coriofoam Documentation

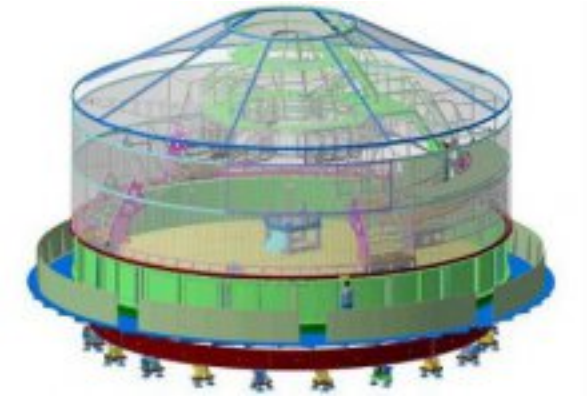
Release 0.0

LEGI

Jan 30, 2018

Contents

Simulating flows in the Coriolis platform with OpenFOAM. Tutorials are proposed and explained. The solver is detailed



CHAPTER 1

what is CorioFOAM?

CorioFOAM is an OpenFOAM solver adapted to the Coriolis Platform.

1.1 Coriolis platform

The Coriolis Platform is the largest experimental rotating platform in the world located in the LEGI lab (Laboratory of Geophysical and Industrial Flow).

The Coriolis platform is used for the simulation of ocean/meteorological phenomenons, geophysical flows and the study of the Coriolis force.

Its main activity is the experimental modeling of geophysical flows and oceanographical and meteorological phenomenons.

It is 13 meters wide tank which allows it to take into account the Coriolis force that results from the rotation of the Earth.

Indeed the large size of the tank provides access to the inertial regimes that characterize ocean dynamics, with little influence of viscosity and centrifugal force.

Various changes to the fluid can be made : there can be density stratification or even added topography.

Laboratory experiments can thus provide a support to model ocean dynamics and develop their physical parameterizations.

The measuring equipment includes ultrasonic velocity meters, profilers for salinity and temperature.

Measures are increasingly focused on digital image processing.

Concentration fields are obtained by laser fluorescence (LIF), and velocity fields by image correlation (PIV).

These techniques allow three-dimensional measurements in volume by a scanning system of the laser plane.

But they are limited in scale.

1.2 OpenFOAM

OpenFOAM (for "Open source Field Operation And Manipulation") is a C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics problems, including computational fluid dynamics (CFD).

The code is released as free and open source software under the GNU General Public License.

CorioFOAM is no more, no less a solver developped with OpenFOAM.

Thanks to CorioFOAM, we can run simulations. We used it through several tutorials

CHAPTER 2

Tutorials

2.1 The parameters of the tutorials

The tutorials and their results depend on the parameters defined in certain files. Below are described these files.

2.1.1 0_init

- Requires : dimensions, internal field (set value in domain), boundaryfield (set boundary conditions)
- Contains values :
  - alphat = turbulent viscosity
  - P = pressure
  - p_rgh = dynamic pressure
  - rhok = density
  - U = velocity

2.1.2 Constant

- polymesh file : contains blockMeshDict, blockMeshDict_orig, make_blockMeshDict.py. These are programs to change the mesh
- g : defines gravity field
- MRFproperties : set platform rotation speed
- transportproperties : set values laminar viscosity, Prandtl number, turbulent Prandtl number
- turbulence properties : sets simulation type :
  - Laminar (direct simulation)
  - RAS (Reynolds Average Simulation)

  - LES (Large Eddy Simulation)

2.1.3 System

- controldict : sets input parameter, essential for database creation (timesteps, simulation period/length)
- decomposeParDict : modifies number of processors used, and how they are used
- funkySetFieldsDict : sets density gradient
- fvSchemes : numerical schemes (implicite, explicite)
- fvSolution : equation solver,tolerances and algorythms control
- topoSetDict : Operates on cellSets/faceSets/pointSets through a dictionary (by default system/topoSetDict). The results are stored in constant/polyMesh/sets

2.1.4 .sh files

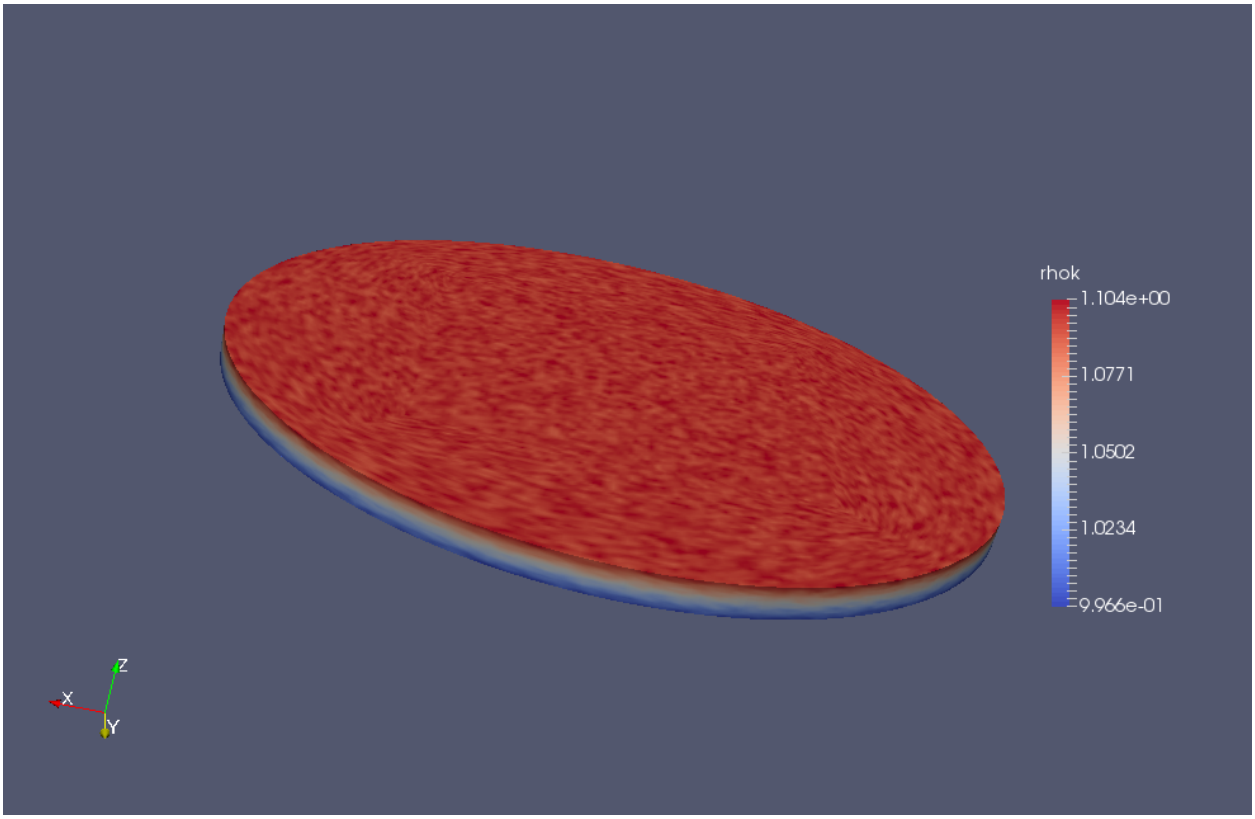
- clean.sh : This command is used to remove the initial files created from any previous simulations like the time directories and the polymesh content in "Constant".
- init_simul.sh : This command generates the mesh from the "blockmeshDict" file and copies the "0_init" information into coriofoam.
- init_parallel.sh : This command decomposes a case to be run in parallel, it uses the file "decomposeParDict".
- launch_sequencial.sh : This command is used to launch the simulation on the computer and the output is sent to "log.txt".
- lauch_parallel_local.sh : This command defines the number of cores with the dict file and sets the output to "log.txt".
- lauch_parallel_cluster.sh : This command launches a parallel simulation and should be used with the cluster. The simulation job is named with the time and date. It sets the output to "log.txt".

