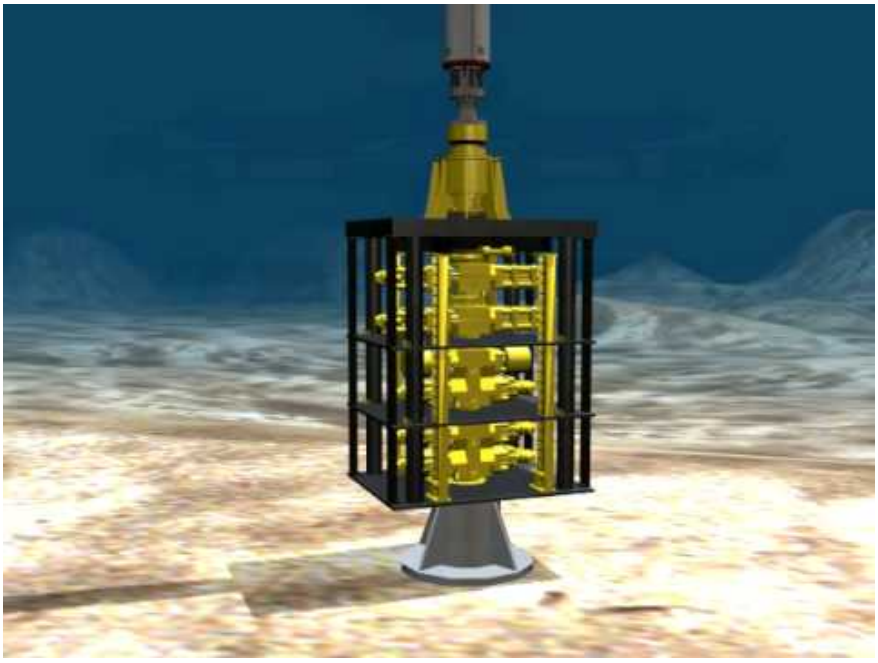

Figure 1: A B.O.P in deep-water.

The B.O.P in Figure 1 can be approximated as a bluff body and the water body flows around that can be seen as a turbulent flow, so that it is important to know the actual conditions like the pressure load level and the flow patterns around the body to keep the component in safe and efficient operational conditions.

In real industrial problems, the circumstances, such as flow conditions, the geometry of the body, and structural mechanics, are often complex. Extensive experimental [7] and theoretical/numerical research[8][9] have been carried out to better understand more realistic interactions present in real world scenarios. However, during the last century, the study of steady and unsteady flow around an idealized circular cylinder remains as a challenging and widely studied problem[7][10].

1.1.2. Flow past a cylinder

Flow past a cylinder (FPC) is a classic case, because it is seen as a simplified benchmark problem for many industry cases. According to the research of B. N. Rajani[11] such a case can validate the modelling code and then extended and modified to a more general case. Namely flow

past a circular cylinder is a good start to this project. It is a representative of the unsteady three-dimensional flow containing separation, reattachment, free shear layer instabilities so that it is an appropriate example for three dimensional disturbances and transition from laminar to turbulent state of flow in the wake of the cylinder. Along with the varying Reynolds number (Re), flow past a circular cylinder reveals distinct flow patterns into different flow regimes[11].

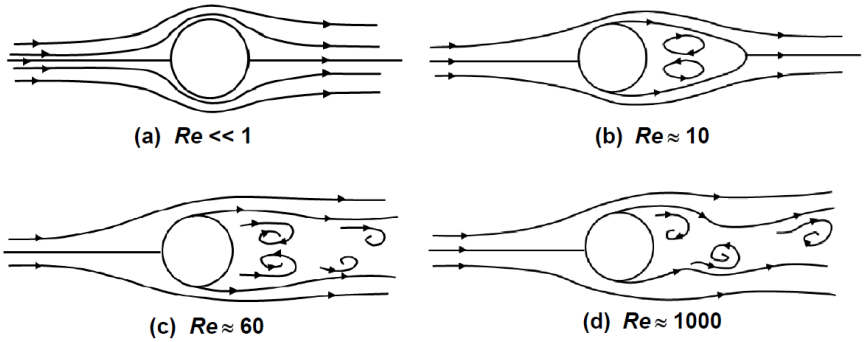

Figure 2: Flow pattern generated around a circular cylinder.[9]

As the Figure 2 shown and conclusions from Beaudan 1994 [8], the flow at small Reynolds number between around 5 to 47 stays at a steady and laminar status. When it is getting closer to 47 and continuing up to a value of Re of about 190, there will be indication that alternate vortices shedding from the cylinder surface. In this stage, the vortices perform as quite regular, which is known as the von Karmen vortex street[12]. If the Re keeps getting larger, the instability appearing in the two-dimensional flow will lead to disturbance in the far wake region in a three-dimensional way[11].

The dimensionless quantity Re is subjected to relative internal movement due to different fluid velocities, and has the expression:

$$Re = \frac{\rho u L}{\mu} = \frac{u L}{\nu} \tag{1}$$

Where, ρ is the density of the fluid (SI units: kg/m3), u is the velocity of the fluid with respect to the object (m/s), L is a characteristic linear dimension (m), μ is the dynamic viscosity of the fluid (Pa·s or N·s/m2 or kg/m·s), ν is the kinematic viscosity of the fluid (m2/s).

According to many research work, Re = 3900 is a lower sub-critical value for the flow, which will remain laminar properties but have transition and separation characteristics in the free shear layer of the wake and turbulent eddies shed periodically along the wake[13][11][14][15][7].

1.1.3. Important Quantities in Physical Space

The flow around a circular cylinder at a sub-critical Reynolds number, which has become a benchmark for more application about complex flows in real technical issues[16]. This benchmark is with simple geometry but can have complicated flow phenomena, including laminar separation with no fixed separation point, transition to turbulence in the thin shear layers, that are separating, and shedding of large-scale vortices[17][18] (as Figure 2 shown).

Often, when studying turbulent flow around, engineers are interested in:

(1) the pressure distribution and drag coefficient[19] on the bodies, which is required in the design process[18];

(2) the velocities around the bodies, which is required for the assessment of stability and the structure performance[11].

