FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Simulation of Flow in a Conical Helix Tube

João Ranita

Master in Chemical Engineering

Supervisor: Domingos Azevedo Gonçalves Barbosa (Professor Auxiliar)

Co-Supervisor: Paulo Aloísio Edmond Reis da Silva Augusto (Investigador)

11^{th} July, 2011

Abstract

This work presents numerical simulation results, for incompressible isothermal Newtonian single phase liquid flow in a tube geometry not found in the literature, a conical helix tube of circular section, with constant pitch but linear radius variation. The simulation results will aid in the geometry optimization of the new separation device for Project MagPro2Life.

This work also explored the application free and open-source software to obtain the simulation results. The path to obtain the simulations is presented, including the steps taken to solve simulation convergence problems. The OpenFOAM software is described in some detail.

The results obtained are physically sane and are in agreement with some of the findings in the literature. The application of OpenFOAM was positive, even though issues were risen regarding meshing. Pre-processing using Salome-Meca was also a positive experience, specially considering that all the geometry was automated with a script. Post-processing with ParaView proved to be difficult to use and with numerous crashes.

Keywords: CFD, OpenFOAM, Conical Helix

Resumo

Este trabalho apresenta resultados de simulações de fluxo de um fluido Newtoniano incompressível, isotérmico, num tubo de secção circular, mas disposto como uma hélice cónica com variação linear do raio. O fluxo nesta geometria não se encontra estudado na literatura. Os resultados da simulação serão aplicados na optimização de um novo dispositivo de separação para o Projecto MagPro2Life.

Este trabalho também explorou a aplicação de software gratuito e de código aberto para obter os resultados de simulação. É apresentado todo o trajecto desenvolvido para obter as simulações, incluindo todos os passos para resolver os problemas de convergência da simulação. O software OpenFOAM é descrito com algum detalhe.

As simulações obtidas são fisicamente sãs e estão de acordo com alguns resultados da literatura. A aplicação do OpenFOAM foi uma experiência positiva, apesar dos problemas na construção da malha. O pré-processamento com usando o Salome-Meca foi também uma experiência positiva, especialmente considerando que foi possível gerar automaticamente a geometria através de um script. O pós-processamento com o ParaView provou ser difícil e com numerosas falhas.

Palavras Chave: CFD, OpenFOAM, Hélice cónica.

Acknowledgements

I would like to thank Dr. Paulo Augusto for the opportunity of work in a demanding Project like MagPro2Life. In my life, so far, this is one of the few occasions that challenged my ability to learn, both in speed and quantity.

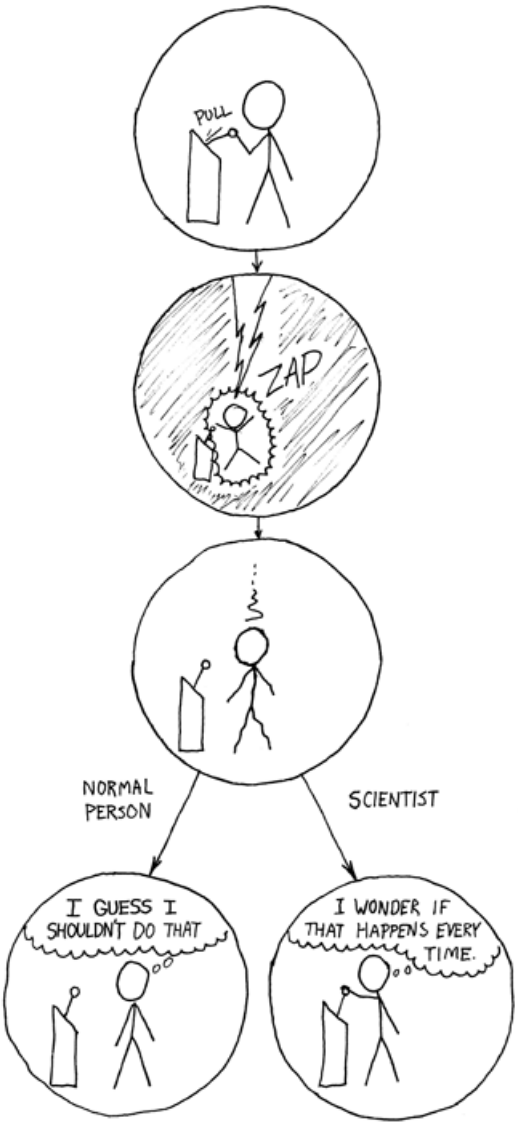
I would like to thank Dr. Antonio Martins, for directing me to OpenFOAM, and Prof. Domingos Barbosa for supervising me.

A thanks is also due to the on-line communities, many of my questions and doubts were answered because someone else had already asked the same questions I had.

A very special thank you is kept in my heart for my wife.

"Most people say that is it is the intellect which makes a great scientist. They are wrong: it is character."

Albert Einstein



The Difference
(*http://xkcd.org/242/*)

Contents

List of Figures

LIST OF FIGURES

Nomenclature

a	Tube radius
d	Tube Diameter
D	Helix Diameter
De	Dean Number
F_s	Straight tube Fanning friction factor
F_c	Coiled tube Fanning friction factor
He	Helical number
K	Dean dynamic similarity factor
L	Tube length
p	pitch of helix
Re	Reynolds number
Re_c	Critical Reynolds number
U	Fluid velocity
ρ	Fluid density
μ	Fluid viscosity
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
LES	Large Eddy Simulation
RAS	Reynolds Averaged Simulation

Chapter 1

Introduction

Helically coiled tubes have found widespread use in heat exchangers, boilers, stills and many more devices. The main feature of this kind of tubes, is that the curvature of the tube produces a centrifugal force in the fluid, that will induce a secondary flow. Approaches to resolving the flow of curved tubes have been tried analytically and numerically.

The present work will focus on the simulation of flow in a conical helix tube with constant pitch and linear radius variation. In the literature, no work was found regarding simulations for this kind of helix. The complexity of the flow in helical tubes is a challenge even with the advent of CFD. Many problems persist that were not yet thoroughly studied, like the case of helical tubes with variable tube section, bifurcations, etc. Both from an analytical and a numerical perspective, the flow in helical tubes, still presents many challenges. In this work we obtained the flow profile for the case of a conical helix pipe. To accomplish this we applied the OpenFOAM software to solve numerically the Navier-Stokes equations, and used other software packages for pre- and post-processing.

1.1 Context

The present work is framed within the European project MagPro2Life[1], regarding further optimization of a patented magnetic nanoparticles separation/classification device.

The interesting physical-chemical characteristics of nanoparticles (1 nm <= diameter <= 100 nm) are being explored in separation processes. Their application in bioseparation, where the products have high value (pharmaceutical molecules, proteins), is of great interest.

The nanoparticles for separation processes must have some kind of affinity to attach to the target molecules, which is achieved through functionalization. After attaching to their targets they must be separated from the bulk and, if the nanoparticles exhibit strong magnetic (superparamagnetic) properties, this separation may be easier to accomplish.

[1] http://www.magpro2life.eu/

Project MagPro2Life aims at one end to develop large-scale functionalised magnetic nanoparticles manufacturing processes and, on the other end, to build pilot-scale devices for efficient nanoparticle separation processes in pharmaceutical, feed and food industries.

1.2 Goals

The aim of the present work is to obtain numerical simulation results, for incompressible isothermal Newtonian single phase liquid flow in a conical helix tube of circular section. The simulation results will aid in the geometry optimization of the new separation device, and also support the design of experiments for testing the device.

Secondary objectives for this work are the acquisition of OpenFOAM development skills, competence in geometry and mesh design softwares, and the ability to display results in ParaView.

1.3 Thesis Structure

This thesis is composed by an Introduction followed by 6 chapters.

Chapter 2 presents the state of the art, referring to some of the most interesting related works.

Chapter 3 is a brief introduction to the geometry of the system, and the fluid flow in helical tubes.

Chapter 4 introduces openFOAM, the software used to perform the numerical simulations for this work and details how to apply OpenFOAM to solve this particular problem.

Chapter 5 presents and discusses the results of the simulations.

Chapter 6 presents the conclusions and the future work.

Annex A contains important simulation configuration files.

Chapter 2

State of The Art

Throughout this chapter references will be made to works regarding flow in helical tubes. Most of these are related to heat transfer, due to the typical applications in heat exchangers. It is also noticeable that most of the work has been done in the last 50 years. A review of published relevant articles that involve single phase liquid flow in helical tubes will be presented. The review will start with the early studies on curved pipes, these early studies introduced essential basic concepts and correlations.

2.1 Flow in Curved Tubes

One of the first researchers to study the problem of flow in curved tubes was Eustice [1, 2], he experimented with curved tubes and observed that the pressure drop was higher when compared to a straight tube of equal length. Dean in the 1920's was the first to reach an analytical solution for steady flow in coiled tubes and published his findings in the Philosophical Magazine [3, 4]. In his theoretical work he found that, for low flow rates, the pressure drop would be a function of K, a function of the Reynolds number Re, the tube diameter d and the curvature diameter D (2.1). This parameter K led to the definition of the Dean De dimensionless number [Eq. (2.2)]. He also introduced the Fanning Friction Factor ratio between curved and straight tubes, this is usually used for laminar flow.

$$K = 2\frac{d}{D}\mathrm{Re}^2 \tag{2.1}$$

$$\mathrm{De} = \mathrm{Re}\sqrt{\frac{d}{D}} \tag{2.2}$$

In 1929 White experimented with laminar flow in circular curved tubes [5]. He realized that the transition to turbulent flow may occur at higher Reynolds (compared to straight tubes), and

determined and empirical correlation to describe the pressure drop. In the same year Taylor experimentally determined that the flow transitions did in fact occur at much higher Reynolds numbers [6]. In his experiments, for $\frac{d}{D} = 0.0313$ the laminar flow would hold until $\mathrm{Re} = 5010$.

The next major input in the field was by Ito [7], who first proposed the Critical Reynolds Number Re_c [Eq. (2.3)]. This kind of flow presents a later transition from laminar to turbulent, so the Reynolds number was no longer adequate in this flow, the Critical Reynolds number addresses this issue.

$$\mathrm{Re}_c = 20000 \left(\frac{d}{D}\right)^{0.32} \tag{2.3}$$

In 2006 a good review article, biased towards heat transfer, was published by Naphon and Wongwise [8]. They present works for both single-phase and two-phase flows.

2.2 Flow in Helical Tubes

In 1979 Misha [9] completed an exhaustive work measuring the pressure drop, in the laminar and turbulent region, for Newtonian fluids flowing through 60 different helically coiled tubes (constant pitch, curvature and torsion). He obtained empirical correlations for the pressure drop for laminar flow and $1 < \mathrm{He} < 3000$:

$$\frac{f_c}{f_s} = 1 + 0.033\left[\log_{10}\mathrm{He}\right]^{0.32}, \quad \text{where } \mathrm{He} = \mathrm{Re}\sqrt{\frac{\frac{d}{D}}{1+\left(\frac{p}{\pi D}\right)^2}} \tag{2.4}$$

For turbulent flow, $4500 < \mathrm{Re} < 10^5$, $0 < p/D < 25.4$, and $6.7 < D/d < 346$ Misha obtained:

$$f_c = 0.0791\mathrm{Re}^{-1/4} + 0.0075\sqrt{\frac{d}{D}} \tag{2.5}$$

Patankar studied the flow in helically wound tubes [10], and he was one of the first to address numerically the Navier-Stokes equations [11].

The Pressure Implicit Split Operator algorithm (PISO), for the solution of the Navier-Stokes equations, was proposed by Issa [12].

Since then many worked on the problem and, a good compilation of correlations in helical coil tubes was presented to us by Ali [13]. From experimental data he obtained a set of empirical correlations to better fit the pressure drop in such flows.

2.3 Flow in Conical Helix

A very interesting work by Gammack and Hydon [15], achieved analytical solutions for steady state flow in tubes where torsion and curvature were not uniform. They followed the key work of

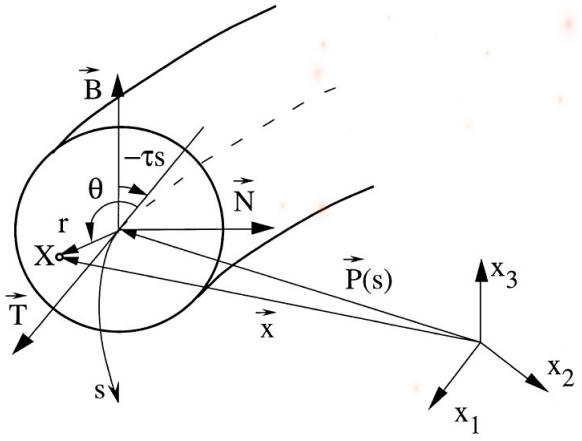


Figure 2.1: Orthogonal helical coordinate system[14].