2.1.5 other files

- template.oar (This is were simulation outputs usually go when they are not redirected to "log.txt")
- plotcontourux.py (plots the ux contour in center plane with python)
- empty_for_paraview (This is used to view the simulation results with paraview)

2.2 Tutorial gravitational_instability_norot

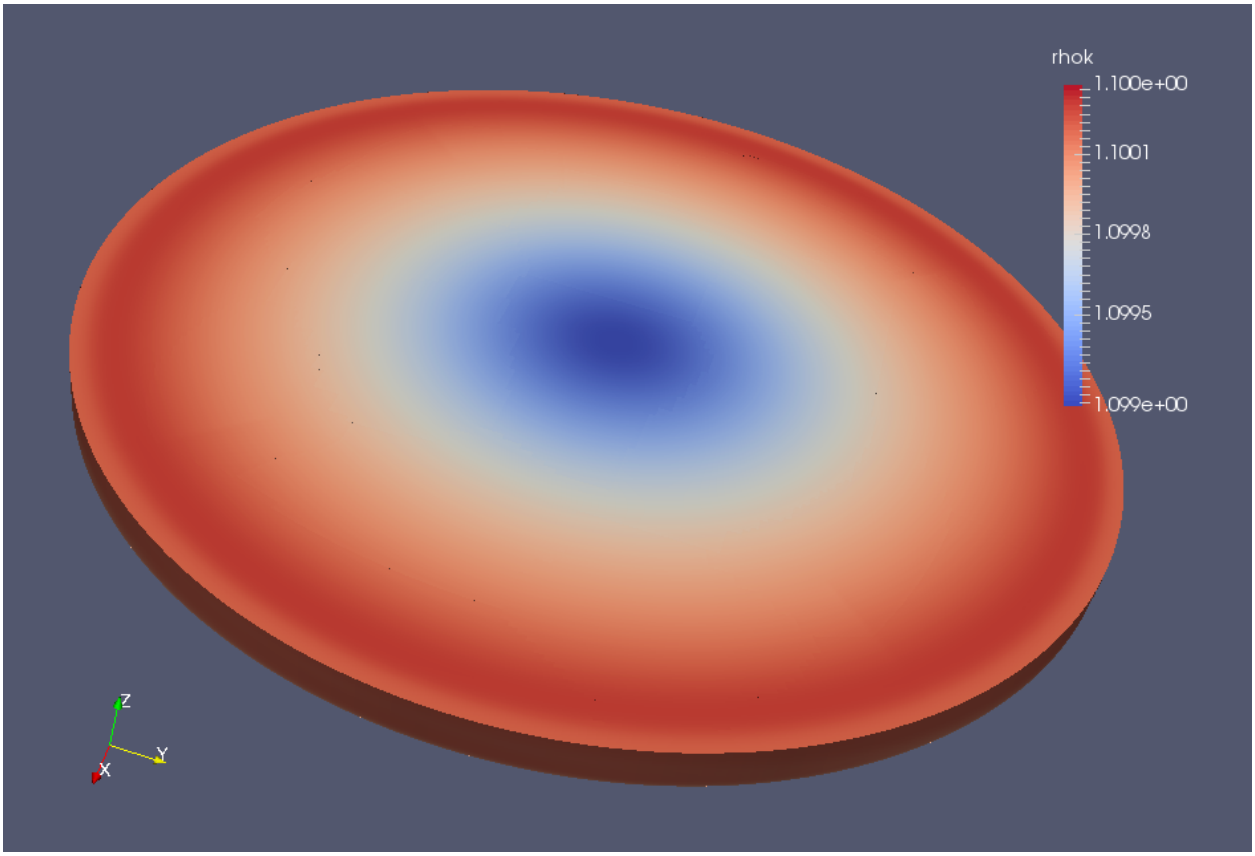
The time 0 of this case has a density stratification which is higher at the top than at the bottom. The platform contains a fluid whiwh is similar to honey and rotates at a speed of 0.2 rad/s. Thus the evolution of the fluid is determined by the influence of gravity and of the coriolis force. This simulation uses 4 processors.



2.3 Spinup case simulation

The time 0 of this case has a density stratification which is lower at the top than at the bottom. The platform contains a fluid whiwh is similar to honey (nu = 0.01, Pr = 7) and begins to rotate at a speed of 0.2 rad/s. This simulation uses 16 processors.

The fluid with the higher density latches very quickly onto the edges of the platform. The fluid doe snot become homogeneous with time, there is still a separation in the density of the fluid. After a long time the higher density stays closer to the edges even if there is a strong diffusion.



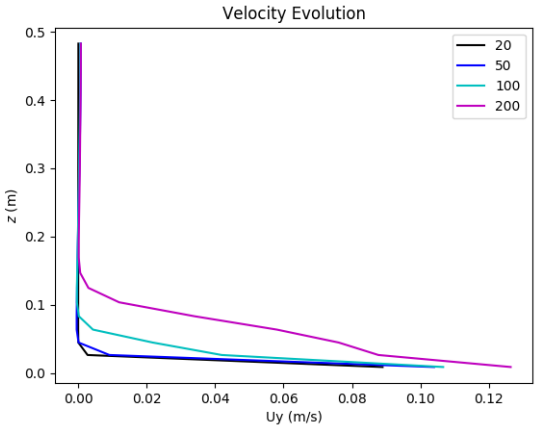
2.4 Spinup stratificated komega case

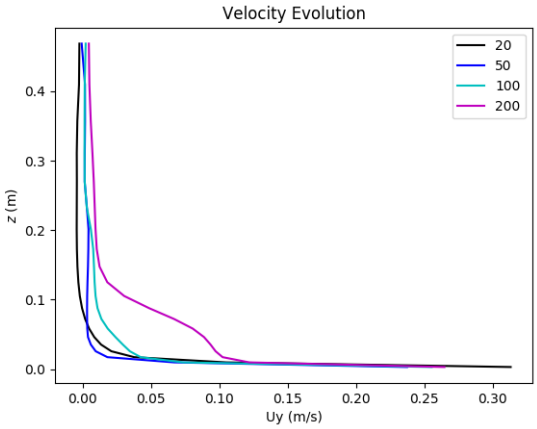
Initially this case also has a density stratification which is lower at the top than at the bottom. The platform contains a fluid whiwh is similar to water (nu = 10-6, Pr = 7) and begins to rotate at a speed of 0.2 rad/s. But a k-omega turbulence model has been implemented. This simulation uses 16 processors.

The results are similar to the spinup case with honey but they take a lot longer to reach the final state, in fact we did not have the time to reach it.

2.5 Interpretation and Validation of the Solver and the K-Omega Model

One of the important question we had to answer was whether the K-Omega model is useful or not. In order to do so we had to run the spin-up strat simulations on water, with and without the K-Omega model, on water. We then plotted the velocity evolution for the two cases :

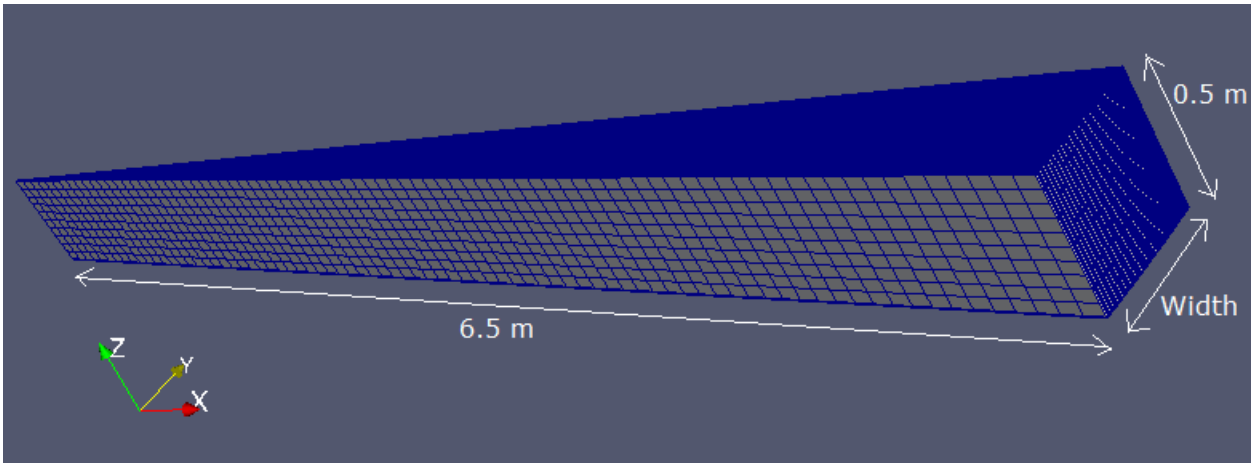




Concerning the physics behind problem, at the beginning (between time 0 and 100), there is no turbulence therefore there is only the rotation of the molecules. Since there is only the rotation of the molecules, the only mechanism at work is the molecular viscosity, which explains that there are very important velocity gradients at the beginning of the experience. Then, turbulence appears and is diffused in the flow. But since the turbulent viscosity is 1000 times larger than molecular viscosity, the gradients can no longer be enormous, so we witness a diminution of the velocity gradients when time passes. On the graph located on the left, the effect of turbulence diffusion on the velocity profile is obvious (time 200), whereas on the right the profile obtained is not very precise (time 200). Moreover the profiles on the left seem smoother whereas the profiles on the right seem to be discontinuous, they are not very well rendered. The velocity profile displayed with K-Omega seems to be more accurate, so the K-Omega model is necessary.