From the research results of Franke 2002[18], Rajani 2016[11] etc. and which they have mentioned, the analysis about the simulation model of flow past a circular cylinder can be visualized as follows:

(1) the drag coefficient in pressure expression[19]: Surface pressure coefficient distribution by computing the surface pressure coefficient along with the angle θ.

$$Cp = \frac{p - p0}{\frac{1}{2}\rho U^2} \tag{2}$$

Where Cp is Pressure coefficient, p is the surface pressure, p0 is free stream static pressure, ρ is air density and U is free stream air velocity.

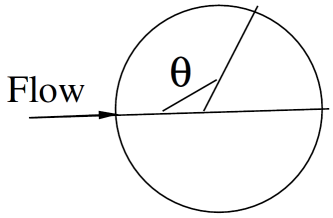

Figure 3: Schematic diagram of the angle θ.[11]

(2) the velocities around the bodies: Mean streamwise velocity along wake centreline: $U_{CL} - x/D$

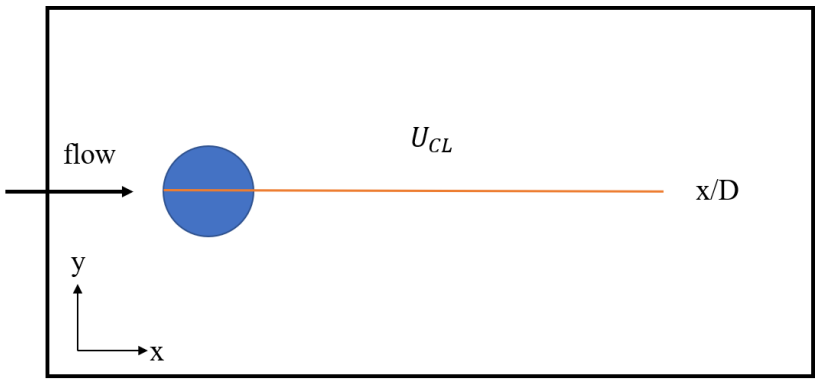

Figure 4: Schematic diagram of $U_{CL} - x/D$.

In both the near wake region and the farther wake region, for comparison reasons, it is sensible to choose three different distances x/D at each, like 1.06, 1.54, 2.02, and 4, 7, 10 from B. N. Rajani et al. 2016[11].

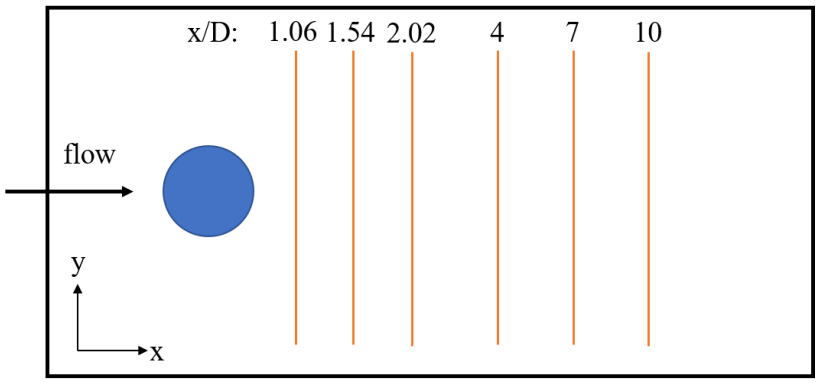

Figure 5: Schematic diagram of the choices in near and farther wake regions.

At those 6 positions, there are several quantities can be the output:
1) Transverse profiles of mean streamwise velocity: $\bar{u}/U - y/D$
2) Transverse profiles of mean cross-stream velocity:$\bar{v}/U - y/D$

The quantities above are the proper objectives for data analysis after the CFD simulation. With the bench-marking purpose of the case FPC, the selected fundamental equations in CFD can be validated by experimental data to enhance the results reliability[3].

1.2. Problem Statement

The Imperial College Finite Element Reservoir Simulator, known as IC-FERST (`http://multifluids.github.io/`), based on **Fluidity** (`http://fluidityproject.github.io/`) that is able to solve the Darcy and the Navier-Stokes equations in 2 and 3 dimensions, is a general-purpose code for simulating multiphase flow and transport in complex geological reservoirs. IC-FERST is based on Control-Volume Finite Element methods and dynamic unstructured mesh optimization, which is a key feature of it compared to fixed grid methods which used by the most other simulators[20][21].

The proposed work in this project is related to a research project between Imperial College and BP to evaluate the advanced CFD codes IC-FERST developed in Imperial College by Applied Modelling and Computation Group (AMCG). IC-FERST uses dynamic unstructured mesh optimisation and has been parallelized using MPI. It is well-suited to utilize the powerful computational resources like high performance computing (HPC).

For further development of the commercial and industrial appliaction of IC-FERST, it is considerable to integrate the simulation with pre- and post-processing sections. This project is aimed to implement a proper integration of simulation results from IC-FERST and processing functions to help users having a better experience in engineering data analyzing.

In this independent research project, the main objective is to develop a practical post-processing tool for cases relevant to flow past a cylinder, while some other projects will have some solutions about the pre-processing tool which compose the final integration goal with the results here.

2. Technical Methodology

2.1. Technical Back-end

2.1.1. Python

Post-processing, the analysis of the simulation results, is traditionally visualized using ParaView, which is an open-source multiple-platform application for interactive, scientific visualization and can be downloaded from `https://www.paraview.org/`.

ParaView has many useful functions to display the data for better analysis[22]. Those functions in ParaView are called "filters", some relevant features are shown in Figure:

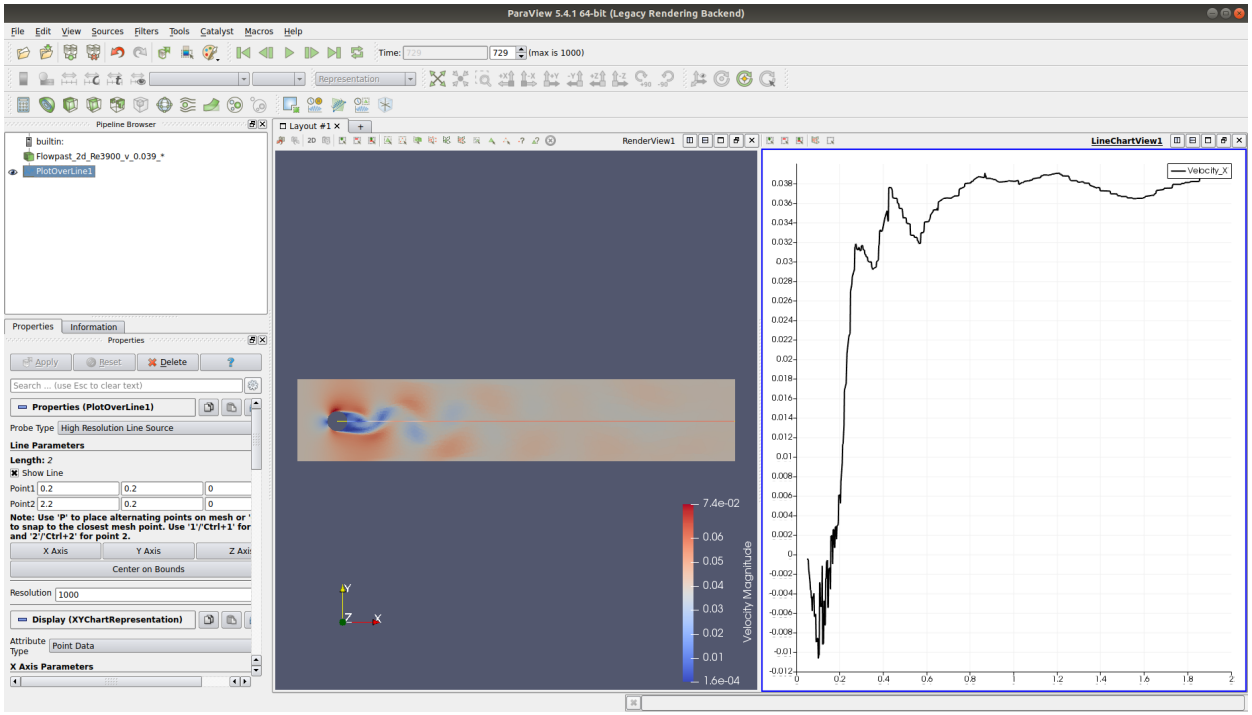

Figure 6: ParaView interface and plot over line filter applied.

As a powerful interactive visualization platform, ParaView is based on such a user interface. For advanced technical environment, like HPC context, it is better to find another way to achieve similar functions without manually operations through the interface, and that is strongly refers to an integrated scripts.

Due to the CFD code IC-FERST is based on **Python**, the development of the processing functions of this project keeps in the same style of **Python**.

2.1.2. VTK library

The Visualization Toolkit (VTK) (`https://vtk.org/`) is open source software for manipulating and displaying scientific data. The VTK library analyses the output data in *.vtu/*.pvtu files such as scalar fields, vector fields and nodal positions, into NumPy arrays using **Python**.

Some examples of **Python** script using the vtk library can be found here: `https://www.vtk.org/doc/nightly/html/pages.html`, and also `https://lorensen.github.io/VTKExamples/site/Python/`.

2.2. Existing Code Ecosystem

As mentioned in section 1.2, this project is for further development of IC-FERST. IC-FERST is developed and run on Linux, and the installation guidelines can be found in `http://multifluids.github.io/license/`.

To run a general simulation case, the work flow is to creating proper meshes from the geometry, and to output results through the CFD modelling tool.

A *.geo file is a text/plain file containing geometry information of the model. The software used to generate the geometry (*.geo) and the mesh (*.msh) is **GMSH**. **GMSH** is a mesh generator freely available at `http://geuz.org/gmsh/`. Both the geometry and the mesh can be created by **GMSH** using the graphical interface. The geometry file also can be written by hand through a general text editor with more flexibility.

Command lines of generating *.msh from *.geo by **GMSH**:

Serial:

```
gmsh -2/3 <name>.geo
```

Parallel:

```
gmsh -2/3 -bin <name>.geo
/path to fldecomp/fldecomp -n {nprocs} <meshname>
```

In order to run a simulation in parallel, the mesh first needs to be decomposed using **fldecomp**. Here, the line `gmsh -2/3 -bin <name>.geo` is for preparing meshes in different file-type which can be used in **fldecomp**.

The *.mpml file is the one for run the simulation model. A number of options for the simulation are available to be set up in the graphical interface **Diamond**.

Command line of opening **Diamond** GUI:

```
diamond -s /source_folder/legacy_reservoir_prototype/schemas
           /multiphase.rng <name>.mpml
```

Through the GUI, it is easy to modify the simulation conditions. In Figure X, it shows contents of a simulation model in **Diamond** GUI.

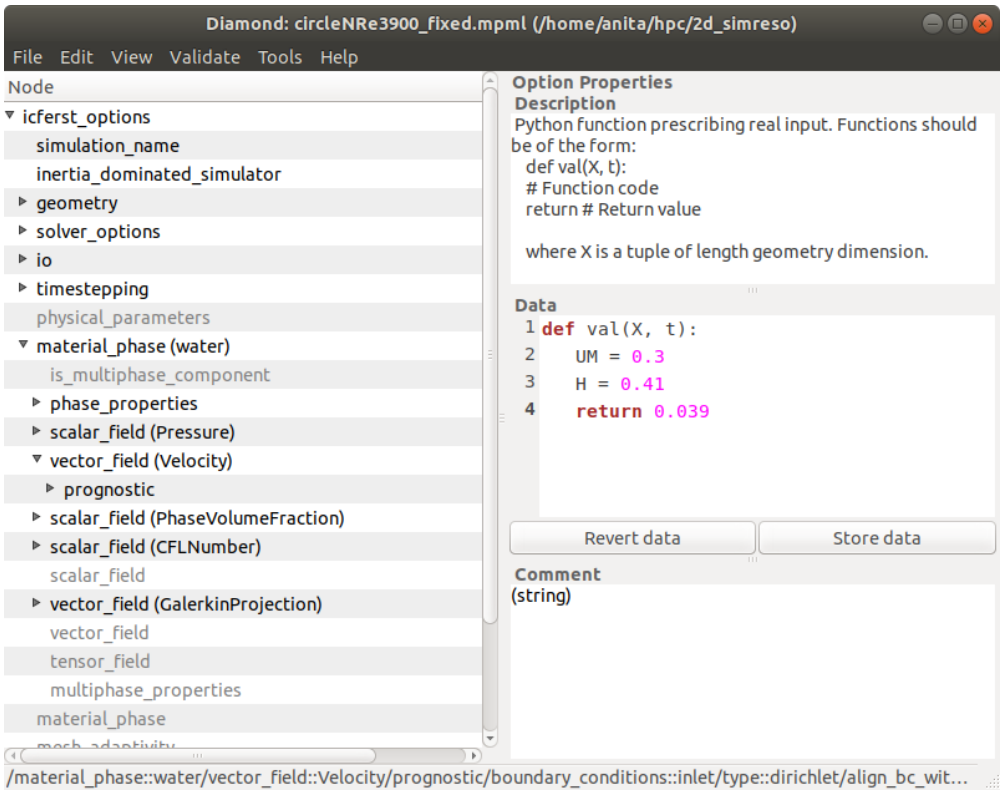

Figure 7: Contents of a simulation model in **Diamond** GUI.