Germano [14] in changing the system reference (2.1). They obtained analytical solutions for different helix geometries 2.2. To solve the unsteady flow conditions they had to solve the equations numerically.

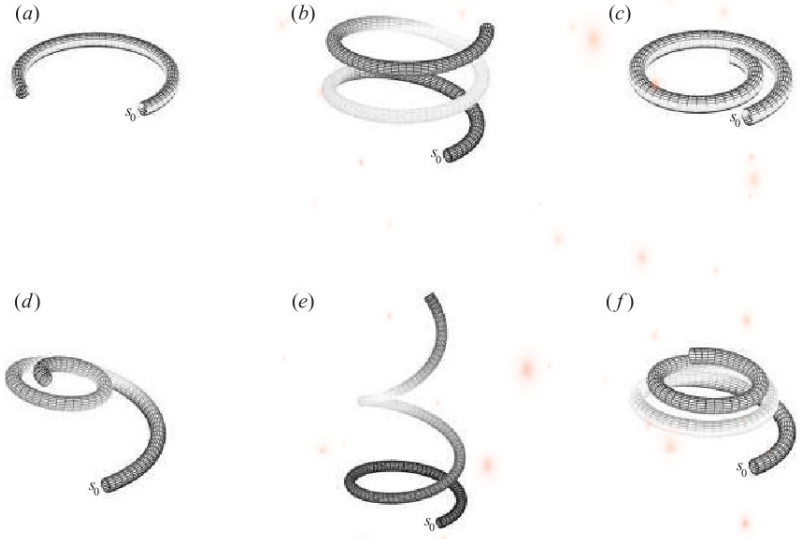


Figure 2.2: Tube shapes (a) torus, (b) helix, (c) spiral, (d) spiral with uniform torsion, (e) stretched helix, (f) pipe with curvature inversely proportional to torsion[15].

The results (2.3) show a difference between toroidal and helical tubes. As expected, for toroidal tube the secondary flow is horizontal. In the helical tube (the inlet was at the bottom) the secondary flow is rotated, this is due to the vertical displacement component of the fluid flow.

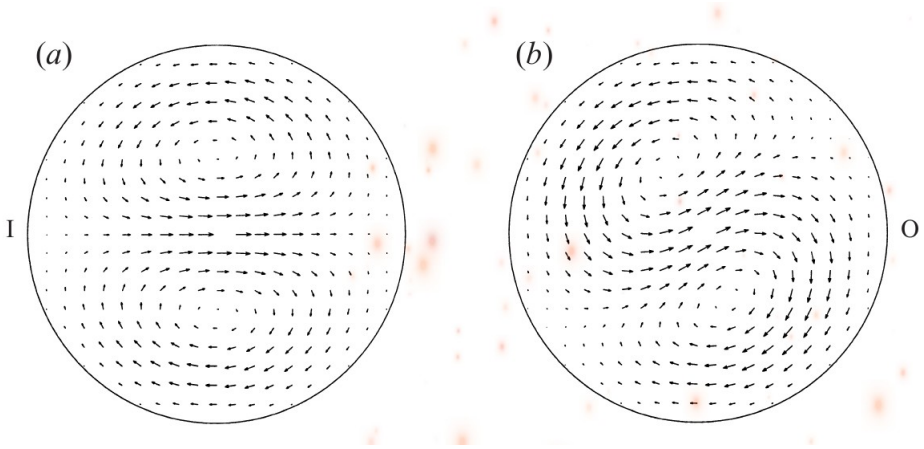


Figure 2.3: Secondary velocity vector field plots: (a) Dean flow, *Curvature* = 0.09, *Torsion* = 0.0; (b) helical flow, *Curvature* = 0.09, *Torsion* = 0.4. I and O are the innermost and outermost parts of the pipe. wall[15]

Chapter 3

Conical helix geometry and flow

This chapter will begin with a section presenting the conical helix geometry equations and characteristic parameters. These equations will later be used to generate the geometry for the simulation mesh. The second section of this chapter will present the problem of flow in helical tubes, and the reason why

3.1 Geometry

A coil spring (3.1) is typically shaped has an helix. An helix is defined by the direction (clockwise or counter-clockwise), pitch, curvature and torsion. Pitch is the distance between consecutive turns of the helix, measured parallel to the central axis of the helix. Curvature is a measure for the bending, closely related to the radius of the coil. Torsion is a measure for the twist, closely related to the pitch of the coil.

For clockwise helix radius a, with constant pitch, constant curvature and constant torsion, the formula in Cartesian coordinates is:

$$\begin{aligned} x(t) &= a * cos(t) \\ y(t) &= a * sin(t) \\ z(t) &= b * t \\ t &\in [0, N] \end{aligned} \tag{3.1}$$

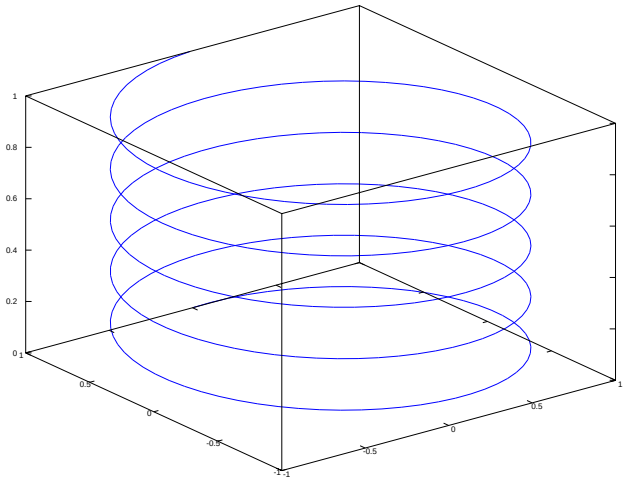


Figure 3.1: Helix with constant pitch, curvature and torsion

The mathematical definitions of helix pitch, curvature and torsion are:

$$\begin{aligned} pitch\, p &= \frac{2}{\pi b} \\ curvature\, c &= \frac{|a|}{a^2+b^2} \\ torsion\, \tau &= \frac{b}{a^2+b^2} \\ length L &= \sqrt{(\pi DN)^2 + (pN)^2} \end{aligned} \tag{3.2}$$

In this work we will address a conical helix geometry (3.2) . The pitch is constant but radius, curvature and torsion change. This is possible by making the radius a a linear function of the height.

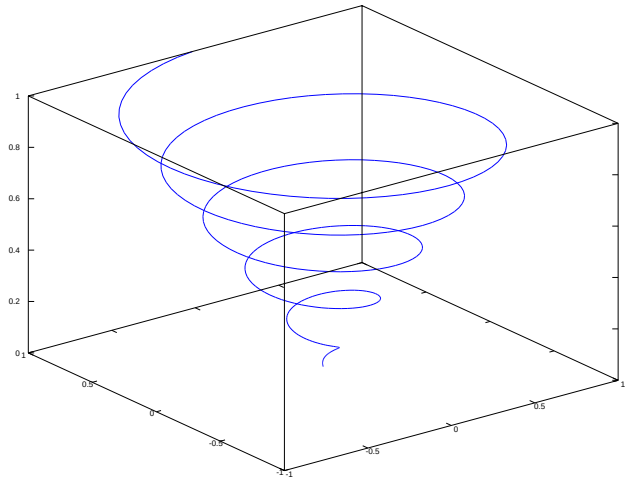


Figure 3.2: Conical helix with constant pitch and linear radius variation.

3.2 Navier-Stokes

In 1822 Claude-Louis Navier and George Gabriel Stokes formulated the Navier-Stokes equations.

Vector Form These are the equations written using compact vector notation.

The continuity equation (conservation of mass):

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \vec{u} = 0 \tag{3.3}$$

The motion equation (conservation of momentum):

$$\rho \frac{D\vec{u}}{Dt} = -\nabla p - \nabla \cdot \tau + \rho \vec{g} \tag{3.4}$$

Shear stress constitutive equation:

$$\tau = -\mu \left(\nabla \vec{u} + \nabla \vec{u}^T - \frac{2}{3} \nabla \cdot \vec{u} \right) \tag{3.5}$$

The simplified motion equation for an incompressible Newtonian fluid with uniform viscosity:

$$\rho \frac{D\vec{u}}{Dt} = -\nabla p + \mu \nabla^2 \vec{u} + \rho \vec{g} \tag{3.6}$$

One of the first to address the numerically the Navier-Stokes equations was Patankar, he also proposed the Semi Implicit Method for Pressure Linked Equations (SIMPLE) [11] and studied the flow in helically wound tubes [10].

The SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) allows for coupling of the Navier-Stokes equations with the use of an iterative algorithm with the following steps:

1. Set the boundary conditions.
2. Solve the discretized momentum equation, get the intermediate velocity field.
3. Compute mass fluxes at cells faces.
4. Solve the pressure equation and apply under-relaxation. (correct non-orthogonality)
5. Correct mass fluxes at cell faces. (correct non-orthogonality)
6. Correct velocities based on the new pressure field.
7. Update the boundary conditions.
8. Repeat until convergence or max iterations.

The method used to solve the simulations in this work was the Pressure Implicit Split Operator algorithm (PISO), proposed by Issa [12].

The PISO (Pressure Implicit with Splitting of Operators) is an efficient method to solve the Navier-Stokes equations in unsteady problems. The PISO algorithm is the following:

1. Set the boundary conditions.
2. Solve the discretized momentum equation, get the intermediate velocity field.
3. Compute mass fluxes at cells faces.
4. Solve the pressure equation, do not apply under-relaxation. (correct non-orthogonality)
5. Correct mass fluxes at cell faces. (correct non-orthogonality)
6. Correct velocities based on the new pressure field.
7. Update the boundary conditions.

8. Return to step 3 (repeat N times)

9. Increase the time step, return to step 1.

There is no under-relaxation. Steps 4 and 5 may be repeated to correct for non-orthogonality.

3.3 Conclusions

The work will take the numerical approach due to the fact that the extension of analytical results, to embrace flow and a mix of nanoparticles with different sizes and shapes would be, not impossible, but too complex to be solved in a short time frame.

Chapter 4

Simulation Setup

In this chapter the OpenFOAM software and the way detail to apply it, to solve this particular problem, will be presented. OpenFOAM is a generic tool for solving problems of Continuum Mechanics, like CFD simulations.

4.1 Hardware

The workstation used to develop and perform the simulations had the following components:

- intel XEON w3680 – 12 processors, 12 MB cache
- 24GB ECC RAM
- Asus P6T7WS-Supercomputer Motherboard
- NVIDIA GTX460 Graphics Card
- 500 GB Hard Drive with 32 MB cache
- Antec P193 Case and CP850 PSU

4.2 Introduction to OpenFOAM, Salome-Meca and ParaView

```
/*---------------------------------*- C++ -*-------------------------*\
| =========                 |                                         |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox |
|  \\    /   O peration     | Version:  1.7.1                         |
|   \\  /    A nd           | Web:      www.OpenFOAM.com              |
|    \\/     M anipulation  |                                         |
\*-----------------------------------------------------------------*/
```

From the OpenFOAM web site[1]:

> The OpenFOAM® (Open Field Operation and Manipulation) CFD Toolbox is a free, open source CFD software package produced by OpenCFD Ltd. It has a large user base across most areas of engineering and science, from both commercial and academic organisations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetism.

OpenFOAM:

- The most flexible
- Powerful scripting capabilities for automated processing
- Fast
- Reliable results
- State of the art: solvers, numerical methods,meshing algorithms
- Open Source (as opposed to "black box")
- Good support community
- Readily available and much cheaper than the commercial alternatives

Among the companies that use OpenFOAM are:
ABB Corporate Research, Audi, Airbus, Bayer, CD-adapco group, Danone, Daimler, Hitachi, Mitsubishi, Obayashi, SKF, Shell, Toyota, Tokyo Gas, Volkswagen, Electrolux, Fluid Technology, Icon-CG, The Technology Partnership plc, Imperial College London, Kings College London, Chalmers University, University of Exeter, University of Guelph, Hirosaki University, Tokyo Institute of Technology.

Compared to other solutions OpenFOAM has its disadvantages:

- Long and steep learning curve;
- Not integrated into a package;
- A good knowledge of C++ is required.