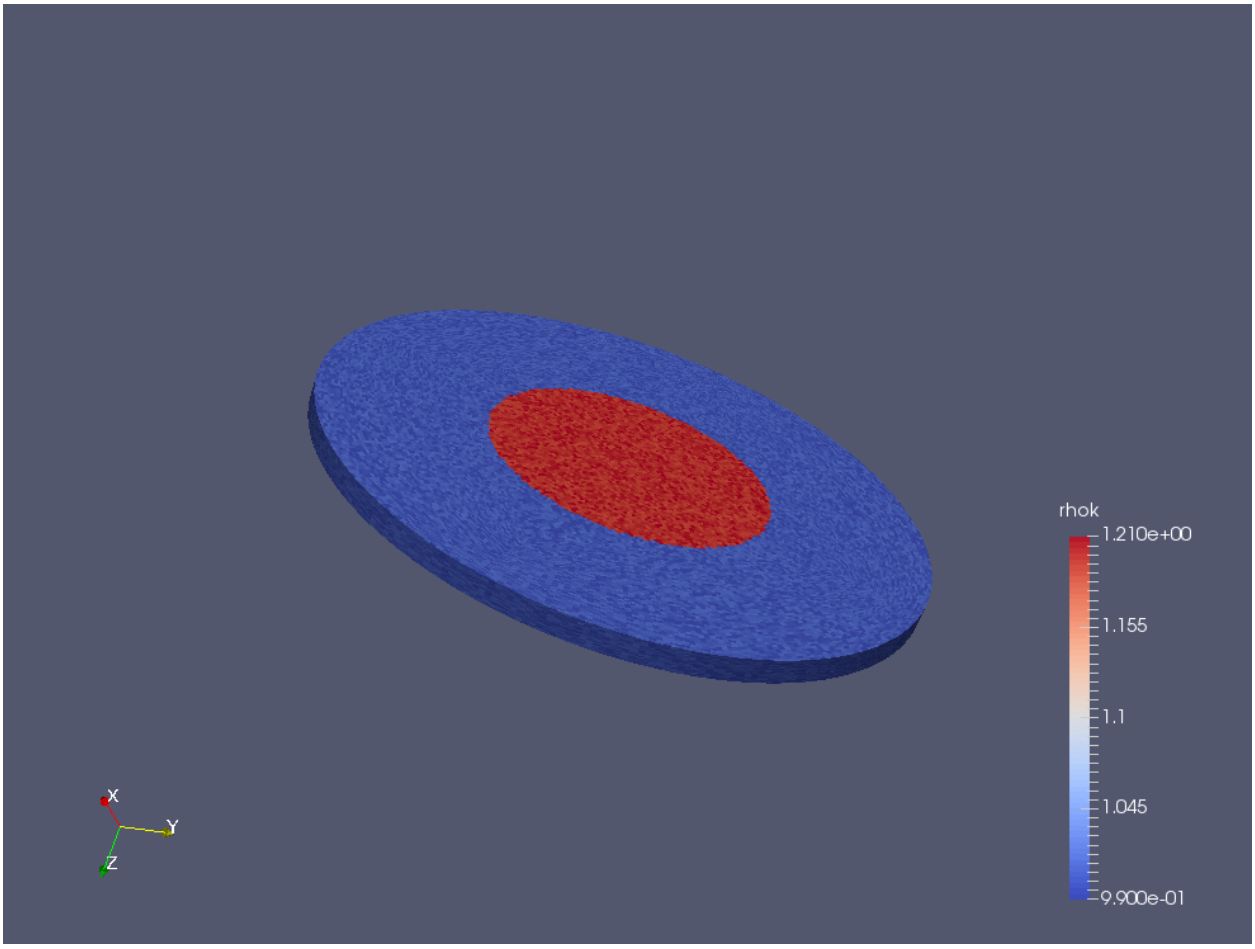
2.6 Spinup stratificated piece of cake case

For calculation time reasons, a new simulation was created. The idea was to reduce the calculus domain to a small "piece of cake" to accelerate the calculations. The initialisation of this case is the same as the SpinupStrat K-Omega case.



2.7 Center cylinder

The initial condition at t = 0 of this new tutorial is the following; the platform is rotating in permanent regime, it means that the case of solid rotation is valid. In the center of the platform, there is a cylinder of fluid with a higher density than the fluid outside of it. At time t = 0, the fictive border between the two fluid with different densities disappears.



Thanks to this tutorial, we would like to bring out the Coriolis effect. At time t=0, we are in solid rotation to avoid

a quick mix between the two phases. The expected effect is the following; the center cylinder will collapse because of gravity, and the dense fluid will go to the edges because of the inertial force. But we hope that the trajectory of a particle of fluid will be deviated (it means that it will not follow a radius) on the right, to bring out the Coriolis effect.

The work of initialization must be made in the file named "funkysetFields", located on the folder "system".

- Conditions on density

We define the density field (rhok) with the following expression on C++ language:

```
{
variables
(
"center_cylinder_radius=3;"
);
field rhok;
expression "(sqrt(pow(pos().x,2) + pow(pos().y,2))) < center_cylinder_radius ? 1.2 +␣
↪0.01 * 2 * (rand() - 0.5) : 1. + 0.01 * 2 * (rand() - 0.5)";
}
```

It means that for each point located in the 3 meters wide cylinder (value which can be modified in the script), the density is imposed to 1.2 and the density is equal to 1.0 everywhere else. In the expression, we introduce a part of randomness with the rand() function. It is useful to have a model which is more realistic than a "perfect numerical simulation". Without this precaution, it would be difficult to model something with a realistic evolution. Thanks to the non-uniformity of the field, some imperfections appear and those imperfections will enable the code to evolve correctly.

- Conditions on Velocity

The velocity is given by: U = rω for a solid rotation:

```
{
variables
(
"w=0.2;" //rotation speed
);
field U;
expression "vector(w * sqrt(pow(pos().x,2) + pow(pos().y,2)) *
(pos().x/sqrt(pow(pos().x,2) + pow(pos().y,2))) , w *
sqrt(pow(pos().x,2) + pow(pos().y,2)) * (pos().y/sqrt(pow(pos().x,2) + pow(pos().y,
↪2))),0)";
}
```

- Conditions on Pressure

Because of the solid rotation, the pressure field is not uniform. There was a difficulty since the numerical model does not represent exactly what is happening in the reality. On the real platform, the surface of fluid is a free surface and during an experiment, the free surface is determined by the relation:

gradP = ρg + ρrω2ur

After resolution, the mathematical expression of the free surface is:

$z = z_0 + \frac{\omega r}{2g}$

The expression of the pressure is:

$P = rho * g * (z - z_0) + \frac{\rho \omega r}{2}$

The definition of the pressure field is specified in C++ language as fol- lows:

```
{
variables
(
"w=0.2;" //rotation speed
"z0=-2943;"
"g=9.81;"
"rho=1000;" //water
);
field p_rgh;
expression "rho * g * (pos().z - z0)+rho * ((pow(w,2) * pow(sqrt(pow(pos().x,2) +␣
↪pow(pos().y,2)),2)/2))";
}
```

The definition of the pressure field might not be correct, because there were many problems with the pressure field. It has to be defined with one constant on a particular point. It means that the valeu of pressure has to be define on a chosen place of the platform. But imposing this value creats a special point which becomes a kind of anomaly for the simulation. Moreover, the definition of p and p-rgh are sometimes confusing, and it is difficult to know which field needs to be registerd. It exists also situations where the code requires adimensionnal values but it is also difficult to know when. Those problems made the tutorial unsucessful because we did not succeeded to launch a correct simulation.

A solution can be to impose the pressure field not only on a point, but on the complete surface (for example on the surface on the top) for each time step. It might give a more stable simulation. But there is still a problem with this proposal. As seen before, the pressure field depends on the density but it is not uniform on the top surface. For the first step, it is not difficult to define the pressure field with the condition on the radius but for the next step : the density field is unknown !

A lack of time did not enable us to go further for this tutorial, but we hope that those indications will be useful for others.

2.8 Post processing

Python codes :

- plot_crosssection.py : Plots the density of a section at a chosen time and plots the velocity on a section also at a certain time.
- plot_crosssection_profiles.py : Plots the velocity on the z axis from a point where the radius is equal to ‘ m at a chosen time.
- plot_evolution.py Thanks to this script we can visualize the evolution of the velocity profile. This can help us determine how close we are to the solid rotation.
- plot_bottom_boundary.py : This script helps vizualize the velocity value of the first points at the bottom of the tank. We can vizualize how the boundary layer is calculated and the errors due to bad mesh.

CHAPTER 3

Solver

The solver is devided into two parts, the first part of the script includes the Files called outside of the main loop. Then there is the second part of the solver which is the loop where the equations are solved with the PIMPLE method.

3.1 The files called outside the main

- fvCFD.H : collection of fundamental tools for performing finite volume calculation
- singlePhaseTransportModel.H : a simple single-phase transport model based on the viscosity model
- turbulentTransportModel.H : type definition of the turbulence model for incompressible flows, there are three models available:
  - LAMINAR -> direct simulation, Navier-Stockes equations are solved on the all calculation domain (no turbulence model)
  - RAS -> uses Reynolds Averadged simulation modelling
  - LES -> uses large eddy simulation modelling
- fvOptions.H : provides a base set of controls (source type, start time, duration,...). Mainly used to add source terms to equations of the mathematical model (for instance the coriolis force).
- pimpleControl.H : supplies convergence information/checks for the PIMPLE loop.