Command lines of running the *.mpml file to start simulation:

Serial: `/source_folder/bin/icferst <name>.mpml`

Parallel: `mpirun -n {nprocs} /path/to/icferst <name>.mpml`

Simulation results are stored in *.vtu/*.pvtu files. In addition, the *.stat file in the outputs records a number of information of the simulation in detail, such as time-step size, simulation time and real elapsed time of each time-step.

2.3. Software Development Life Cycle (SDLC)

The conventional software development methodology Waterfall, which is introduced by Dr.Winston W.Royce in 1970[23], is what this project used. In the Waterfall approach, the whole process of software development is divided into separate sequential stages.

In this project, the main six stages are:

1) Requirements Gathering: Capture all possible information about the project and get familiar with the background knowledge. *Section 1* and *2.2* are for this stage.

2) Analysis: Plan the work flow of this project and document it on the understanding of the requirements and the demand of users. This stage can be extended to *Section 2.1*.

3) Design: Prepare the design rationale and implementation strategy of the code, discussion with supervisors and other professional practitioners. See *Section 2.4* for details.

4) Implementation: Develop code from units to a system, and test them systematically for the functionality and elegant developments. In *Section 2.4* and *3*, there are the method and results of the unit and system tests respectively.

5) Integration and Testing: For general using purposes, test the integrated code and do comparison with different running conditions and experimental data for bench marking or looking for the possibility of further development. See *Section 3* for those results.

6) Generalization and Conclusion: Fully document the code and modify it adapted for general cases and conclude the project, with any possible future work as well. More information is in *Section 2.5* and *4*.

2.4. Design Rationale and Implementation Strategy

According to the project plan, there is a flow chart shows the development of the integration code:

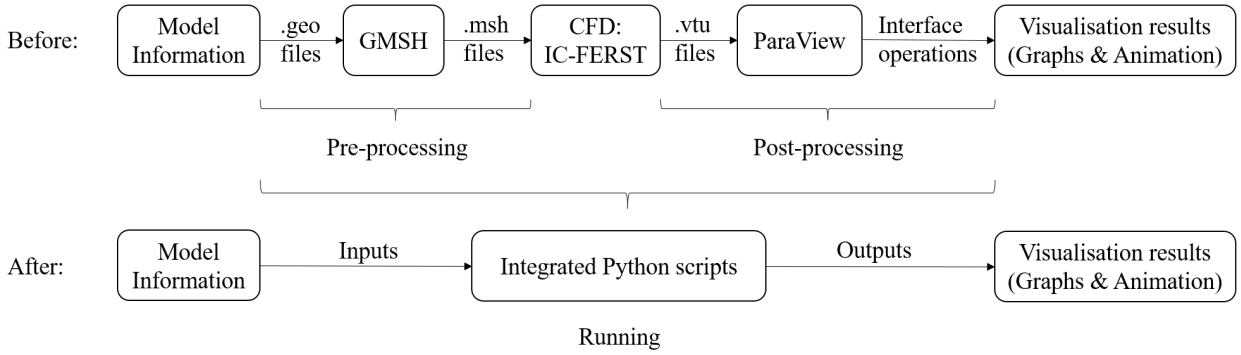

Figure 8: Flow chart of the development of the integration code.

The "Before" and "After" in the chart refer to the conventional method of running a simulation case using IC-FERST as mentioned in *Section 2.2* and the integrated tool in **Python** scripts after development respectively.

For post-processing tool, the work flow is as the following chart:

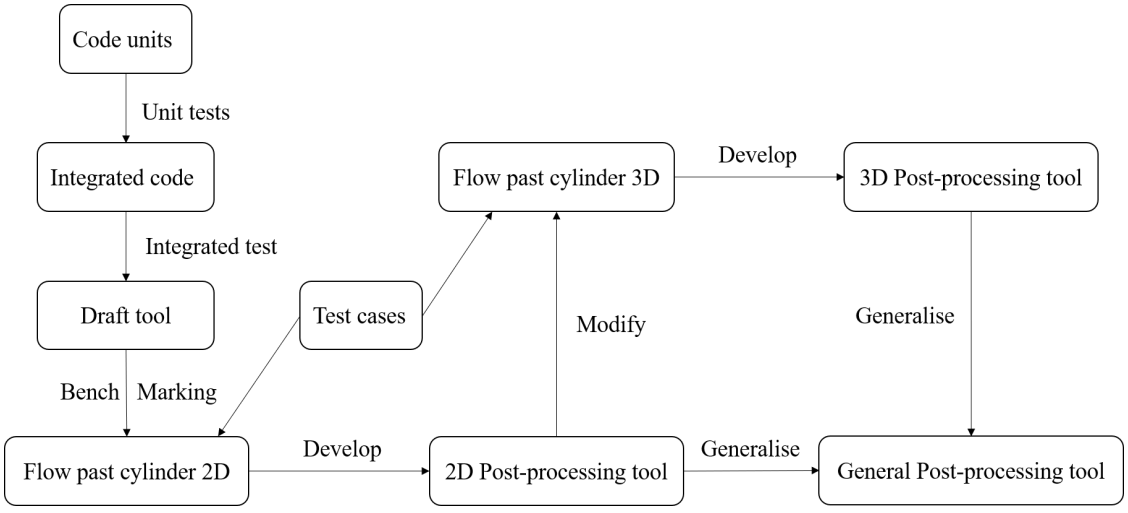

Figure 9: Work flow of developing post-processing tools.