OpenFOAM is composed of very efficient parts, and is an excellent example of high quality C++ code. These parts (or modules) include the solvers, i.e. code tailored to simulate specific problems;

the libraries, these are the basis for the solvers. There are libraries for physical constants, algorithms, turbulence models, ...;

[1] http://www.openfoam.com/

the utilities, these are small programs that accomplish pre- and post-processing tasks, regarding pre-processing these are usually related to mesh creation/conversion, the post-processing utilities relate to data manipulation and visualization;

OpenFOAM has more than 80 solvers and 170 utilities, and is able to simulate a wide range of problems.

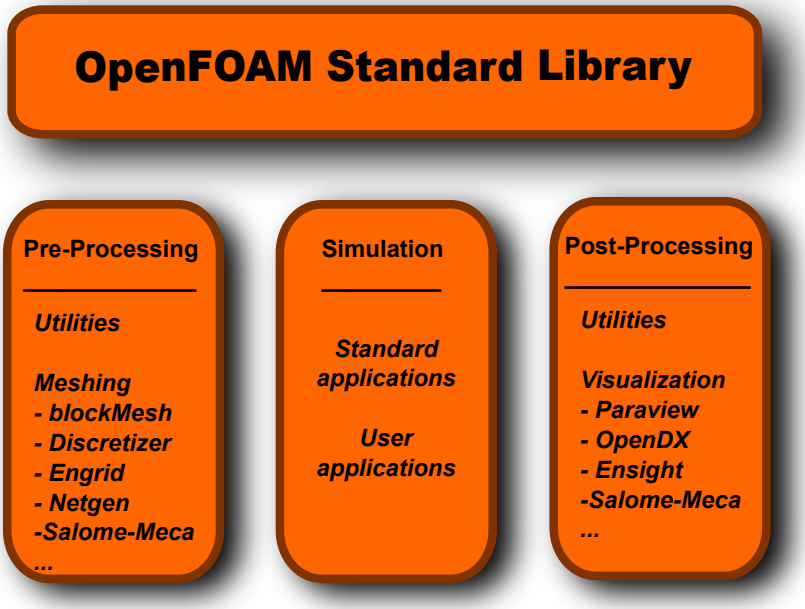


Figure 4.1: OpenFOAM Structure

In the following section the role of each part of OpenFOAM will be explained in grater detail.

4.3 Applying OpenFOAM

Assuming that OpenFOAM is correctly installed and working (there are numerous references on how to do it [web]), the common approach to apply the software will be to adapt an existing *case*.

The *case* is just an existing piece of code that solves a specific problem, it includes the complete definition of the system geometry, boundary conditions, initial conditions, algorithms ...

Many problems may be solved by adapting an existing *case*, so one simply has to copy a *case* similar to the system we pretend to simulate (usually from the tutorials directory of OpenFOAM); change the directory name, run the *case* to check if everything is OK; change the mesh to the one of our problem; adjust boundary and initial conditions; adjust the numerical method parameters until it converges; and finally visualize the results to check if the solution is valid. For simple/common problems this approach achieves good results.

An OpenFOAM *case* has the following directory structure:

- 0 - The initial values and boundary conditions for the simulation;

- constant - Where mesh, turbulence properties, transport properties and other informations that define the system are kept;

- system - Where the information regarding numerical methods and their set-up is stored.

4.3.1 Pre-Processing

Pre-Processing involves all the steps prior to the simulation, in this work the steps to build the mesh, i.e. the set of points that constitute volume elements for our system. In OpenFOAM the mesh is either constructed with the blockMesh utility, or constructed with some other software and later converted for OpenFOAM (engrid, gmsh, fluent, ...).

4.3.1.1 BlockMesh Utility

The blockMesh [16] utility is a simple but basic tool. It creates good quality meshes but requires the user to supply all the points and edges of the geometry faces. blockMesh takes the information of the geometry and generates the mesh for the volume.

The file that defines the geometry is called blockMeshDict, inside constant->polymesh directory. The number of volume elements in each direction and their shape are also define on this file. Defaults to hexahedral shape for volume elements, but other shapes are available. This utility is capable of processing very complex geometries very fast, but to obtain a high quality mesh for a conical helix, one way would involve transforming the circular section of the tube into a icosagon (20 side polygon).

Meshing with blockMesh gives the user a small grain control of the process, but if the geometry is very complex, this can be difficult to achieve. BlockMesh turns meshing into a form of art where each user has a different solution to generate a mesh closer to the optimal one.

4.3.1.2 Salome-Meca

For simple shapes like straight tubes, square ducts, boxes, ... blockMesh is perfect. But for a conical helix a CAD like software would be more appropriate.

Salome-Meca[2] is a open-source CAD software for meshing and simulation. Here drawing a conical helix possible with the following steps: create the path for the helix, create the face (a circle) at the top and extrude it along the path. All this is achieved in a graphical interface where you immediately visualize the results.

One other good point in using Salome-Meca is that all graphical interface commands have counterparts in python programming language commands. Instead of repeating the same steps each time the geometry changes, this can be achieved just by changing the values (pitch, curvature, tube diameter, ...) in a python script (figure 4.5.

[2] http://www.salome-platform.org/

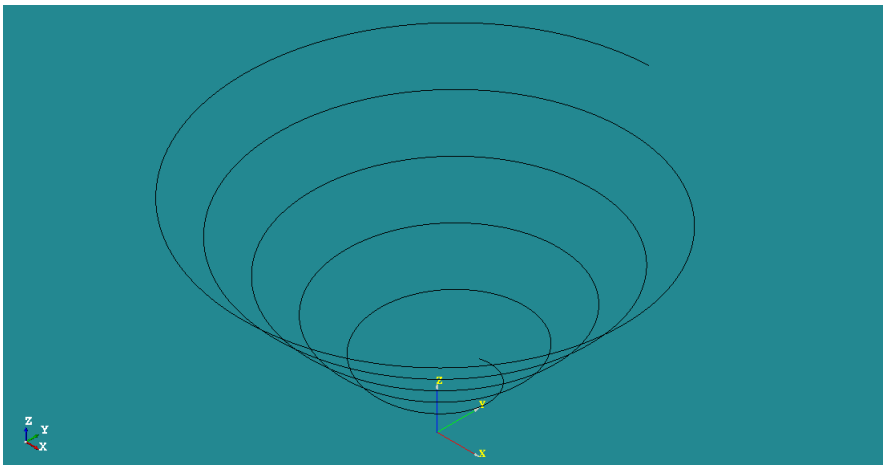


Figure 4.2: Geometry of helix with constant pitch, curvature and torsion.

After creating the geometry the next step is to create the mesh (figure 4.5. Salome, like blockMesh, can use several algorithms to generate the mesh, some examples are: Netgen, GHS3D, GHS3DPRL, BLSURF.

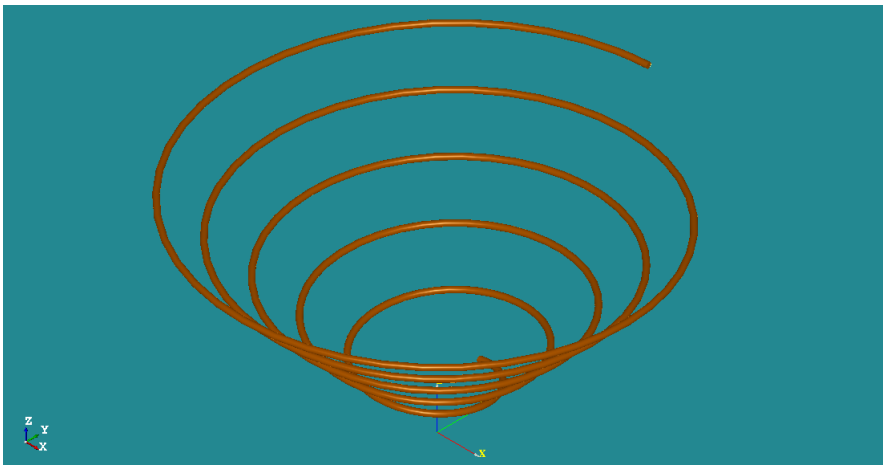


Figure 4.3: Extrusion of helix with constant pitch and not constant curvature and torsion.

To use the mesh created by Salome-Meca in OpenFOAM it must be saved as a UNV file, and aftewords converted with the utility ideasUnvToFoam [17] .

Another important point is that Salome-Meca, like most (mechanical engineering biased) CAD packages, works in mm and OpenFOAM works in m. Before running the simulation just run the the transformPoints utility with the following instruction:

```
transformPoints -scale '(0.001 0.001 0.001)'
```

An optional but advisable additional step is to run the utility checkMesh to verify if the mesh is valid.

The mesh quality is very important for both convergence and speed of the simulation. For the initial set-up of the *case* and for early tests (on a laptop), coarse meshes were used, resulting in non-convergence. Using finer meshes had the effect of increasing the stability of the simulation, but the mesh alone could not achieve the necessary stability of the simulation.

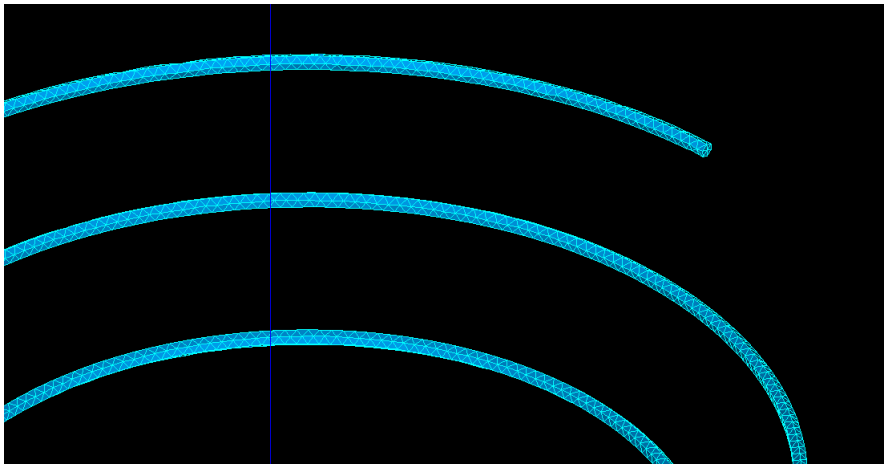


Figure 4.4: Coarse mesh

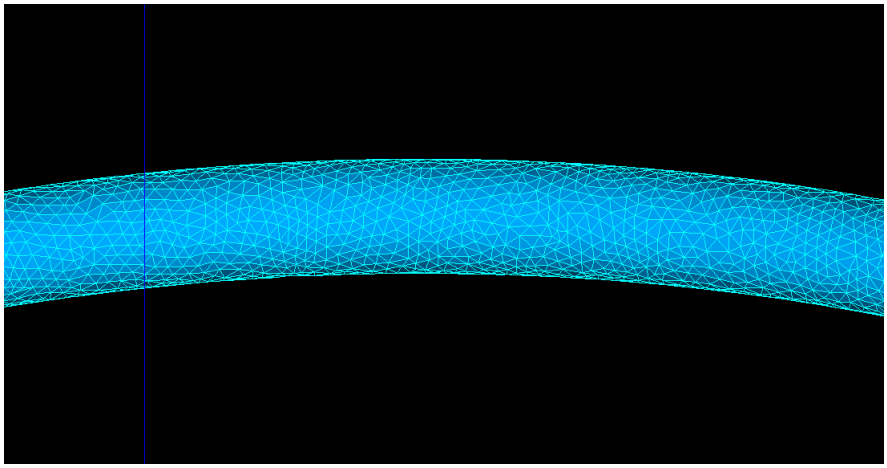


Figure 4.5: Fine mesh

4.3.2 Simulation

4.3.2.1 Solvers

OpenFOAM can be used to solve many engineering problems. For incompressible isothermal single phase Newtonian fluid flow in a stationary tube, OpenFOAM has the following solvers at its disposal.

Transient Solvers:

icoFoam Laminar flow.

pisoFoam Turbulent flow.

pimpleFoam PIMPLE (mixed PISO-SIMPLE) algorithm.

Steady-state Solvers

boundaryFoam 1D turbulent flow.

simpleFoam Turbulent flow.

In OpenFOAM the solver is an application to be executed in the *case* directory. The solver has the high level control regarding the algorithm, i.e. the sequence/steps to solve the Navier-Stokes pressure and velocity equations. These algorithms described in the literature, namely in Ferziger's book [18].