3.2 The files called inside the main

- postProcess.H : executes application FUNCTIONOBJECT to post-process.
- createMesh.H
- createControl.H : Control of the PISO algorithm
- createFields.H

- createFvOptions.H
- createTimeControls.H : Read the control parameters used by setDeltaT
- CourantNo.H : calculate mean and maximu Courant number.
- setInitialDeltaT.H : sets initial time step.
- initContinuityErrs.H : Declares and initiates the cumulative continuity error.

All these files are not specific to the coriolis case, they are found in almost every OpenFOAM solver.

Most of the constants and parameters declared in these files are not to be changed within these files. You should change them in the system folder of the tutorials (Dict files) which will then change the .H files.

3.3 The loop and the files called in the loop

3.3.1 The files called in the loops

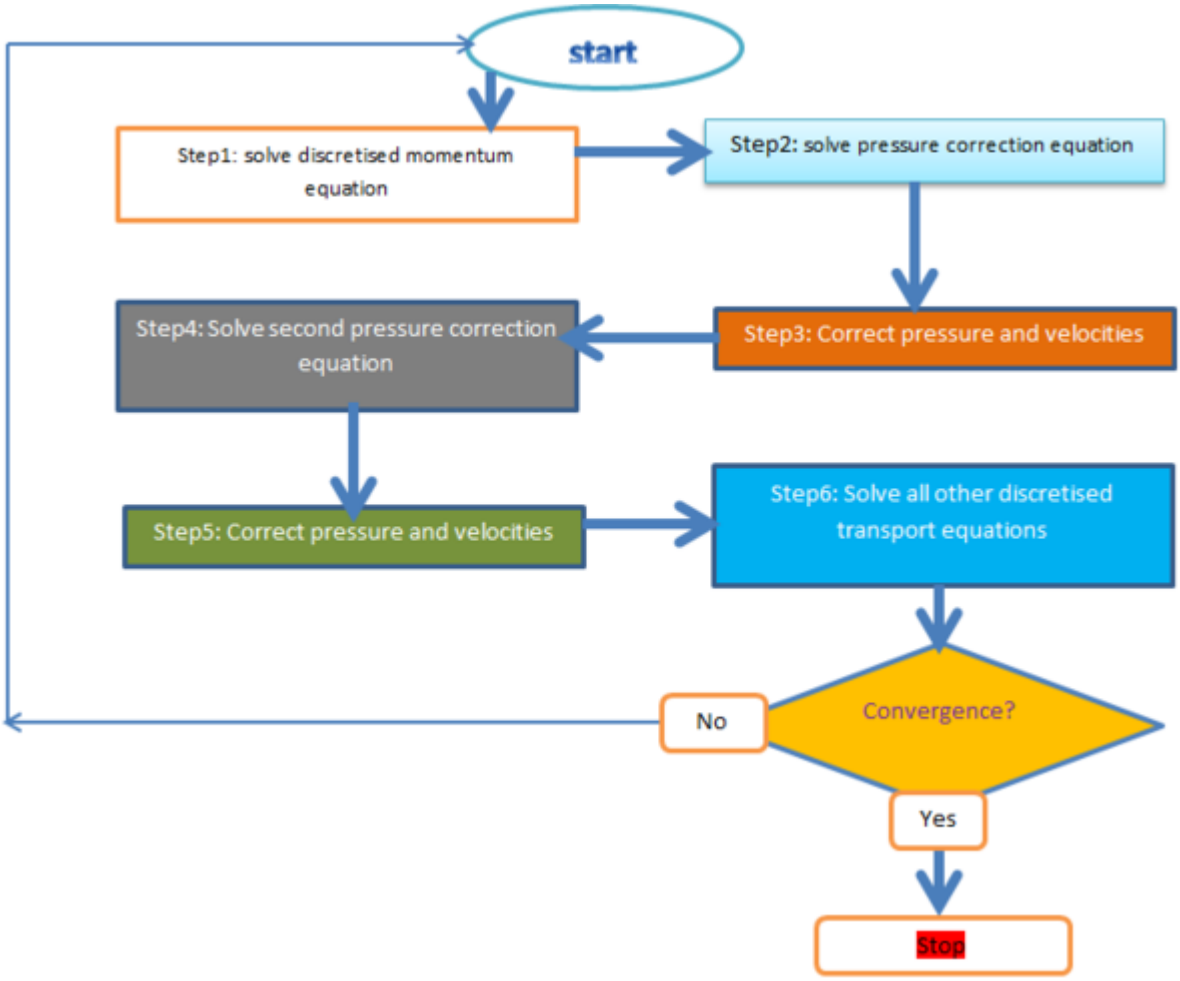
UEqn.H : Contains the momentum equation which is the equation solved to obtain the velocity field.

rhokEqn.H : Contains the equation solved for the density which is the internal energy equation.

pEqn.H : Contains the equation solved for the pressure. It is simply the divergence of the momentum equation.

3.3.2 The solver loop

Coriofoam code uses the PISO method combined to the SIMPLE method (PIMPLE). Basically it is a SIMPLE loop nested in a PISO loop. This is an explanatory scheme of the PISO method :



source image

The coriofoam solver uses 3 time-loops. In the second one the momentum equation (file UEqn.H called) and the density equation (file rhokEqn.H called) are solved. Whereas, the pressure equation is only solved in the third time loop (correction).

3.4 Momentum equation with Boussinesq approximation

The version of the momentum equation solved is the version with the boussinesq approximation.

Boussinesq approximation :

- incompressible flow
- density and temperature do not vary much overall around their mean values

Rotating terms have to appear in our Navier-Stockes equation : The coriolis force and the driving force.

velocity Equations (UEqn) $D_t\vec{u} = -\vec{\bigtriangledown}\frac{p}{\rho_0} + \nu\Delta\vec{u} + \frac{\rho'\vec{g}}{\rho_0} - \Omega\vec{r} - 2\vec{\Omega}\wedge\vec{u}$

In this equation, we have the classic Navier-stockes equation, but also the coriolis force $Fc = -2\Omega\vec{u}$ the driving force $Fe = -\Omega\vec{r}$ And the flotability term $\frac{\rho'\vec{g}}{\rho_0} = -b\vec{e_z}$

The pressure equation is found by taking the divergence of the momentum equation :

$\Delta p = -\rho(\bigtriangledown(u \cdot \bigtriangledown u) - \Omega \bigtriangledown r - 2 \bigtriangledown (\vec{\Omega} \wedge \vec{u}))$ is the equation that we have after simplifications

The pressure and the velocity equations are coupled, that's why we use the PISO method.

Then we have the rhô equation, taking into account the flotability term :

$D_t(b) = K\Delta b$

The incompressible property give us the following equation :

$\frac{\partial \rho}{\partial t} + \vec{\bigtriangledown} (\rho \vec{u}) = 0$

All these equations are solved or used in the PISO algorithm, defined in the file Coriofoam.C

3.5 Important parameters of the loop

The important control parameters of the loop are defined and can be changed in the system folder. Especially in the Dict files located in 0_init, constant and system. These folders can be found in the tutorials part.

CHAPTER 4

Mesh

4.1 The Geometric Discretization of the Physical Domain

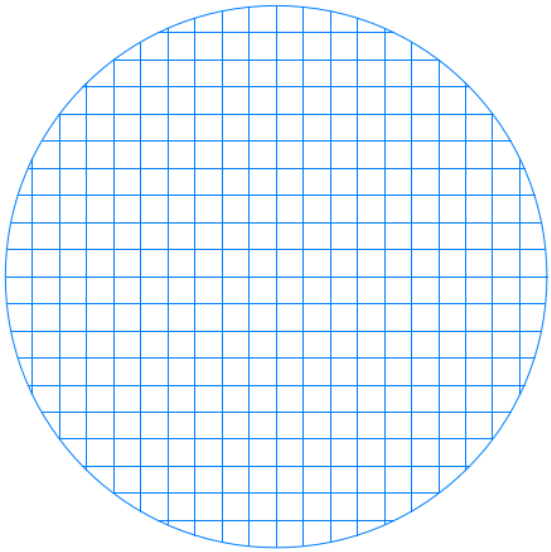
4.1.1 Mesh Quality

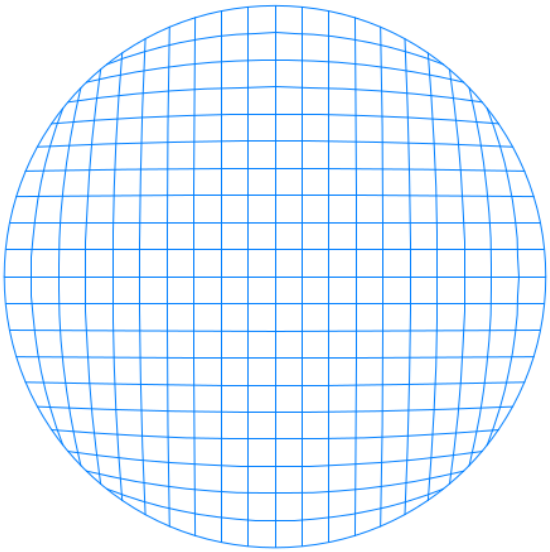
Obtaining a mesh for the domain is one of the primary tasks of the project. The task is achieved by discretization of the model. Discretization of the model can be expressed as generating grids in two or three dimensional geometry and results in mesh. The mesh consists discrete elements defined by a set of vertices and it is bounded by faces. The cells or elements that compose mesh system are not overlapped and completely fill the domain. The elements are bounded by faces that are possibly shared by adjacent elements, except at the boundaries. The mesh plays a vital role in accuracy of the solution and minimizing the computer resources, therefore generating a good mesh is a foremost task in any CFD simulation. The basic rules for generating mesh that followed for the project are listed below:

- All faces do not intersect each other.
- Mesh density is adequate to capture all relevant flow features.
- The mesh located at the boundary is fine enough to resolve the boundary layer flow.
- Aspect ratio, which is the ratio of longest edge length to shortest edge length of the element, is close to unity.