The start point of tool developing is the model of FPC in 2D, which is already introduced in *Section 1* for its physical background and motivation in real cases. The tool obtained can be used to benchmark the whole system (simulation and post-processing) with some experimental data. Then that 2D tool can be modified to 3D FPC model and then becomes a tool with 3D compatibility. In the next stage, generalization of the tool can be implemented on the 2D and 3D tool with more general cases. General cases here refer to some model with different body shapes in the turbulence flow around a bluff body context.

Briefly, there are three stages in the development:

1). 2D tool developed on FPC 2D;

2). 3D tool from modifying 2D tool on FPC 3D;

3). General tool developed from 2D and 3D tool (Generalization).

In the first stage, considering the post-processing phase refers to the circumstance that the geometry, the mesh and the simulation model (the *.geo, *.msh, *.mpml files) are all settled before, there should be a function to start the simulation. Standing upon theoretical explanation of some important quantities and areas of the physical model, the code units focus on getting those data from the simulation results and visualizing them.

The idea above describes three main objectives of the code:

1) Run the simulation and get output files;

2) Read data from output and implement filters;

3) Plotting the data processed by filters.

Each objective is a cluster of a number of sub-objectives. Upon this point, **Classes** in **Python** that provides a means of bundling data and functionality together, is a good way to integrated code pieces. Using **Classes** to wrap parameters and functions can make the code structure clearer and reduce the waste of repeated parameter assignments. Corresponding to three objectives, it is expected to be three **Classes**:

```
class simulation()
class filters()
class plotting()
```

The `class simulation()` aims to start the simulation by running the *.mpml file. The `class plotting()` contains functions to output figures and save them in files. The `class filters()` wraps all of the processing operations to the data. Such as the important physical quantities introduced in *Section 1.1.3*, there are separate functions for extracting specific data along the centerline, along flow transverse, and the drag coefficient along the cylinder surface. The snippet below implements the extraction of the data along the flow centerline (default to streamwise velocity):

```
def flowcenterline(self, data_name = 'Velocity', scalar = False,
                        quanchoose = 0):
        """
        Extracting field data along flow centerline
        :param data_name: field name for extracting
        :param scalar: field type
        :param quanchoose: for vector field (scalar = False), value 0, 1, 2
                            refer to x, y, z
        :return: coordinates and data seperately in array
        """
```

```
        ugrid = self.reader.GetOutputPort()

        #Initial and last coordinate of the probe
        x0 = self.c0[0] + self.D/2
        x1 = self.geolen[0]
        y0 = self.c0[1]
        y1 = self.c0[1]
        z0 = 0.0
        z1 = self.geolen[2]

        # Resolution of the probe
        resolution = 1000

        # Create the probe line
        hx = (x1 - x0) / resolution
        hy = (y1 - y0) / resolution
        hz = (z1 - z0) / resolution

        detector = []
        for i in range(resolution+1):
            detector.append([hx * i + x0, hy * i + y0, hz * i + z0])

        points = vtk.vtkPoints()
        points.SetDataTypeToDouble()
        for i in range(len(detector)):
            points.InsertNextPoint(detector[i][0], detector[i][1], detector[i][2])
        detectors = vtk.vtkPolyData()
        detectors.SetPoints(points)
        probe = vtk.vtkProbeFilter()
        probe.SetInputConnection(ugrid)
        probe.SetSourceConnection(ugrid)
        probe.SetInputData(detectors)
        probe.Update()
        data = probe.GetOutput()
        FS=[]
        for j in range(points.GetNumberOfPoints()):
            FS.append(data.GetPointData().GetScalars(data_name).GetTuple(j))

        # The coordinate along the central line is x
        coordinate = np.array(detector)[:, 0]
        if scalar == True:
            quantity = np.array(FS)[:, 0]
        elif scalar == False:
            quantity = np.array(FS)[:, quanchoose]

        return coordinate, quantity
```

In `flowcenterline`, it creates a probe along the flow centerline with a fixed resolution (refers to the number of points in the probe) at first, and then extracts field data at the probe. Here, the inputs `data_name = 'Velocity', scalar = False, quanchoose = 0` refer to the quantity and the direction are set to be default to streamwise and velocity, but they can be changed be user to other quantities(scalar or vector) and other directions. The function `waketransverse` for data along flow wake transverse is implemented with the similar method .

One thing that needs to be noticed here is the constructor `def __init__(self, geoname, AutomaticFile, AutomaticVTU_Number, parallel, U_in, p0, rho0, shapepoints = 4)` of

`class filters()`. It has twelve instantiation operators from eight input parameters and four automatically extension parameters. The `self.reader` is an instance class object created from `class vtkXMLUnstructuredGridReader()` or `class vtkXMLPUnstructuredGridReader()`, which read *.vtu and *.pvtu files respectively. The `self.c0` comes from the result of the *.geo file (`self.D` and `self.geolen` are as well), storing the position information of the center of the cylinder. The `self.D`, referring to diameter of the cylinder by doubling the radius, is generalized for non-circular-transverse body here by obtaining the approximate radius of the body transverse. The `self.geolen` stores the measurements of the whole model geometry (width and length in 2D). The code snippet below shows the part in the constructor of them:

```
# Reading *.vtu/*.pvtu file
if (len(sys.argv)>1):
    filename   = sys.argv[1]
    vtu_number = int(sys.argv[2])
else:
    filename = self.AutomaticFile
    vtu_number = int(self.AutomaticVTU_Number)
if parallel == True:
    reader = vtk.vtkXMLPUnstructuredGridReader()
    reader.SetFileName(filename+'_'+str(vtu_number)+'.pvtu')
elif parallel == False:
    reader = vtk.vtkXMLUnstructuredGridReader()
    reader.SetFileName(filename+'_'+str(vtu_number)+'.vtu')
self.reader = reader

# Read *.geo file
info = []
infile = open(self.geoname, "rb")
for line in infile:
    if line.split()[0] != 'Point':
        break
    words = line.split('{')[1].split('}')[0].split(', ')

    nums = []
    for i in range(3):
        num = float(words[i])
        nums.append(num)
    info.append(nums)
arrays = np.array(info)

self.c0 = arrays[0]
r = abs(arrays[1, 0] - self.c0[0])
#r = sum(abs(arrays[1:self.bodypoints + 1, 0] - self.c0[0]))/self.bodypoints
self.D = 2 * r
geolen = []
for i in range(3):
    geolen.append(max(arrays[:,i]))
self.geolen = np.array(geolen)
```