A medium level control of the simulation comes from the choice of numerical methods to solve a specific equation, i.e. the methods to solve the sets of linear equations generated during the simulation runs. These constitute a medium level control of the solver. The available methods for solving sets of linear equations are:

- Preconditioned (bi-)conjugate gradient (available for symetric and asymetric matrices)
- Smoother
- Generalised geometric-algebraic multi-grid
- Diagonal (for explicit systems)

Each of these methods need parameters to operate[16]. Tolerances, both absolute and relative, are obvious parameters, but some need additional data like the choice of preconditioner or smoother (along with their specific parameters).

The informations on regarding this medium level of control are kept in the file fvSolution of the system directory.

OpenFOAM also gives the user a lower control level. This is accomplished through the choice of numerical methods regarding how OpenFOAM approaches the mathematical operators. For example, those familiar with numerical methods know that a derivative could be forward, backward, central, interpolation could be linear, quadratic, etc.

OpenFOAM gives the users control over the numerical methods for the mathematical operators. Derivatives, gradients, laplacian, each have a choice of specific methods to choose from[16].

Opting for the more simple methods (linear methods) speeds up calculations but, depending on the problem, several trial and error tests with different methods may be needed until convergence to a solution is possible.

This lower level numerical methods set-up is stored in the fvSchemes file of the system directory.

4.3.2.2 Turbulence Models

The solvers need more information in order to work. One piece of information they need is the turbulence model. Turbulence models[19] exist to simplify the solution of the time dependent Navier-Stokes equations, posing a tradeoff between obtaining an approximate solution and the maybe unattainable solution for the physical system.

It is important to mention that OpenFOAM also has a Direct numerical simulation (DNS), wich solves the Navier-Stokes equations numerically without any turbulence modelling. This DNS simulation is implemented in the dnsFoam solver, it would yeld solutions closer to the physical reality but at the expense of a large computation time.

OpenFOAM library has turbulence models already implemented for Reynolds-Averaged Simulation (RAS) and Large Eddy Simulation (LES).

Within RAS choice exists between several turbulence models and, for incompressible fluids, among them are: kEpsilon Standard high-Re k-e model kOmega Standard high-Re k-w model kOmegaSST k-w model

In OpenFOAM LES also has a vast choice of turbulence models, many of them are the same as in the RAS, some of them are:

Smagorinsky Smagorinsky model

kOmegaSSTSAS k-w SST scale adaptive simulation (SAS) model

oneEqEddy k equation eddy-viscosity model

For LES the filter and the delta must be selected. Available filters are laplace, simple and anisotropic; the deltas are prandtl, cubic root of cell, and smoothing delta.

4.3.2.3 Initial conditions and Boundary conditions

The key to successful simulations that translate the physical phenomena in a system, is a correct definition of its limits/frontiers. This definition is expressed as a set of initial and boundary conditions. If these conditions are not valid the results will have no physical meaning, as the numerical methods will be departing from false assumptions.

For these simulations it is assumed that initially entry and exit points are at atmospheric pressure and the fluid is resting in the tube.

Throughout the simulations the entry of the tube has a constant velocity profile in the direction of the pipe.

The walls of the pipe will have 0 velocity, this is the no-slip condition.

These informations are stored in the 0 directory, corresponding to the 0 instant of time. As the simulation runs it will write results to time directories, directory 1 will keep the results for the 1 minute instant of the simulation.

4.3.2.4 Simulation Control

Having selected boundary and initial conditions, solver and turbulence model and, in our case Newtonian transport model (there are others for non-Newtonian or multiphase fluids), the next step is adjusting simulation control parameters.

Simulation control has two sides, one is the control of iteration time intervals, start time, end time, write-out results interval.

4.3.3 Post-Processing

The post-Processing involves the tasks regarding visualization of simulations results. OpenFOAM has no visualization abilities so an external software package must be used. OpenFOAM is usually matched with ParaView with the help of the paraFoam utility. This utility initiates ParaView with a configuration that loads openFoam simulation results. There are many other softwares like OpenDX or Ensight, Salome-Meca, and commercial alternatives.

In favour of ParaView is the open-source, and the modular C++ nature. This could be used to extend OpenFOAM with visualization capabilities and, for example, real-time visualization of simulation results. Like Salome-Meca, ParaView has full scripting capabilities through the use of python, so it is possible to automate the process of results visualization.

The downside of ParaView (or paraFoam) is the steep learning curve, the community is not nearly as active as OpenFoams', and the lack of more detailed documentation and examples.

4.3.3.1 Visualising Simulation Results

The use of paraFOAM, an utility to open ParaView with the simulations results OpenFOAM obtained, simplifies data loading into the software. This data loading has drawbacks, for example, to display a stream-tracer starting from the inlet surface, one has to detach the inlet surface from the main geometry.

For this work the OpenFOAM results were converted to VTK format, with the foamToVTK utility, and than loaded the data for volume, inlet and outlet. This approach prevents the loading off all the data at once, the user chooses what data he will need and loads it to the software for processing.

ParaView rescales the color maps taking into account the all displayed data. So with a single cut for all sections, the top sections would exhibit for ex. high pressures and the lowest sections low pressures. This way makes it impossible to see the small pressure gradient inside a section, if the section has a 10 units variation but all the displayed data as a 1000 units variation, each section will be displayed as a solid color. To present the small gradients, each section had to be isolated by cutting and slicing.

4.4 Using OpenFOAM

BlockMesh is a powerful tool, fast and precise, it gives the user full control over mesh creation but, for complex geometries, is very difficult to use.For mesh creation commercial software may be used and the meshes easily converted with one of the utilities. Salome-Meca 5.1.3 is a powerful CAD like software, with some bugs in the extrusion process (a helix with small pitch can be troublesome, extruding multiple connected pipes is a problem in version 5.1.3), but it gives the user the possibility to create quality meshes very fast.

While running the simulation, information is displayed regarding important steps of the solver algorithm. This information may reveal what part of the algorithm is causing the method to diverge, and may give a hint on what method works better.

In early tests simulation convergence was easier using LES, so RAS was discarded. RAS was faster then LES but initially both would take approximately 24 hours to simulate 10 seconds (using all 12 processors), but the simulation would diverge near the first second (after almost 3 hours of computation).

Adding 1 non-orthogonal corrector improved stability and adding a second one improved even further. The second corrector did add a small delay in each iteration.

A change was made to the PISO solver C++ code in order to work with Courant dependent time intervals. Each time the Courant number increased above a predefined limit, the time step of the next iteration would diminished, and keep simulation under convergence. This improved stability until the point where the time interval was so small that the simulation almost didn't evolve. This was a change in the high level control of the simulation, but required changing C++ code and compiling the solver [http://www.cfd-online.com/Forums/openfoam/71145-time-openfoam.html].

From the displayed information it was possible to see that the problem was in the pressure equation. The pressure equation didn't converge and iterated to the limit (1000 iterations). Changing the derivatives from Euler method to a mixed Euler/CrankNicholson improved simulation stability (and speed due to less iterations of the pressure equation). This was a change in the low level control of the simulation.

Even though the simulation was converging, there were still too many iterations of the pressure equation. Making changes to the numerical methods for laplacian, divergence or interpolation had no sensible effect except for delaying the simulation.

Having identified the pressure equation as the critical step in convergence, the solution came from a change in method for approaching this equation. With the use of a GAMG (generalised geometric-algebraic multi-grid) pre-conditioner the pressure equation would converge in less than 5 iterations, usually in just two iterations. The GAMG pre-conditioner makes estimations on a coarse mesh and interpolates them for the PCG to use them in the fine mesh.

The reason for the GAMG pre-conditioner effectiveness is probably due to the characteristics of the flow. The secondary flow often results in a rotation like two coupled gear wheels, so there is the possibility that adjacent volume elements may possess opposite velocities in one of the components (x, y or z). In this context the GAMG pre-conditioner probably provides better estimations,

the other methods probably result in a estimation with a wrong component direction.

After this last change 10 seconds of simulation took approximately 3 hours of computation.

Regarding Post-Processing, ParaView is a complex (and buggy) software package. The sequence of making a cut, selecting cell centres, displaying vectors, surfacing the vectors (displaying their components on the slide face), usually terminated prematurely with a crash. The considerations for pre-processing software are valid for post-processing, the user may use utilities to export results to a format that other specialized software packages can read.

Pre- and Post-Processing are essentially left open for specialized software packages in meshing and in visualization.

The ideal OpenFOAM user has programming skills in C++ and Python, knowledge of numerical methods, and CFD in general, and is comfortable working with multiple software for drawing geometry, building meshes and visualizing results. Besides the very steep learning curve, the major problem of OpenFoam is that it is not a complete package, but a tool to solve equations with finite elements/areas/volumes. By concentrating their efforts in the equation solving process, OpenFoam grew quickly in size and complexity, and it is now possible to apply it to solve the majority of engineering problems.

Chapter 5

Results and Discussion

As previously detailed, these results were obtained via the PISO algorithm. Recapitulating the boundary and initial conditions: the initial conditions were 0 velocity in all the volume elements; the boundary conditions were 0 relative pressure at the outlet, 0 velocity near the tube wall, constant velocity profile at the inlet.

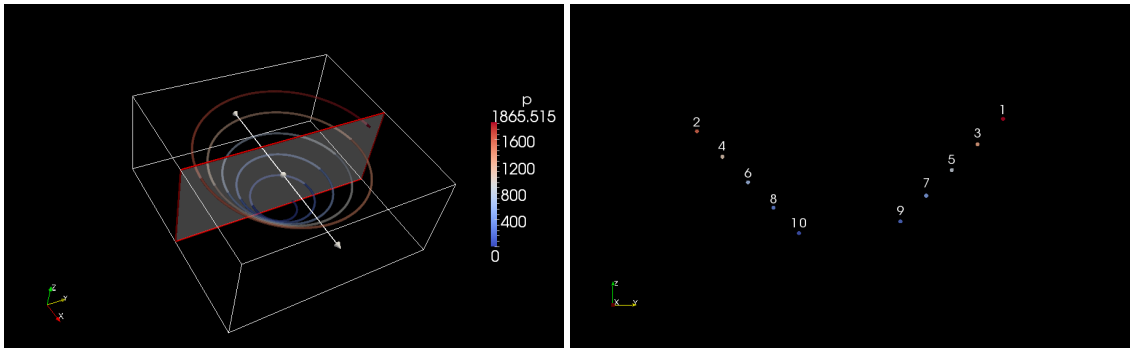
The geometry is the conical helix and the tube internal diameter is 1/2".

The temperature, density and viscosity remain constant throughout the simulation.

The simulations were executed for two different inlet velocities, 0.06 m/s (27.4 l/h) and 1.0 m/s (456 l/h).

5.1 Simulation Results

From this slice 10 sections were obtained and numbered as figure 5.1 shows.



(a) Slice plane. (b) Slice sections.

Figure 5.1: Conical helix vertical slice plane and sections.

Results for pressure and velocity will be presented. The first simulation results write-out instant was 0.1 seconds, at this point the system was already in steady-state, up to instant 25 seconds the results remained unchanged.

A slice of the conical helix was created as it is shown in figure 5.1. The curvature and torsion of this helix increase from top to bottom.

The results will be presented for sections 2, 6, 10, and Outlet. Sections 2, 6 and 10 were selected because they lie on the same side of the helix. The Outlet lies on the opposite side so the X component will be inverted.

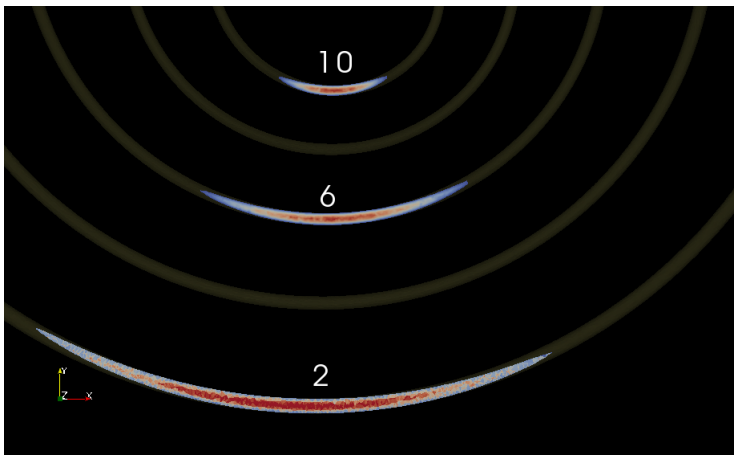


Figure 5.2: Conical helix horizontal slice planes and sections.