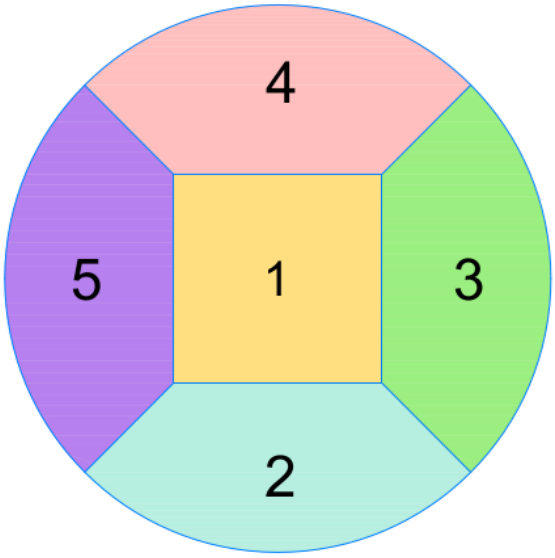
4.1.2 Discretization of the Coriolis Platform

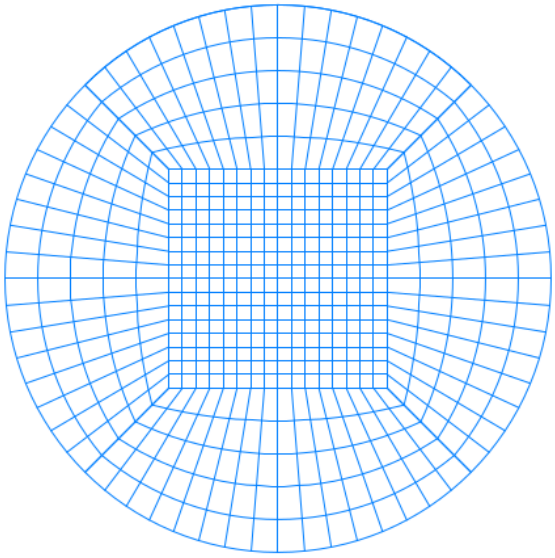
The Coriolis platform has cylindrical shape, therefore discretization of the domain is not complicated as a complex domain which has irregular geometric features. The initial grid system that can be thought at first can be similar with the figures shown below. However, when the above listed considerations are taken into account, it is observed that the aspect ratios of the faces close to the boundary are in conflict with the rule.



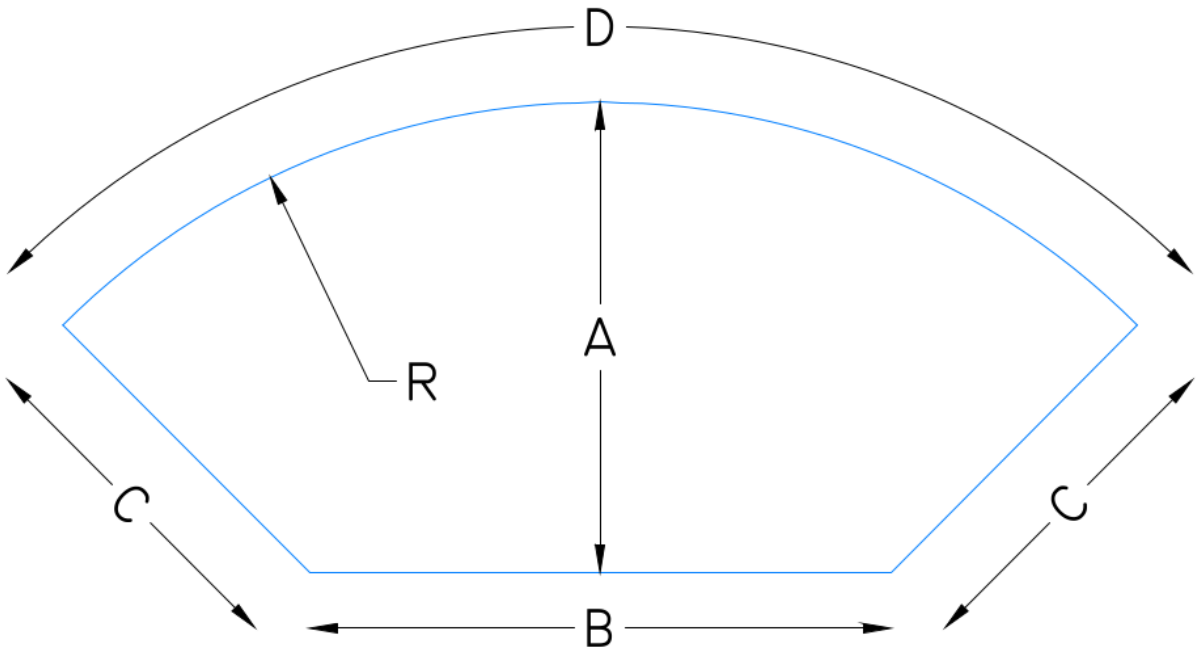


A better solution obtained by simply placing a rectangular inside the circle and dividing the domain by five sub-area as shown in the below figure. The obtained mesh can be classified as structured multi-block mesh.





The edge length of the rectangular that is placed inside the circle affects the mesh quality. If the edge length of rectangular is selected relatively small then the mesh is dense inside the rectangular but very coarse in the outside and vice versa. Therefore optimal length for the edge length was needed to be determined and calculated according to the following approach.



If the sub-area shown below is discretizated such that the optimal mesh is obtained, the ratio of the C to A and B to D must be equal to unity. However it is impossible since the concerned shape is circular. Therefore a compromised solution is to equalize the ratios with each other.

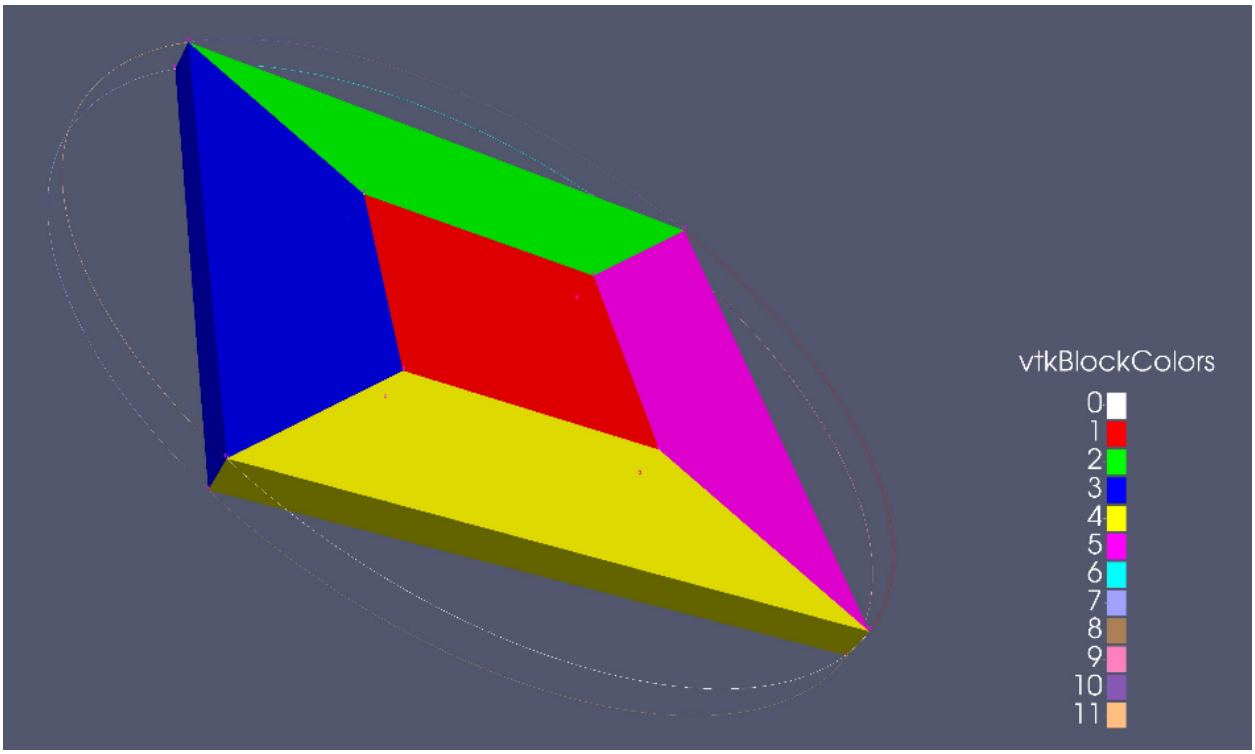
$$\frac{A}{C} = \frac{B}{D} \Rightarrow B = \frac{\left(R - \frac{B}{2}\right) \cdot \left(\frac{\pi R}{2}\right)}{R - \frac{B}{\sqrt{2}}} \Rightarrow B^2 - B.R.\left(\frac{\pi}{\sqrt{2}} + 2\right) + R^2.\pi = 0$$