Those three **Classes** are expected to be used as a module which can be imported in other **Python** scripts. There is a working script example for the mean streamwise velocity along wake centerline:

```
x_label = 'x/D'
y_label = 'U_CL/U_in'
```

```
title = 'Mean streamwise velocity along wake centreline'

FSs = []
for i in range(avenum):
    AutomaticVTU_Number = i
    a = tl.filters(bodypoints, geoname, AutomaticFile, AutomaticVTU_Number,
    parallel, U0, p0, rho0)
    detector, FS = a.flowcenterline()
    FSs.append(FS)
FS = tl.timeave(np.array(FSs))
detector, FS = a.modify_relative(detector, FS, mquan = U0)

b = tl.plotting(detector, FS, x_label, y_label, title, multi = False)
b.plot_single(figurename, save = False)

# Uncomment blew to add experimental data in plot
# xdata, ydata = tl.add_experimental(detector, FS, filename)
```

Development towards 3D tools has a key point: Modification of the probe in corresponding filters. In 3D tools, the geometry determines the filters should implement the average functionality of the quantity over spanwise direction[11], as the code snippet below:

```
for i in range(resolution+1):
    detector = []
    for j in range(resolution+1):
        detector.append([hx * i + x0, hy * i + y0, hz * j + z0])

    points = vtk.vtkPoints()
    points.SetDataTypeToDouble()
    for j in range(len(detector)):
        points.InsertNextPoint(detector[j][0], detector[j][1], detector[j][2])
    detectors = vtk.vtkPolyData()
    detectors.SetPoints(points)
    probe = vtk.vtkProbeFilter()
    probe.SetInputConnection(ugrid)
    probe.SetSourceConnection(ugrid)
    probe.SetInputData(detectors)
    probe.Update()
    data = probe.GetOutput()
    zFS=[]
    for j in range(points.GetNumberOfPoints()):
        zFS.append(data.GetPointData().GetScalars(data_name).GetTuple(j))

    zFS = np.array(zFS)
    fs = []
    for j in range(zFS.shape[1]):
        fs.append(sum(zFS[:, j])/zFS.shape[0])
    FS.append(fs)
```

Comparased with the 2D code above of the function `flowcenterline`, the extraction loop of data in 3D tool changes into a double loop and averages data spanwise: The inside loop create probe spanwise and average the data, the outside loop works for the filter's direction.

2.5. Code Metadata

Here is a table to give a brief description of the metadata.

Nr.	Code metadata	Description
C1	Current code version	v0.0.0
C2	Repository used for this code version and developer documentation	`https://github.com/msc-acse/acse-9-independent-research-project-anitali555888`
C3	Code versioning system used	git, Github
C4	Legal Code License	MIT license
C5	Software code language used	**Python**
C6	Compilation requirements, operating environments & dependencies	vtk, numpy, matplotlib
C7	IC-FERST version and link	v?.?.? (`http://multifluids.github.io/`)
C8	Operating system	Linux (Ubuntu for best supporting) is required by IC-FERST
C9	Continuous integration	Travis CI

Table 1: Code metadata

The Github repository above includes the README and LICENSE.

3. Implementation and code

3.1. Unit Tests and Integrated Test

For each functional code pieces of functions in those three **Classes**, there is a test function in the file pytest.py. The tests information in pytest.py is as follows:

```
def test_flowcenterline():

def test_waketransverse():

def test_drag_coe_Cp():

def test_modify_relative():

def test_timeave():

def test_add_experimental():

def test_plot_single():

def test_plot_multi():
```

The repository of this project code are under the continuous integration service of Travis CI for building and testing the projects. The .travis.yml file is as Figure X:

```
12 lines (9 sloc)  181 Bytes
```

```yaml
language: python
python:
  - "2.7"

# install dependencies
install:
  - pip install -r requirements.txt
  - python setup.py install

# command to run tests
script: python -m pytest
```

Figure 10: The .travis.yml file content.

For integrated system test, it take a FPC model in 2D as the test case. The results of different testing outputs are as follows:

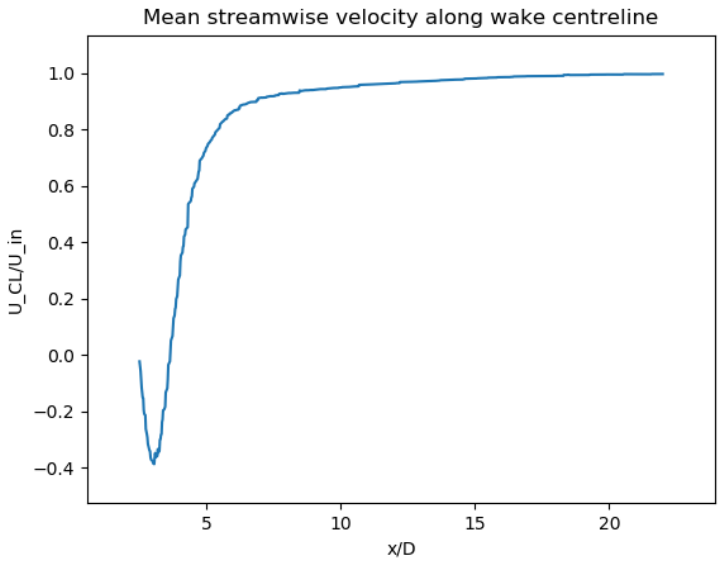

Figure 11: The time-averaged streamwise velocity along flow centerline.

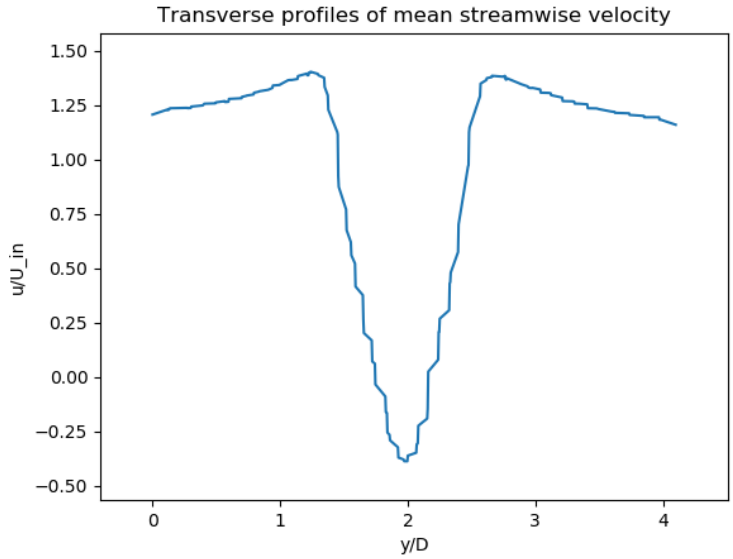

Figure 12: The time-averaged streamwise velocity over a transverse line at x/D = 1.06 in the wake.