Horizontal slices were made, as shown in figure 5.2 to display the velocity profile along the tube axis. The slices were selected to be coincident/perpendicular to sections 2, 6 and 10 of the vertical slices.

The Re_c is 6174, as expected above the 2100 for straight tubes.

For the 0.06 m/s, Re=8523 and 2480 <= De <= 41300. This means that this inlet velocity lies in a flow transition region.

For the 1.0 m/s, Re=142058 and 1358 <= De <= 22600. This velocity is clearly turbulent.

Using theoretical correlations presented in chapter 2, equation (2.5) allows the estimation of some values to validate the simulations. For 1 m/s the pressure drop would be 5630 Pa and for 0.06 m/s 38.84 Pa. These values, even accounting for the hydrostatic are very far from the simulation results.

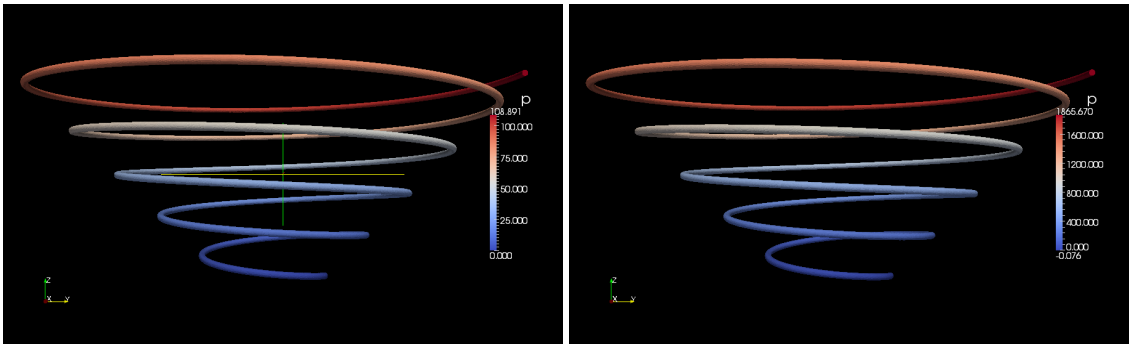
The correlations are know not to be very precise and, were not developed with a conical helix in mind. Nevertheless the differences are very large, the simulation results will have to be validated with experimental results.

5.1.1 Pressure

The figures for the pressure gradient at 0.06 m/s 5.3(a), and for 1 m/s 5.3(b) are consistent with the boundary conditions, both are 0 Pa at the outlet. As it can be seen in figure 5.3 for the 0.06 m/s

simulation the head loss between outlet and inlet is 108.891 Pa, and for 1.0 m/s this is 1865.670 Pa.

For a straight tube the pressure loss would be 27 Pa/m (0.06 m/s). This is consistent with the results of the correlations presented by Mishra [9] that, for this case, would result in a 1.04 f_c/f_s ratio for 0.06 m/s and 1.05 for 1.0 m/s. This correlation only depends on the Reynolds number



(a) Inlet velocity 0.06 m/s.

(b) Inlet velocity 1.0 m/s.

Figure 5.3: Conical helix simulation - Pressure profile.

The pressure has small but noticeable variations across the tube sections 5.9. For the 0.06 m/s case the variations lie within 0.02 Pa, for the 1.0 m/s case the variations go up to 0.2 Pa. Besides the change in patterns, it is also possible to verify that there is an increase in hydraulic pressure from section 2 to 6 and 10.

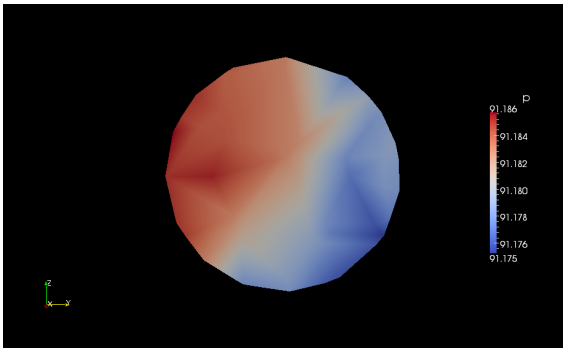
5.1.2 Velocity

Regarding the horizontal sections, showing the flow of 0.06 m/s, of the conical helix tube in figures 5.5(a), 5.5(b) and 5.5(c). Due to the fact that the section is horizontal and that the tube descends at a certain angle, the left side is near the bottom of the pipe section, the middle crosses the pipe center and the right side reaches the top of the tube. The X component of the flow shows, in figure 5.5(a), a gradient starting with low velocities on the left side (near tube bottom surface), reaches high velocities in the center and goes back to low velocities on the write (near tube top surface). This is true for all 3 sections. In the figures is possible to observe that the Y axis points to the top. Because the flow evovlves from the left to the right, the Y component of the flow shows in figure 5.5(b) a gradient starting with low velocities on the left side (negative Y component), reaches near 0 velocities in the center and goes to high velocities on the write (positive Y component). This is also true for all 3 sections. Due to the helix geometry, the flow is continuously descending from left to right. The Z component of the flow, shown in figure 5.5(c), as areas of negative velocity (descending flow) in the center, and zero velocities on the left and on the right (near the tube surface).

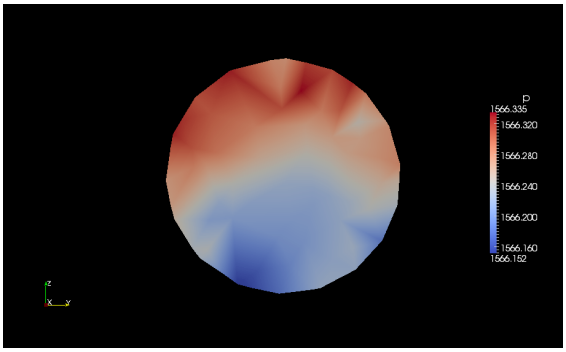
The same analysis can be made for the 1.0 ms case, figures 5.5(a), 5.5(b) and 5.5(c). In fact the results are very similar, even when observing the patterns that emerge. One would expect the

Z component to be more homogeneous, but it is not. Because the tube is descending, a 0 velocity in Z means that the flow is rising. So in both 0.06 m/s and 1.0 m/s cases the Z component displays motion in an unexpected direction.

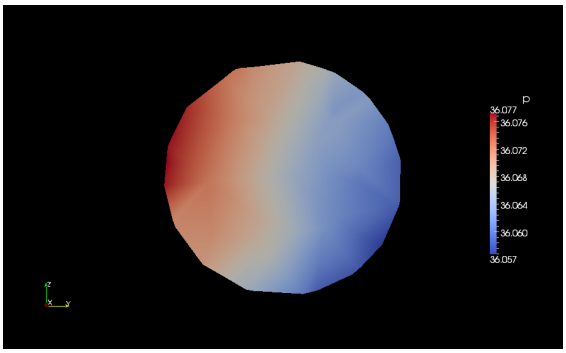
The vertical sections of the tube display higher velocities in the center, and near 0 velocity neat the tube surface. All sections exhibit a not very interesting X component, the X is in the direction of the flow so a circular velocity profile is expected. However, for the Y and Z components, in the outlet sections for both 0.06 m/s and 1.0 m/s (figures 5.10 and 5.14), the Y and Z may form a pattern like the one in figure 2.3 (with the flow rotating in the opposite direction).



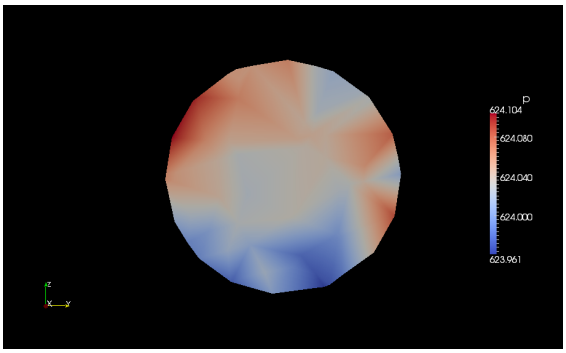
(a) Pressure for inlet velocity 0.06 m/s section 2.



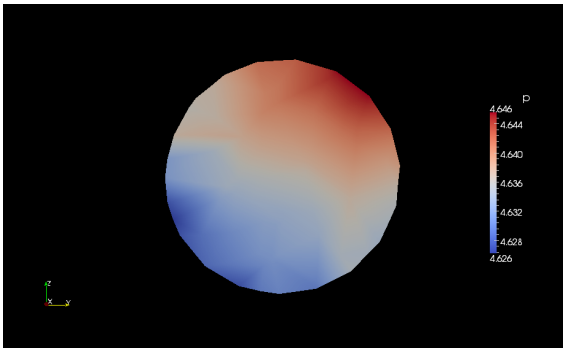
(b) Pressure for inlet velocity 1.0 m/s section 2.



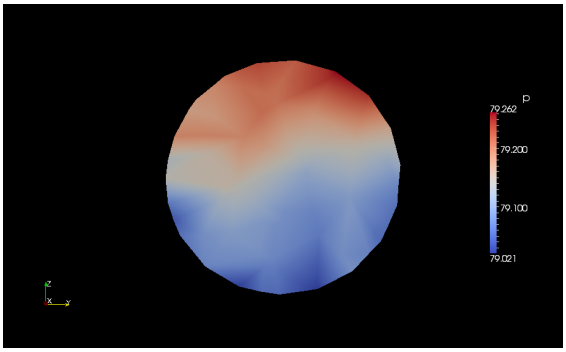
(c) Pressure for inlet velocity 0.06 m/s section 6.



(d) Pressure for inlet velocity 1.0 m/s section 6.

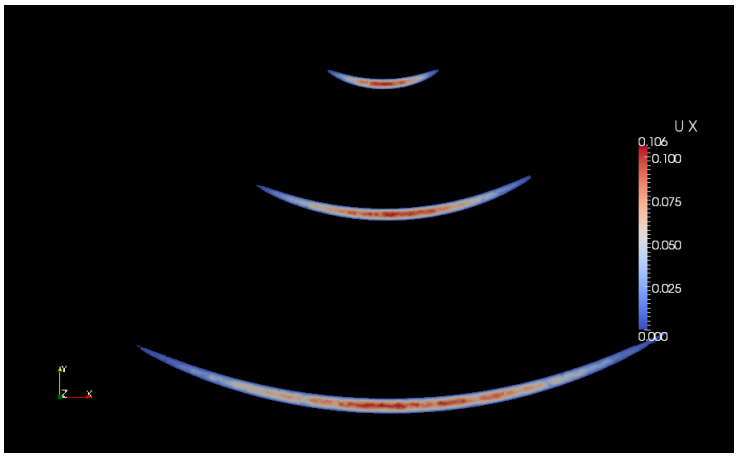


(e) Pressure for inlet velocity 0.06 m/s section 10.

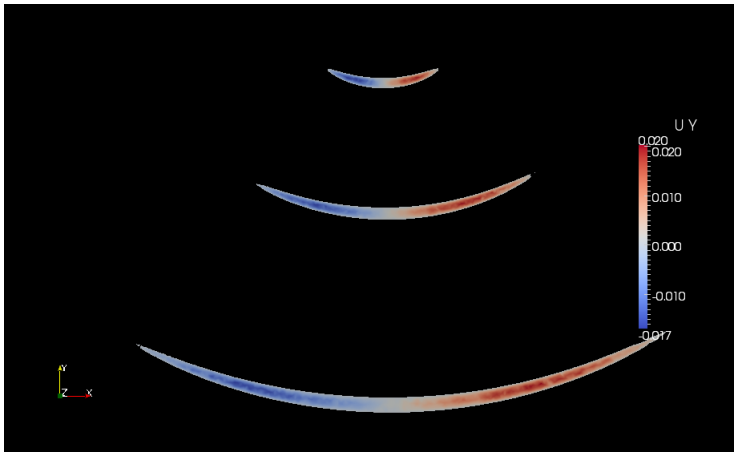


(f) Pressure for inlet velocity 1.0 m/s section 10.

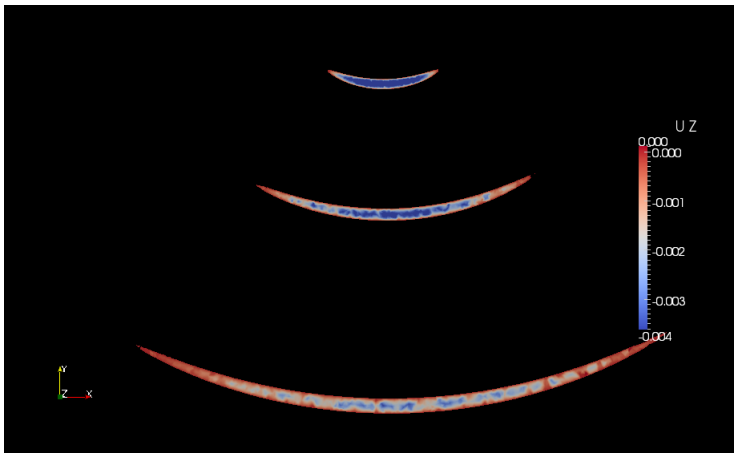
Figure 5.4: Pressure profiles across sections 2, 6 and 10 of the helix for 0.06 m/s and 1.0 m/s.