$R = 6.5$ m $\Rightarrow B =$ Edge of the Rectangular $= 6.27 \approx 6.5$ m

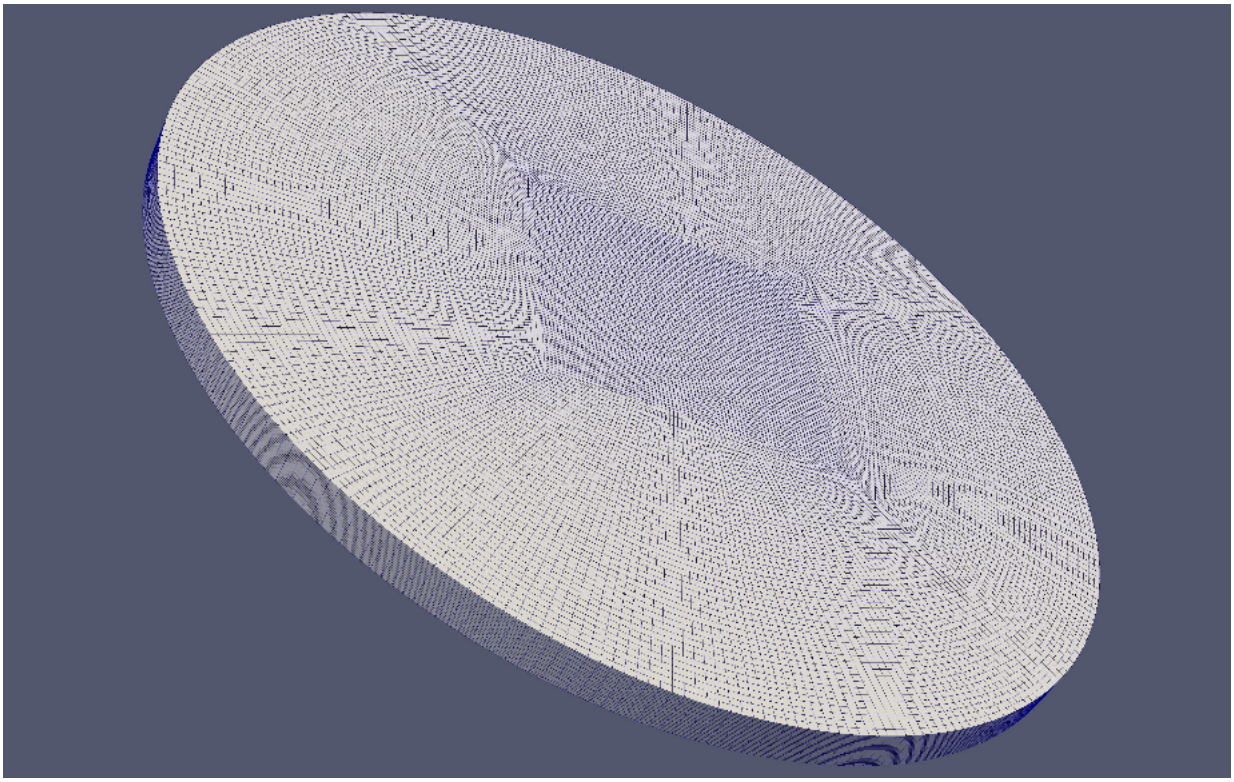
Source

4.1.3 Generating the Mesh with *blockMesh*

After obtaining the optimal length for the edge of inner rectangular, the domain was divided into cells with an aspect ratio close to unity, by a mesh generator called *blockMesh*. The *blockMesh* is a basic and reliable mesh generator of *OpenFOAM* that can be used for simple and multi-block grid systems. Inputs that are required for the process such as vertices, blocks and special edge definitions are stored in a dictionary called *blockMeshDict*. First of all, vertices which are the points of intersection for every individual line in sub-areas (Figure 2) were defined in the dictionary. After that, blocks and edge definitons were defined in order to connect specified vertices in accordance with edge definitions.



Finally, specifying the division intervals for the blocks resulted a mesh for the Coriolis Platform. Also a *Pytho* script was generated to allow the user to modify *blockMeshDict* file in easy and rapid way.



4.2 Python codes for the mesh

The following script will generate a blockMeshDict file for a multiblock mesh of a cylinder. The mesh will be created with O-H Grid method, therefore it will consist a square block located in the center and four other

Variables :

- r : Radius if the Cylinder
- height : Edge Length of the Inner Rectangular
- l2 : x and y for the First Vertice
- divx : Number of Cells in x Direction
- divy : Number of Cells in y Direction
- divz : Number of Cells in z Direction
- divr : Number of Cells Along Rectangular

Python code for the mesh:

```
#!/usr/bin/env python

import os
from math import sin, cos, pi
from runpy import run_path

keys_input = ['r', 'height', 'l2', 'divx', 'divy', 'divz', 'divr']


path_in = os.path.join('system', 'input_make_blockMeshDict.py')
if not os.path.exists(path_in):
    raise ValueError('No file ' + path_in)

d = run_path(path_in, {})

for key in keys_input:
    if key not in d:
        raise ValueError('The variable ' + key +
                         'has to be defined in the file ' + path_in)

default_params = {'converttometer': 1, 'zgrading': 50}

for key, value in default_params.items():
    if key not in d:
        d[key] = value


txt = '''
FoamFile
{
version 2.0;
format ascii;
class dictionary;
object blockMeshDict;
}
'''

txt += '''
```

```
convertToMeters {k};

vertices
(
( {l2} {l2} 0.0) // NorthEstsq = 0
(-{l2} {l2} 0.0) // NorthWestsq = 1
(-{l2} -{l2} 0.0) // SouthWestsq = 2
( {l2} -{l2} 0.0) // SouthEstsq = 3

( {a} {a} 0.) // NorthEst = 4
(-{a} {a} 0.) // NorthWest = 5
(-{a} -{a} 0.) // SouthWest = 6
( {a} -{a} 0.) // SouthEst = 7

( {l2} {l2} {h}) // NorthEstsqt = 8
(-{l2} {l2} {h}) // NorthWestsqt = 9
(-{l2} -{l2} {h}) // SouthWestsqt = 10
( {l2} -{l2} {h}) // SouthEstsqt = 11

( {a} {a} {h}) // NorthEstsqt = 12
(-{a} {a} {h}) // NorthWestsqt = 13
(-{a} -{a} {h}) // SouthWestsqt = 14
( {a} -{a} {h}) // SouthEstsqt = 15
);

'''.format(l2=d['l2'], h=d['height'], a=d['r']*cos(pi/4), k=d['converttometer'])


txt += '''
blocks
(
//square block
hex (
9 8 11 10
1 0 3 2
)
({divx} {divy} {divz})
simpleGrading (1 1 {zgrading})

//slice1
hex (
13 12 8 9
5 4 0 1
)
({divx} {divr} {divz})
simpleGrading (1 1 {zgrading})

//slice2
hex (
9 10 14 13
1 2 6 5
)
({divx} {divr} {divz})
simpleGrading (1 1 {zgrading})

//slice3
hex (
10 11 15 14
```