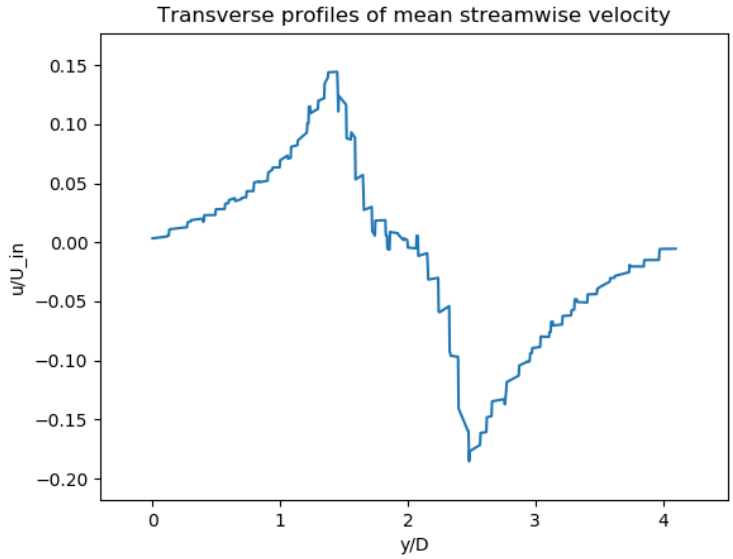

Figure 13: The time-averaged cross-stream velocity over a transverse line at x/D = 1.06 in the wake.

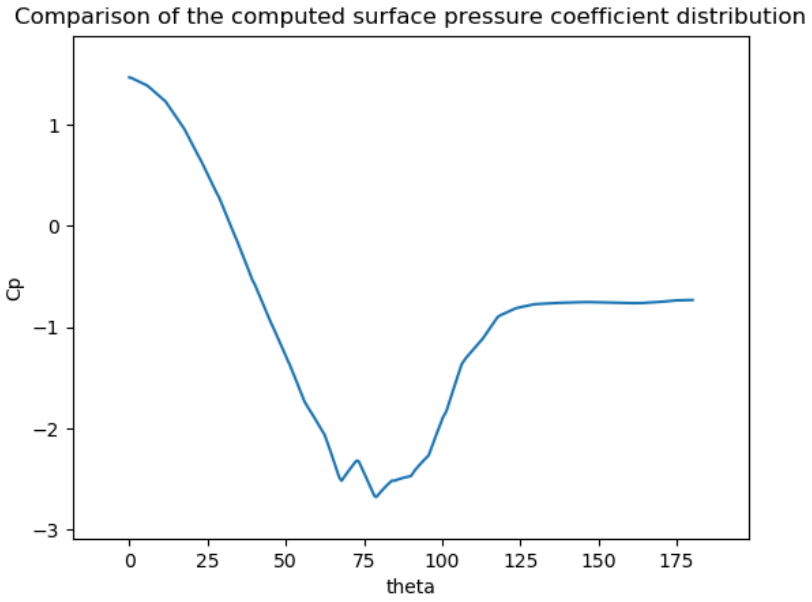

Figure 14: The time-averaged pressure coefficient over the cylinder surface.

3.2. Results with Analysis

Implementation strategy and some key points of the code is explained in *Section 2.4*. According to the work flow of developing the tools, there should be the bench marking results of the test case FPC 2D, and for further verification of the integrated code, the results of the test case FPC 3D are added in the comparison.

For generalization, the general tool has the capacity to handling simulation results from flow past a bluff body with non-circular transverse (2D and 3D) and with various transverse along the cross-stream direction (3D)[5]. That means this tool is capable to analyze models which are created approximately from engineering problems.

3.2.1. Verification and comparisons

The computational details of the bench marking case model FPC 2D are given in Table x and Table X.

Run	Grid resolution	Inlet Velocity in x	Mesh adaptivity	Dumped time period size
Run1	original(p1-5:0.005,p6/9:0.01,p7/8:0.05)	0.039	fixed	0.1
Run2	original(p1-5:0.005,p6/9:0.01,p7/8:0.05)	0.39	fixed	0.1
Run3	original(p1-5:0.005,p6/9:0.01,p7/8:0.05)	0.0039	fixed	0.1
Run4	original(p1-5:0.0005,p6/9:0.001,p7/8:0.05)	0.039	fixed	0.1
Run5	original(p1-5:0.0001,p6/9:0.001,p7/8:0.05)	0.039	fixed	0.1
Run6	original(p1-5:0.00005,p6/9:0.0001,p7/8:0.05)	0.039	fixed	0.1
Run7	original(p1-5:0.005,p6/9:0.01,p7/8:0.05)	0.039	adaptive	0.05
Run8	original(p1-5:0.005,p6/9:0.01,p7/8:0.05)	0.039	fixed	0.05

Table 2: parameters used for present computations

The time averaged flow quantities have been obtained over of 1000 dumped file with dumped period size above, from the beginning of the flow in different models (2D and 3D) and compared with the corresponding measurement data of Lourenco and Shih 1993[24], Wallace 1996[15], HWA experiment 2008[14] and Norberg 1987[7] in Figure x to Figure x.

figures

3d results

4. Discussion and conclusions

4.1. Challenges

The first challenge in this project is the familiarization with such as software like ParaView (operations on GUI)and **Diamond** (the simulation condition setting operations). Via **Diamond**, the model parameters can be modified into from simple constants to python functions of complex formula. And ParaView is so powerful that for a person new to this field it is tough to obtain the visual results as wanted.

Next, the structure plan of the code is another point. For the best convenience of users, the restructuring modification becomes to the most focusing part. At the same time, the implementation of the 3D tool is much time consuming because of the finer mesh and even often causing ParaView crash out.

4.2. Strengths and Limitations

Strengths of this project are the same as the focusing part of users' convenience. At the same time, although this project is on the purpose of development of IC-FERST, the tool here also has

a general functionality for other implementation, like can be used with other simulation code and used on general plotting purpose for input data.