(a) Velocity, X component, for inlet velocity 0.06 m/s.

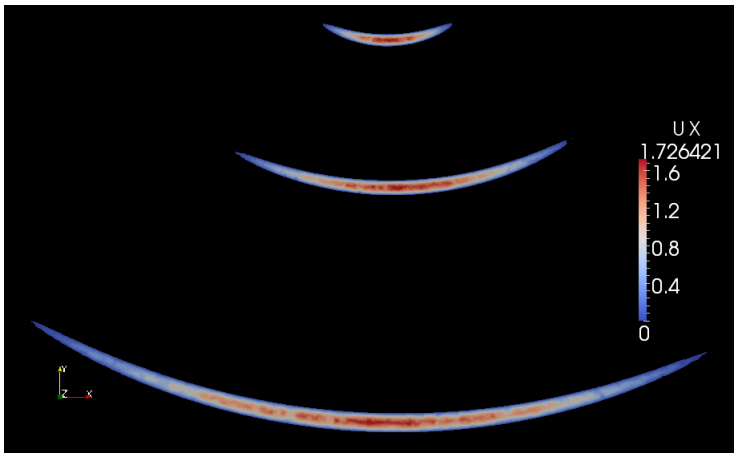


(b) Velocity, Y component for inlet velocity 0.06 m/s.

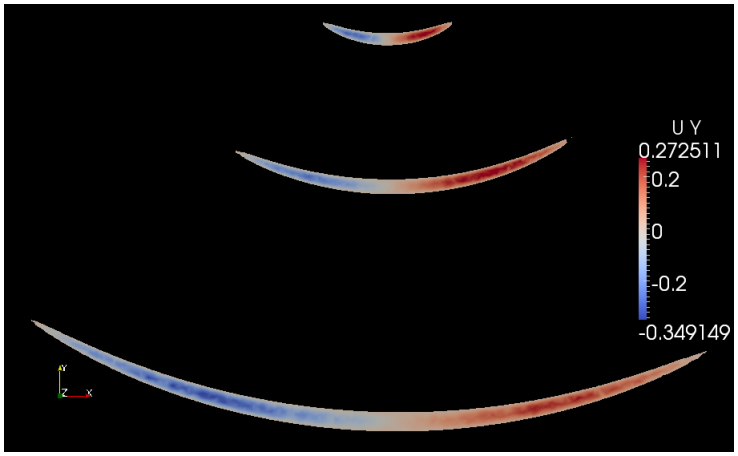


(c) Velocity, Z component for inlet velocity 0.06 m/s.

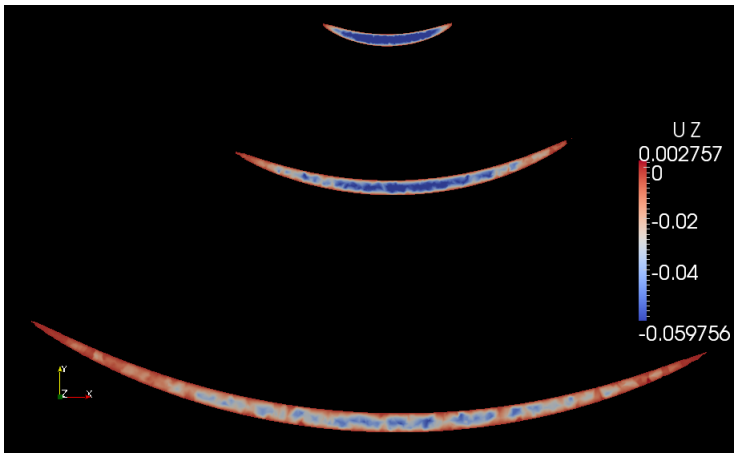
Figure 5.5: Velocity profiles along helix tube for 0.06 m/s inlet velocity.



(a) Velocity, X component, for inlet velocity 1.0 m/s.

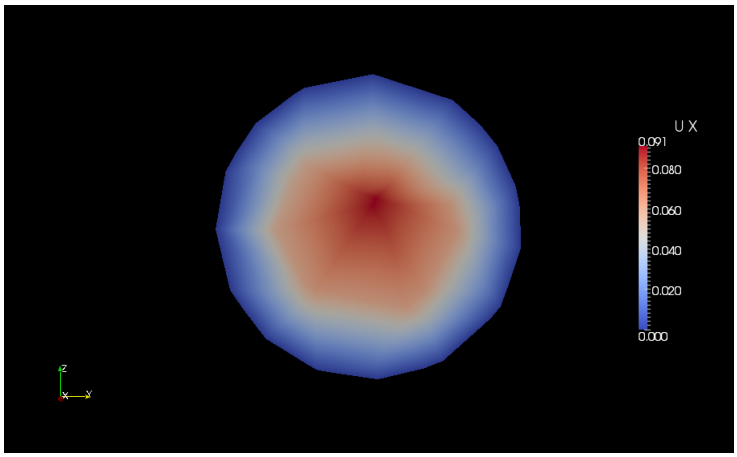


(b) Velocity, Y component for inlet velocity 1.0 m/s.

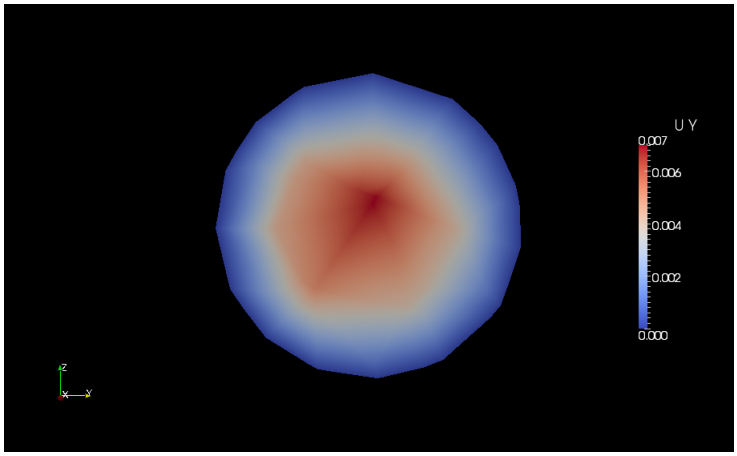


(c) Velocity, Z component for inlet velocity 1.0 m/s.

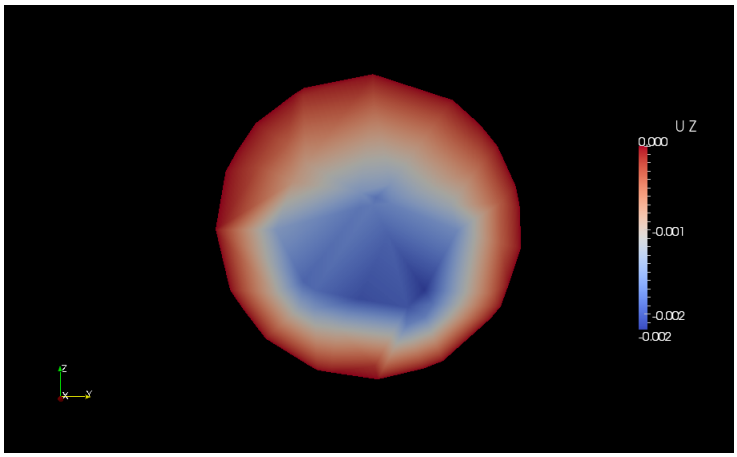
Figure 5.6: Velocity profiles along helix tube for 1.0 m/s inlet velocity.



(a) Velocity, X component, for inlet velocity 0.06 m/s section 2.

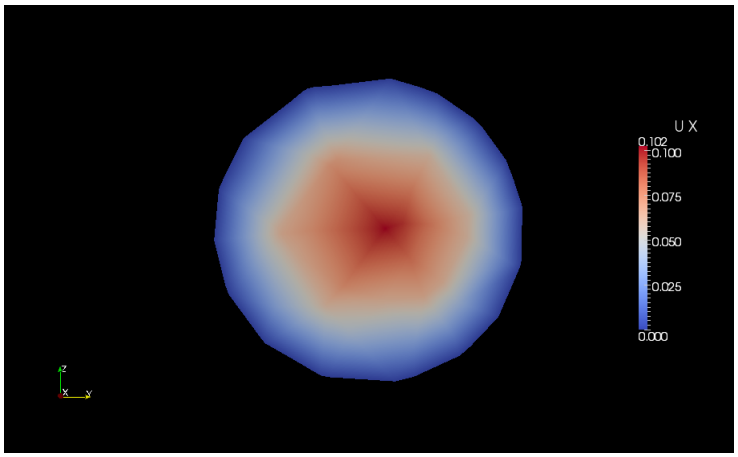


(b) Velocity, Y component for inlet velocity 0.06 m/s section 2.

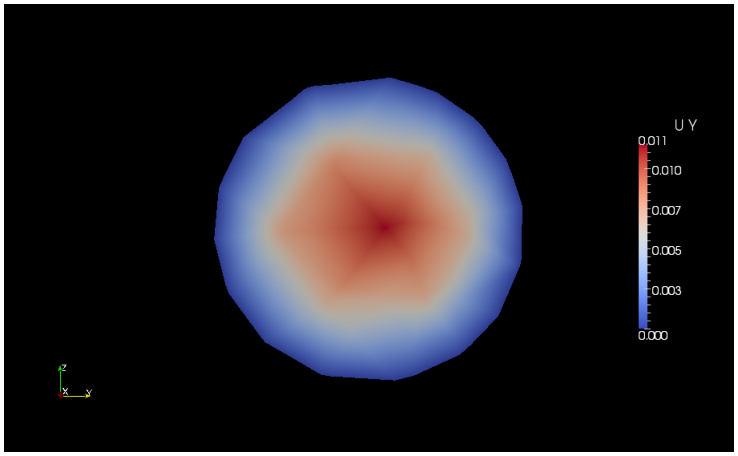


(c) Velocity, Z component for inlet velocity 0.06 m/s section 2.

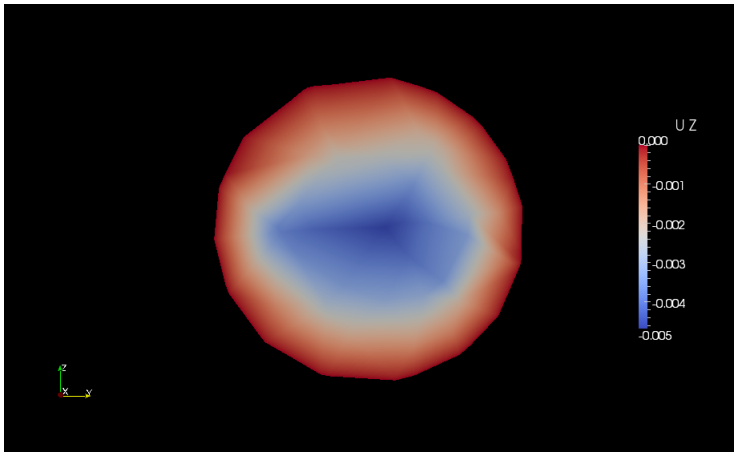
Figure 5.7: Velocity profiles across section 2 of the helix for 0.06 m/s.



(a) Velocity, X component, for inlet velocity 0.06 m/s section 6.

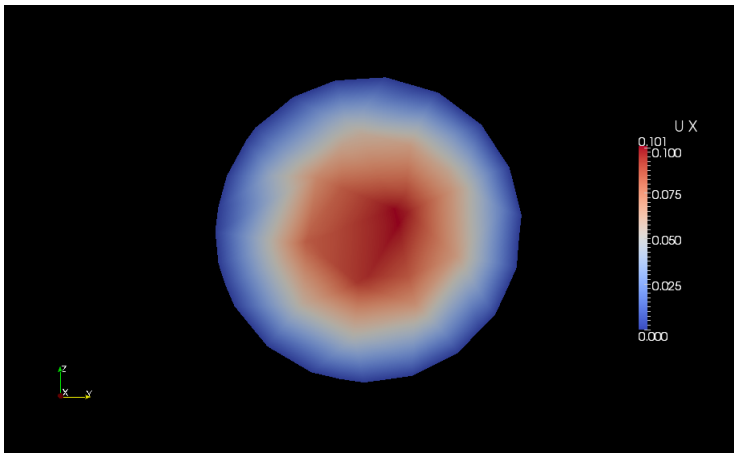


(b) Velocity, Y component for inlet velocity 0.06 m/s section 6.