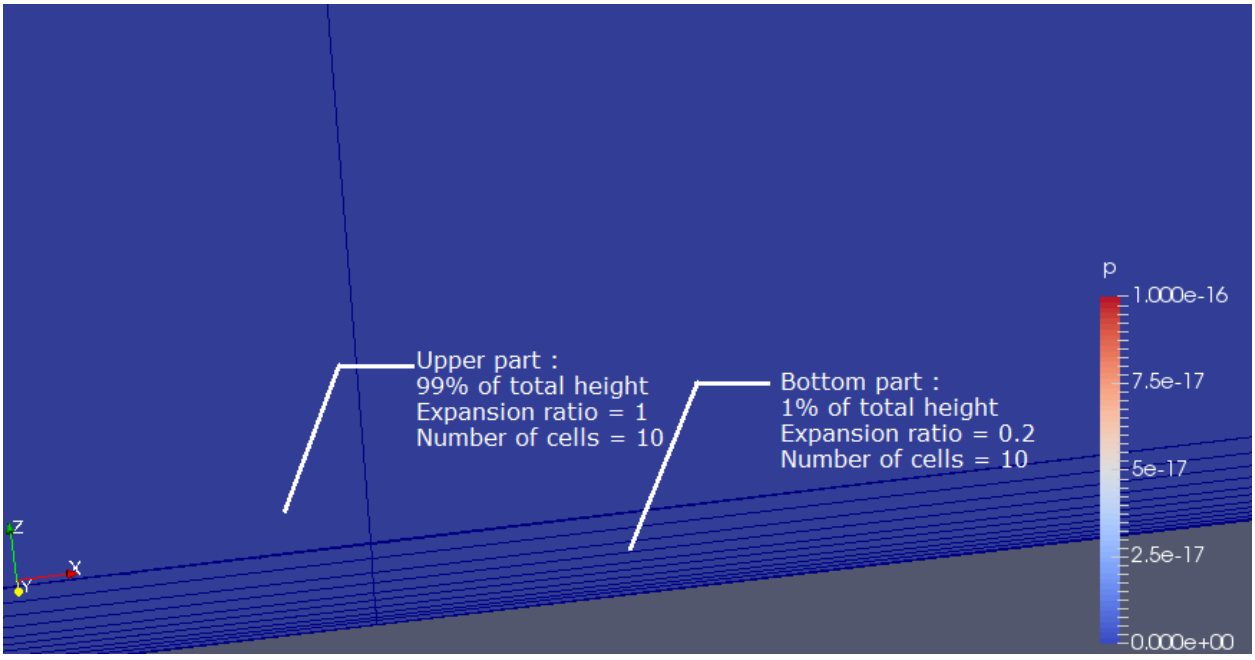
```
2 3 7 6
)
({divx} {divr} {divz})
simpleGrading (1 1 {zgrading})

//slice4
hex (
11 8 12 15
3 0 4 7
)
({divx} {divr} {divz})
simpleGrading (1 1 {zgrading})

);
'''.format(divx=d['divx'], divy=d['divx'], divz=d['divz'], divr=d['divr'],
        zgrading=d['zgrading'])


txt += '''
//create the quarter circles
edges
(
arc 4 5 (0.0 {r} 0.0)
arc 5 6 (-{r} 0.0 0.0)
arc 6 7 (0.0 -{r} 0.0)
arc 7 4 ({r} 0.0 0.0)

arc 12 13 (0.0 {r} {h})
arc 13 14 (-{r} 0.0 {h})
arc 14 15 (0.0 -{r} {h})
arc 15 12 ({r} 0.0 {h})

);'''.format(r=d['r'], h=d['height'])

txt += '''
patches
(

wall surface
(
(8 11 10 9)
(8 12 15 11)
(12 8 9 13)
(9 10 14 13)
(11 15 14 10)
)

wall bottom
(
(0 3 2 1)
(0 4 7 3)
(4 0 1 5)
(1 2 6 5)
(3 7 6 2)
)

wall walls
(
```

```
(5 4 12 13)
(5 13 14 6)
(6 14 15 7)
(7 15 12 4)
)

);'''

print(txt)

path = os.path.join('system', 'blockMeshDict')
print('save new blockMeshDict in path\n' + path)

with open(path, 'w') as f:
    f.write(txt)
```

4.3 Spinup Strat Piece Mesh

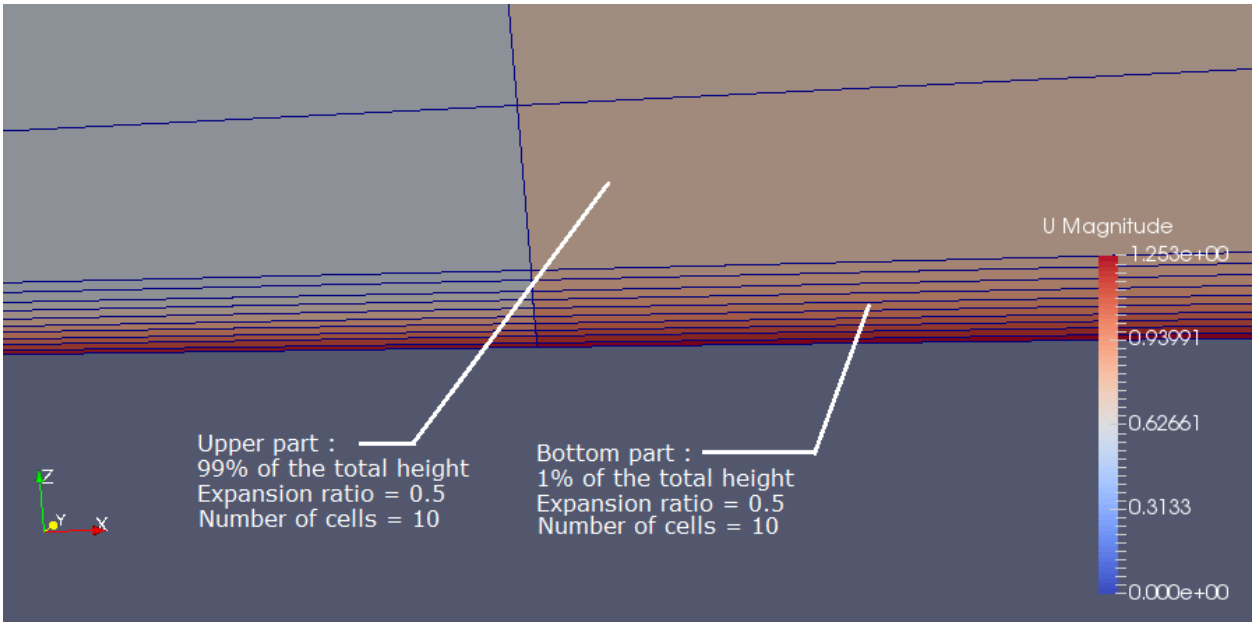
In this simulation the calculus domain has been restricted to a small "piece of cake". The idea behind the restriction was to have a fast runningsimulation that will allow us to test various mesh configurations and choose the one that best fitted our model.

We then wrote a Python mesh generating script for this particular tutorial. In order to have at least 10 cells calculated in the boundary layer and to have very small cells near the bottom wall, we chose to split the z axis into two zones. One zone near the wall that has an expansion ratio of 0.2 (see annex for details and definitions), and an upper zone that has a uniform discretisation. We made sure that the center of the first cell is in the sub-viscous layer. We also wanted to have a simulation that runs as fast as possible, so we reduced the width of the "piece of cake" to 3 mm (3 cells, width = 1 mm) in order to have something that is almost 2D like.



We tried to run a simulation with water using this calculus domain and this mesh, but it did not work, since it completely diverged. The fact that the simulation failed is due to the enormous difference of sizes between the upper and the lower part of the z axis (ratio of 10 000), and the very important aspect ratios on each cell (more than 100).

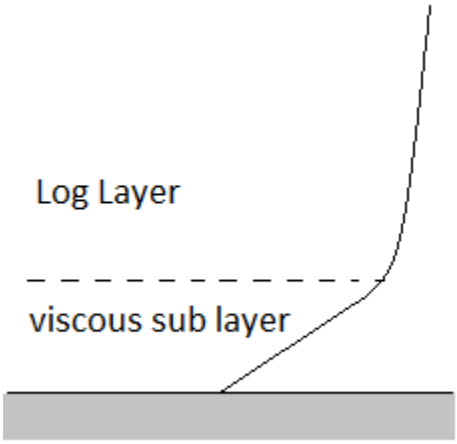
To solve this issue, we decided to make a wider piece, we choose the width of the cell in order to have an aspect ratio of about one for the upper cells (total width = 7.5 cm, 3 cells). We also choose to put a gradient in the upper part of the z axis to have less differences in size between the last cell of the bottom part and the first cell of the upper part.



CHAPTER 5

Boundary Layer

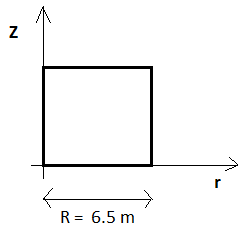
One of the most important things in a fluid mechanics simulation is how the boundary layer is taken into account in the simulation.



In order to have a well re-transcribed boundary layer, we have to ensure that there are enough points in the boundary layer.And, since the solution obtained depends on the location of the first cell, we have to make sure that the center of the first cell is well-located meaning that it has the adequate y+. For the Spinupstrat Case with K-Omega SST :

- **if there are no wall functions** [the center of the first cell has to be located in the sub-viscuous layer] $y+ \in [1; 30]$
- **if there are wall functions** [the center of the first cell has to be located in the log layer] $y+ \in [30; 100]$

The initial mesh available on the various tutorials was a uniform one, meaning that the discretization of the x, y and z axis are uniform (no gradient). The size of each cell on the z axis was 2.5 cm. We displayed the value of y+ in the center of the first cell.



We consider the solid rotation so we can display the velocity in a section of the cylinder (as showed on the figure upward) :
$U = \frac{\omega R}{2}$

We can then estimate the value of u_* using the following approximation : $u_* \simeq \frac{U}{10}$

Finally, y+ is defined as : $y = y_+ \frac{\nu}{u_*}$

For a simulation using water as the fluid, we found a value of approximately y+ = 800 for the center of the first cell (uniform mesh). This value is way too high which means that all the gradients are poorly estimated/calculated. To correct the problem, we thought of improving the basic original mesh by refining it near the bottom wall. We did not directly change the complete mesh of one of the available simulations. We thought it better to create a whole new simulation that would be more adapted to deal with meshrelated issues. A whole new simulation called spin-up strat piece was created.