In addition, further improvement about adding more filters is with considerable convenience for users by accessing and editing module file. The unit testing system will keep working and guide users to an advanced modification experience.

With pleased aspects, limitations of this project exist as well.

1). No parallelization of the tool. This part is discussed in the moving forward section for future work.

2). Some algorithms are not quite friendly to users, like the constraints of strict input format of *.geo file if working along the auto-setting geometry infomation option by reading *.geo file.

3). Generalisation of the tool is not quite enough due to the time constraints. Also discussed in moving foward section.

4.3. Moving Forward

As the above discussion, the next stage of developing this projects results is to generalizing and testing.

Testing the integrated code in models with more difficult body shapes and if possible some real cases in engineering problems will helped a lot for searching the best solution to prospective clients - engineers.

There is a brief conclusion of the expected general tool.

1). Adapted to the bluff body with non-circular transverse (2D and 3D);

2). Adapted to the bluff body with various transverse along the cross-stream direction (3D)[5];

3). Generalized to models approximating engineering problems.

Acknowledgements

The author would like to thank the following people and institutes for their contributions to this project: Dr. Asiri Obeysekara for guidance and instruction

Dr. Andre Nicolle for kind advice of the output quantities for better development on its practical use.

The British Petroleum Company plc (BP) for funding this related project.

Reference

[1] P. Anagnostopoulos, G. Iliadis, S. Richardson, Numerical study of the blockage effects on viscous flow past a circular cylinder, International Journal for Numerical Methods in Fluids 22 (11) (1996) 1061–1074.

[2] H. Park, Cfd advantages and practical applications.

[3] H. Khoshdast, V. Shojaei, H. Khoshdast, Combined application of computational fluid dynamics (cfd) and design of experiments (doe) to hydrodynamic simulation of a coal classifier, International Journal of Mining and Geo-Engineering 51 (1) (2017) 9–24.

[4] J. Derakhshandeh, M. M. Alam, A review of bluff body wakes, Ocean Engineering 182 (2019) 475 – 488. doi:https://doi.org/10.1016/j.oceaneng.2019.04.093.
URL http://www.sciencedirect.com/science/article/pii/S0029801818307418

[5] K. Su, S. Butt, J. Yang, H. Qiu, Coupled dynamic analysis for the riser-conductor of deepwater surface bop drilling system, Shock and Vibration 2018.

[6] W. Li, Y. Tang, Q. Liu, Y. He, Simulation and experimental analysis of critical stress regions of deep-water annular blowout preventer, Engineering Failure Analysis (2019) 104161doi:https://doi.org/10.1016/j.engfailanal.2019.104161.
URL http://www.sciencedirect.com/science/article/pii/S1350630718312949

[7] C. Norberg, Effects of reynolds number and a low-intensity freestream turbulence on the flow around a circular cylinder, Chalmers University, Goteborg, Sweden, Technological Publications 87 (2) (1987) 1–55.
[8] P. Beaudan, P. Moin, Numerical experiments on the flow past a circular cylinder at sub-critical reynolds number, Tech. rep., Stanford Univ CA Thermosciences Div (1994).
[9] M. Sato, T. Kobayashi, A fundamental study of the flow past a circular cylinder using abaqus/cfd, in: 2012 SIMULIA Community Conference, 2012.
[10] M. Braza, R. Perrin, Y. Hoarau, Turbulence properties in the cylinder wake at high reynolds numbers, Journal of fluids and Structures 22 (6-7) (2006) 757–771.
[11] S. M. Rajani B.N, A Kandasamy, Les of flow past circular cylinder at re = 3900, Journal of Applied Fluid Mechanics 9 (3) (2016) 1421–435.
[12] M. Brede, H. Eckelmann, D. Rockwell, On secondary vortices in the cylinder wake, Physics of Fluids 8 (8) (1996) 2117–2124.
[13] V. D'Alessandro, S. Montelpare, R. Ricci, Detached–eddy simulations of the flow over a cylinder at re= 3900 using openfoam, Computers & Fluids 136 (2016) 152–169.
[14] P. Parnaudeau, J. Carlier, D. Heitz, E. Lamballais, Experimental and numerical studies of the flow over a circular cylinder at reynolds number 3900, Physics of Fluids 20 (8) (2008) 085101.
[15] L. Ong, J. Wallace, The velocity field of the turbulent very near wake of a circular cylinder, Experiments in fluids 20 (6) (1996) 441–453.
[16] P. Bearman, M. Zdravkovich, Flow around a circular cylinder near a plane boundary, Journal of Fluid Mechanics 89 (1) (1978) 33–47.
[17] M. M. Zdravkovich, Flow around circular cylinders: Volume 2: Applications, Vol. 2, Oxford university press, 1997.
[18] J. Franke, W. Frank, Large eddy simulation of the flow past a circular cylinder at red= 3900, Journal of wind engineering and industrial aerodynamics 90 (10) (2002) 1191–1206.
[19] M. Mallick, A. Kumar, N. Tamboli, A. Kulkarni, P. Sati, V. Devi, S. Chandar, Study on drag coefficient for the flow past a cylinder, International Journal of Civil Engineering Research 5 (4) (2014) 301–306.
[20] M. Jackson, J. Percival, P. Mostaghimi, B. Tollit, D. Pavlidis, C. Pain, J. Gomes, A. H. Elsheikh, P. Salinas, A. Muggeridge, et al., Reservoir modeling for flow simulation by use of surfaces, adaptive unstructured meshes, and an overlapping-control-volume finite-element method, SPE Reservoir Evaluation & Engineering 18 (02) (2015) 115–132.
[21] P. Salinas, D. Pavlidis, Z. Xie, A. Adam, C. Pain, M. Jackson, Improving the convergence behaviour of a fixed-point-iteration solver for multiphase flow in porous media, International Journal for Numerical Methods in Fluids 84 (8) (2017) 466–476.
[22] U. A. et al., The ParaView Guide Community Edition (2019).
URL http://www.sciencedirect.com/science/article/pii/S1350630718312949
[23] W. W. Royce, Managing the development of large software systems: concepts and techniques, in: Proceedings of the 9th international conference on Software Engineering, IEEE Computer Society Press, 1987, pp. 328–338.
[24] L. Lourenco, Characteristics of the plane turbulent near wake of a circular cylinder, A particle image velocimetry study.