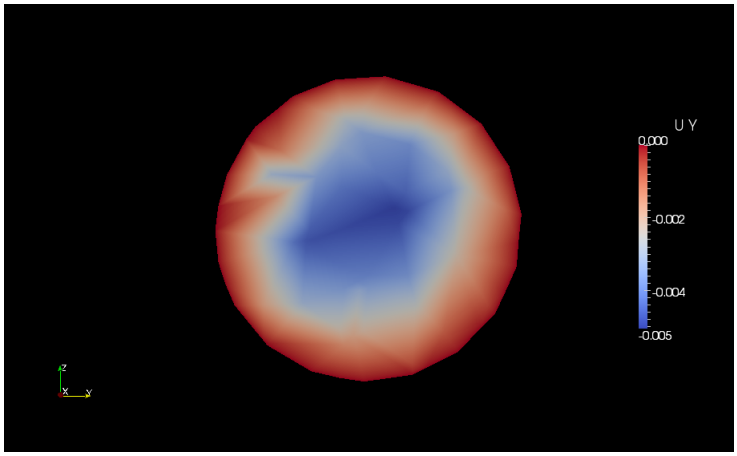


(c) Velocity, Z component for inlet velocity 0.06 m/s section 6.

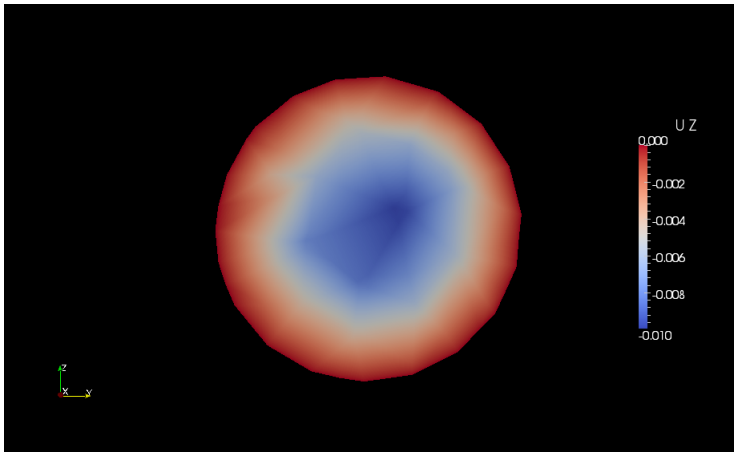
Figure 5.8: Velocity profiles across section 6 of the helix for 0.06 m/s.



(a) Velocity, X component, for inlet velocity 0.06 m/s section 10.

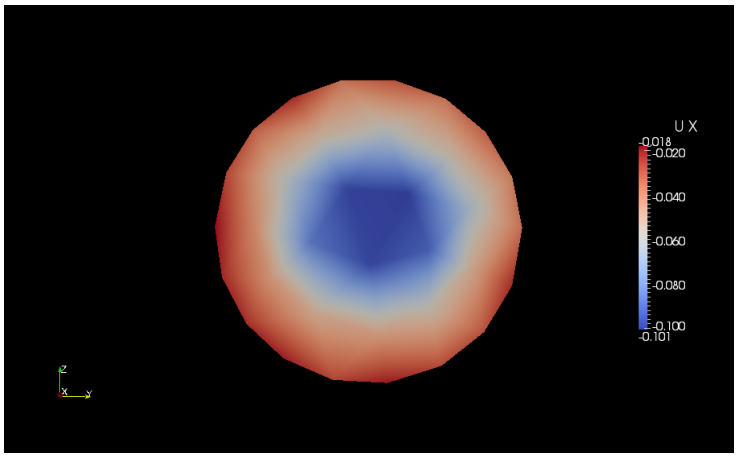


(b) Velocity, Y component for inlet velocity 0.06 m/s section 10.

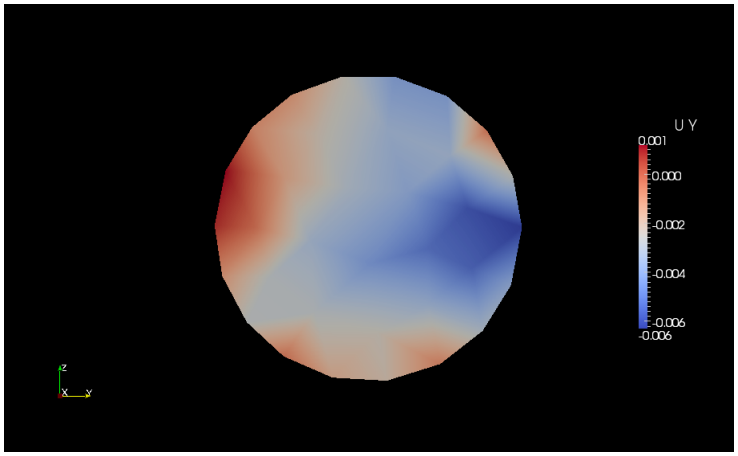


(c) Velocity, Z component for inlet velocity 0.06 m/s section 10.

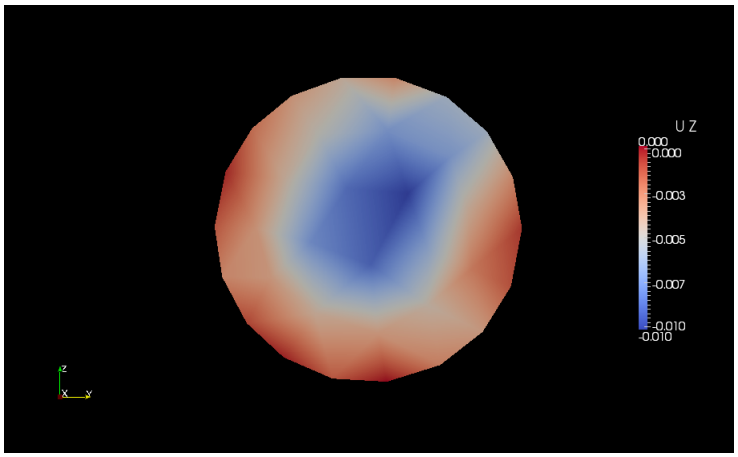
Figure 5.9: Velocity profiles across section 10 of the helix for 0.06 m/s.



(a) Velocity, X component, for inlet velocity 0.06 m/s, Outlet section.

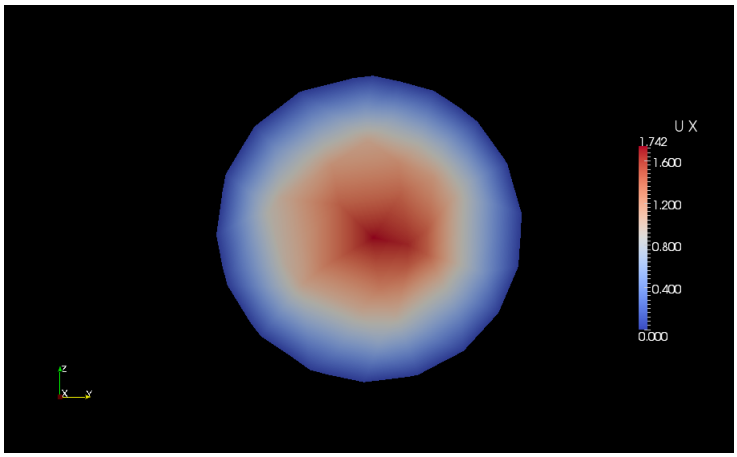


(b) Velocity, Y component for inlet velocity 0.06 m/s, Outlet section.

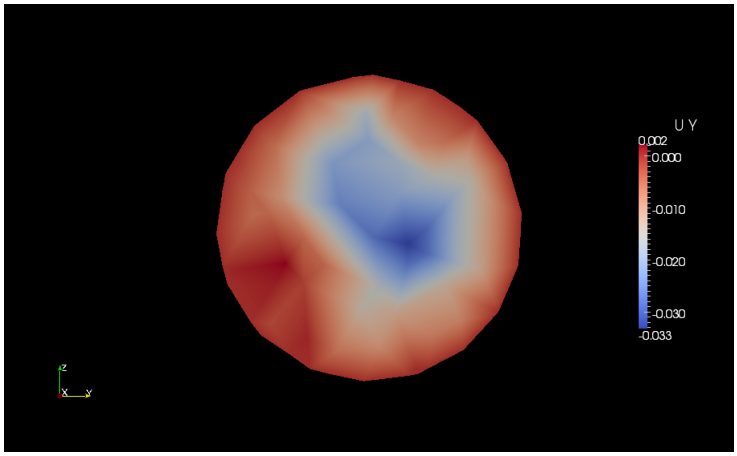


(c) Velocity, Z component for inlet velocity 0.06 m/s, Outlet section.

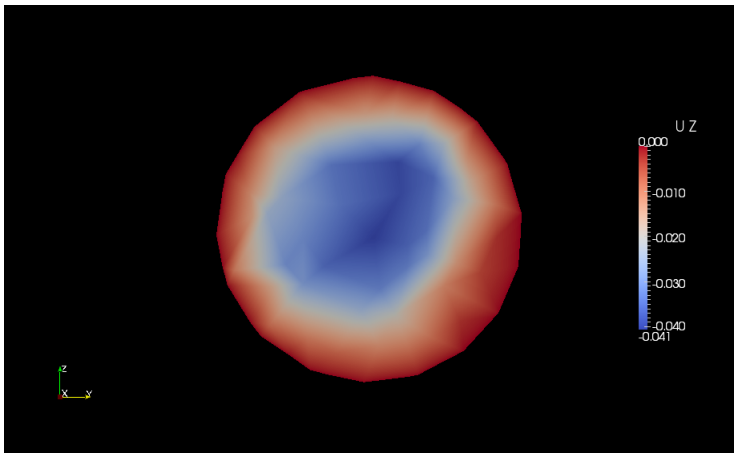
Figure 5.10: Velocity profiles across Outlet section of the helix for 0.06 m/s.



(a) Velocity, X component, for inlet velocity 1.0 m/s section 2.

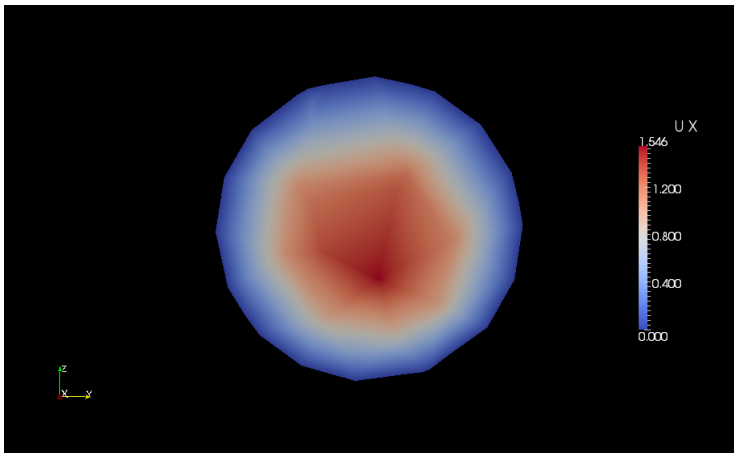


(b) Velocity, Y component for inlet velocity 1.0 m/s section 2.

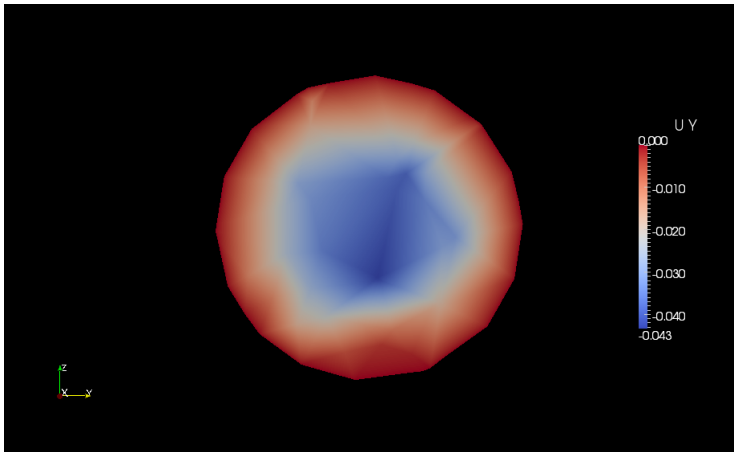


(c) Velocity, Z component for inlet velocity 1.0 m/s section 2.