CHAPTER 6

Ense3 project

6.1 Basic commands

Here you can read all the basic commands we used. PLEASE, use titles in order to precise why these basic commands are used.

6.1.1 Shortcuts in Linux

- Ctrl + Alt + t : loads the terminal
- Alt +Tab : “change the current window”
- Ctrl + Shift + t : opens a new consol in the same directory as the original one
- ls -a : Displays all the files even the hidden files
- Ctrl +C : Kills the process in the terminal

6.1.2 Basic commands in Linux

- cp -r name1 name2 : copies and renames the new file
- cp -r name1 file1 : copies a file into a folder
- mv -r position1 position2 : moves file from one directory to another
- rm -r name1 : deletes a file/folder
- cat name1 : show the written content of the file
- pwd : shows current directory

6.1.3 Helful debugging tricks

Access

- chmod u+x name1 : if permission is denied for a command (life saver !)
- chmod ug+ug name1 : if permission is denied to open a file
- ls -l : to see the accesses (r for read, w for write)

Simulations

- oarstat -u : if no simulation is launched it will do nothing
- oardel jobnumber : deletes a job
- ls processor* : to see the advancement of the simulation

6.1.4 First commands

First we have to make sure that whenever we open the terminal, python and openFOAM are loaded. To do this, we need to change the .bashrc file in the home directory.

We start first by editing the .bashrc file with gedit:

```
gedit .bashrc &
```

Then we write at the end of the file the two following lines:

```
module load openfoam/4.1
module load python
```

6.1.5 IPython

Launch and use IPython

First you to load python (if it is not already done):: module load python

Then to load the environment you use the command:: ipython

To run a script, you use the command:: run filename.py

NB: you can use the command ls inside the ipython consol to see the folders you can access.

To execute a bash command in the terminal without exiting Ipython you use !:

```
!cat README.txt
```

Plot results

Think about “hg pull” and “hg up” in the pyof directory tu make sure that all the files are updated

Pour tracer les profils de pression (par exemple) avec matplotlib, il faut modifier et executer le script : plot _ crossection.py dans les sous dossiers de ../tutorials .

Comments on the file “plot _ crosssection.py” :

You MUST execute the commands written in pink at the beginning (ipython –matplotlib for example) before running the file.

When you launch the following command : reconstructPar -time X (Where X is the time at which we want to see the results) Python creates a folder named X in the subtutorial directory /grav _ which contains the reconstructed results at time. Python will use this folder to plot the results at time X.

xm, ym, zm = coordonates of the center of the mesh's cells.

ym _ unique enables to keep only one exemplar of xm, ym, zm (to avoid repetitions) ym _ unique.sort() orders the values dy gives the maximum difference between two centers of cells in the mesh

y0 = 0. gives the plan where we want to plot [[it is not sure that there are points of the mesh here]]

y0 = ym _ unique[abs(ym _unique - y0).argmin()] gives the points of the mesh which are the nearest y0 = 0.

cond = (y0 - dy < ym) & (ym < y0 + dy) gives an interval among the y direction to get several points because the mesh is not cartesian (cylindral regions in the mesh)

key = 'grandeur' to change what we want to visualize

timename = 'X' corresponds to the folder where the the results at time X have been stored (the folder is located in the processor folder in the subdirectories of ../tutorials).

6.1.6 Use the cluster for calculation

For heavy calculation, we have to use the cluster. To do this, we have to log to the central computer ????? through the platform X2GO. We first have to launch X2GO and double-clic on the new session window. The new session window will require some informations which are the following :

- Host : nrj1sv223
- indentifiant : your loging
- session : XFCE

Don't forget to change the bash.rc on the new window (X2GO). You will have to follow the steps presented in the fith paragraphe to launch the simulation.

6.1.7 How to run a simulation

- Step 1 : compiling the code

The first step when we want to run the simulation is to compile the code. To do this we have to go the the directory containing the code of the solver and execute the command:

```
make
```

For more help go to . . . /lastname/coriofoam/README.srt

- Step 2 : launching the simulation

You just have to follow the instructions contained in the README.srt (. . . /lastname/coriofoam/tutorials/nameoftutorial/README.srt)file contained in the simulation explaining the different steps to launch the simulation. If you want to make a simulation in the cluster, the .sh files already contain the commands to launch the calculus with oarsub.

If you want to launch the simulation using more than 4 cores, you have to change the the decomposeParDict file in the ../tutorials/grav../system. You open the file with gedit, then you change the field:

numberOfSubdomains 'X'; where X is the number of cores we want to use.

- Step 3 :

Once the calculation has been launched, some files and directories will appear in the /tutorials/gravitational_..... The processors directories

0_init : contains the files ccx ccy ccz which contain the "coordinates" of the centers of the mesh

cells.

6.1.8 How to share your files with the others

First mandatory steps

First of all you have to make an hg status to see the differences between the common deposit and your own deposit. To make sur that you have the latest version of the documents on the common deposit, you have to make an hg pull. Then you have to make an hg update to update your files and change them to the latest version of the common deposit. You finish by using the command hg status, to make everything went all good.

It is important to do it in this order even if you only want to push your own file, because it avoids us to use the command hg merge.

You have to update to the latest version the folder coriofoam and pyof !

To push your files in the common deposit

You have to use the commands hg commit, hg push.

Be careful when you use the command hg push. You have to check that there are note unnecessary

files that will be pushed to the common depository like simulation outputs (appears like a long red

list), if so make sur to delete them before using hg push.

To summarise the steps :

1. hg status
2. hg pull
3. hg update
4. hg status

   (4,5. hg add filename -> to add a file in the depository that doesn't already exist)
5. hg commit -m "file XXX added" (fait une photo/ copie de votre dossier)
6. hg push
7. hg satus

Useful commands

In order to see the difference between your file and the common depository file you can use the command:

```
hg diff
```

To undo what you just did:

```
hg revert
```

To see the changes in a file:

```
hg heads
```

En cas de problème:

```
hg merge
hg resolve
```

6.1.9 How to use firefox

To open your file in the website you have to enter the following command in the terminal :

firefox _build/html/index.html &

6.1.10 Other commands

To kill a process:

```
pkill firefox
```

6.2 Session report

6.2.1 Session 1 : 08/02/2017

Today morning, we have learnt to manipulate python and write into the terminal with python language (thanks to command ipython). In order to do this, we need to modify the bash.rc file in adding at the end : *module load openfoam/4.1* et "module load python"

We ran a tutorial code on the local computer, the PC crashed because we needed more processor. That's why we moved to the cluster for computation. The tutorial code ran with success and we saw that the results of the coriolis modelisation could be improved (example : gravitational instability), especially the mesh. We saw many complicated things that have to be precised. That's why we defined our objectives this morning for today and the next sessions. We have to:

- write documentation for us, of the basics commands and paths (tree structure) in order to work quickly (sara)
- write documentation on the general functioning of the tutorial code (understand what the solver did to resolve the problem, in order to see what can be improved in the code) We didn't read it today, the next time we'll start.
- writing python script in order to modify the mesh (finished, it worked, detailed in the mesh documentation) (Nicolas)
- create an internet site in order to put in common what we did and make a report for each session (julien)
- exploration of the tutorial commands and entries definition of the initial data, time control We started to explain the openfoam commands (sav et emere)

for the next sessions : try to understand deeply the codes and the solver in order to start modifying it.

6.2.2 Session 2 : 15/02/2017

Today morning, Savanah finished her work on the tutorials After she heldped nicolas on the mesh python program emre and I Tried to understand the solver code , very difficult, we need explanations on the Pimple method next session. We have to understand physical equations and write documentation on it

We had explanations from joel the platform responsible he will give us experience results next time. We had a course on the boundary layer.

For the next session : run calculation for the spinup case for different viscosity and understand the code

6.2.3 Session 3 : 01/03/2017

We tried to launch simulation, but we didn't succeed, we had to move to greener for 10 o'clock The objectives are reported to the next session.

6.2.4 Session 4 : 08/03/2017

Today, we ran the spinup stratification code and we obtained some results, we need more results and applications in order to analyse it. We'll see the analysis for the next session. We took some picture of our results for our presentation. (Savannah, Nicolas)

Emre create a precise GANTT planning

Sara and Julien worked on the solver and tried to understand the PIMPLE method, and wrote the mathematical equations, they did it for U.

6.2.5 Session 5 : 15/03/2017

Savannah and Nicolas treated the previous simulation results from the last session. The results are in the file "data" and the simulations are in the file "Run". They used paraview in order to see the results. Savannah and nicolas launched an other simulation with a different viscosity.

Sara, Emre and Julien worked on the equations in the solver. Julien Chauchat gave us explanations on the SIMPLE algorithm and gave us the main equations that are solved by the algorithm. Pierre told us we have to calculate the profile of the free surface in the case of a solid rotation and compare to experimental results for different times. We have to do this in two cases : startified flow and homogenous flow. We also have to make an estimation of the time it takes to see a solid rotation appear in the experiment. This time is a caracteritic time.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search