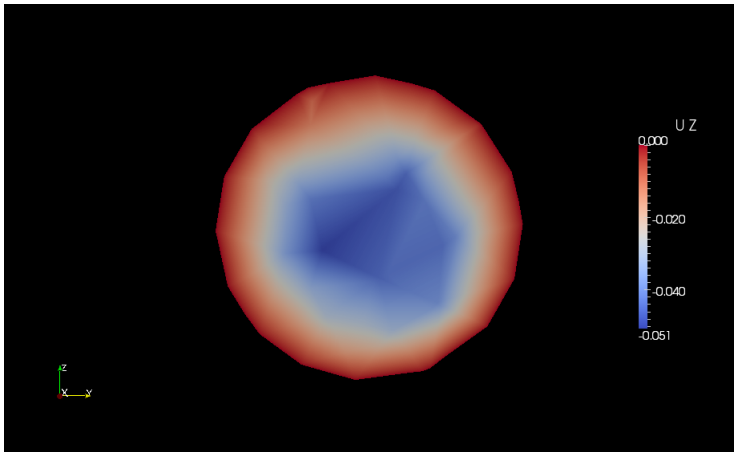
Figure 5.11: Velocity profiles across section 2 of the helix for 1.0 m/s.



(a) Velocity, X component, for inlet velocity 1.0 m/s section 6.

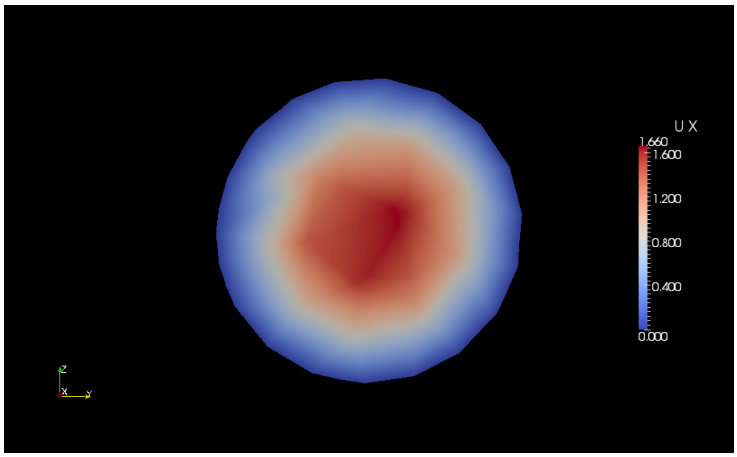


(b) Velocity, Y component for inlet velocity 1.0 m/s section 6.

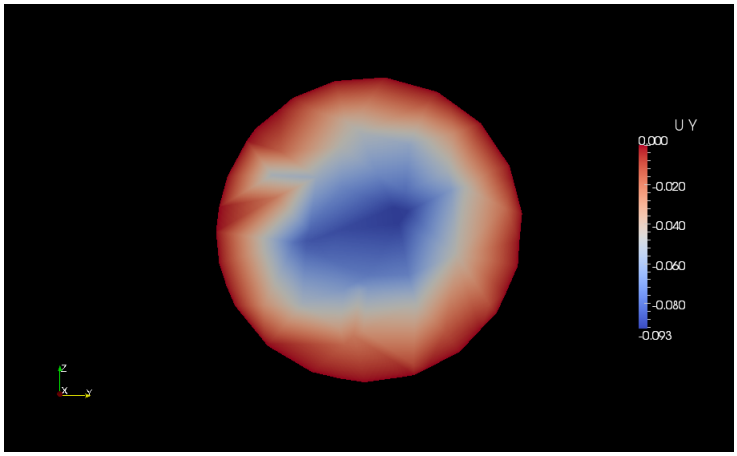


(c) Velocity, Z component for inlet velocity 1.0 m/s section 6.

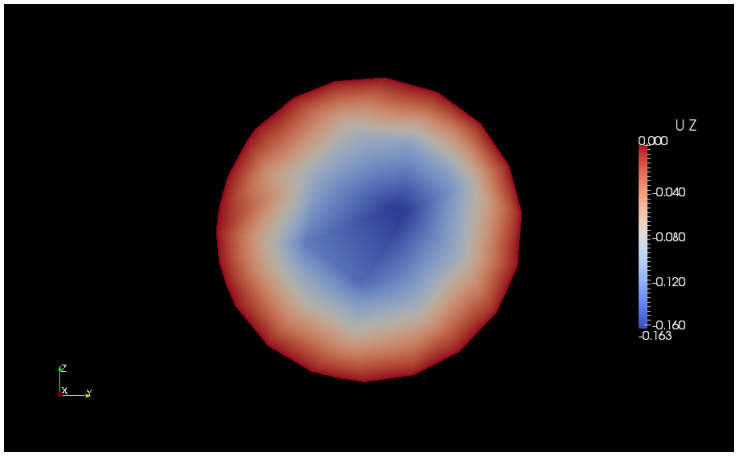
Figure 5.12: Velocity profiles across section 6 of the helix for 1.0 m/s.



(a) Velocity, X component, for inlet velocity 1.0 m/s section 10.

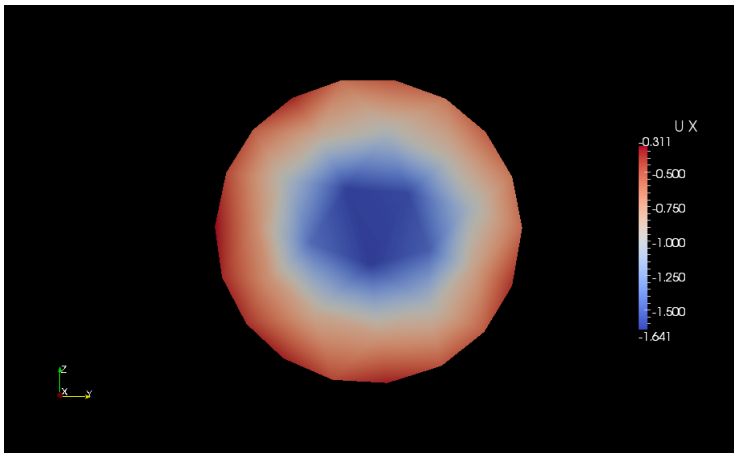


(b) Velocity, Y component for inlet velocity 1.0 m/s section 10.

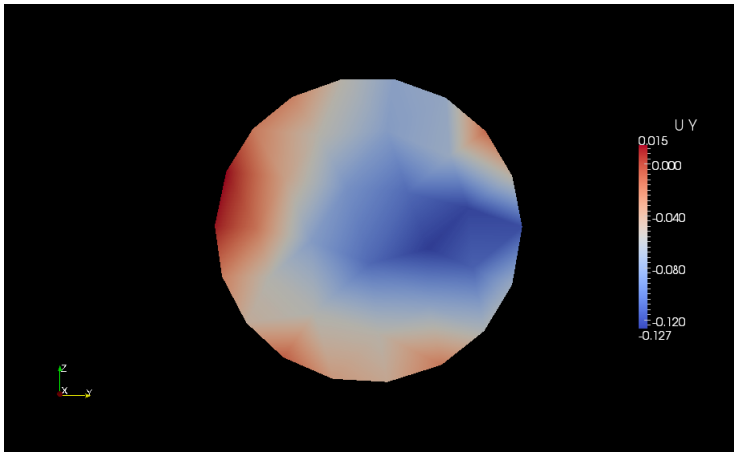


(c) Velocity, Z component for inlet velocity 1.0 m/s section 10.

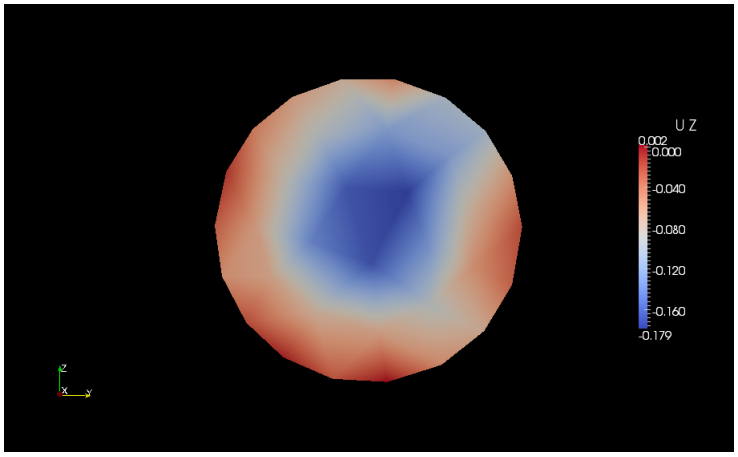
Figure 5.13: Velocity profiles across section 10 of the helix for 1.0 m/s.



(a) Velocity, X component, for inlet velocity 1.0 m/s, Outlet section.



(b) Velocity, Y component for inlet velocity 1.0 m/s, Outlet section.



(c) Velocity, Z component for inlet velocity 1.0 m/s, Outlet section.

Figure 5.14: Velocity profiles across Outlet section of the helix for 1.0 m/s.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The main objective of this work was the simulation of the incompressible isothermal single phase liquid flow, in a conical helix tube of circular section.

This simulation was performed on a conical helix geometry (constant pitch, linear radius variation) that was not found in the literature. This geometry was simulated for two different inlet velocities.

The results were shown as sections of the helix to aid in the visualization and interpretation, they seem to be consistent but lack experimental results to confirm.

This work demanded a very steep learning process involving CFD, python, C++, OpenFoam, Salome-Meca and ParaView.

OpenFOAM can almost be used out-of-the-box for simple geometries and flows. The solvers can be applied to many different problems and it is possible to extend them to cover other problems. The software is an excellent example of high quality C++ code, this has advantages like few bugs, but can sometimes be difficult to adapt to particular needs.

For more complex geometries aid is needed from external packages.

For visualization ParaView can satisfy the users in the majority of occasions, it has the potential to be extended almost like OpenFOAM, but it has to much to improve in many aspects.

For C++ training two excellent references were used [20, 21].

6.2 Future Work

It is clear that the results presented are part of a work in progress. In the future other conical helix geometries will be simulated with other inlet velocities and turbulence models.

The simulations presented were already used for a first successful Lagrangian particle tracking simulation. At the moment I am working in changing the OpenFOAM PISO solver to adapt it to our needs.

A device is under construction and, in the future, the simulation results will be validated with experimental results.

References

[1] J. Eustice, "Flow of water in curved pipes," *Proceedings of the Royal Society of London*, vol. 84, pp. 107–118, 1910.

[2] J. Eustice, "Experiments of streamline motion in curved pipes," *Proceedings of the Royal Society of London*, vol. 85, pp. 119–131, 1911.

[3] W. R. Dean, "Note on the motion of fluid in a curved pipe.," *Philosophical magazine*, vol. (7) 4, no. 20, pp. 208–223, 1927.

[4] W. R. Dean, "The stream-line motion of fluid in a curved pipe," *Philosophical magazine*, vol. (7) 5, no. 30, pp. 673–695, 1928.

[5] C. White, "Streamline flow through curved pipes," *Proceedings of the Royal Society of London*, vol. 123, pp. 645–663, April 1929.

[6] G. I. Taylor, "The criterion for turbulence in curved pipes," *Proceedings of the Royal Society of London*, vol. 124, pp. 243–249, June 1929.

[7] H. Ito, "Friction factors for turbulent flow in curved pipes," *Trans. Amer. Soc. Mech. Eng.*, vol. (D) 81, pp. 123–134, 1959.

[8] P. Naphon and S. Wongwises, "A review of flow and heat transfer characteristics in curved tubes," *Renewable and Sustainable Energy Reviews*, vol. 10, pp. 463–490, 2006.

[9] P. Mishra and S. N. Gupta, "Momentum transfer in curved pipes. 1. newtonian fluids," *Ind. Eng. Chem. Process Des. Dev.*, vol. 18, no. 1, pp. 130–137, 1979.

[10] S. V. Patankar, V. S. Pratap, and D. B. Spalding, "Prediction of turbulent of in curved pipes," *Journal of Fluid Mechanics*, vol. 67, pp. 583–595, 1975.

[11] S. Patankar and D. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," *International Journal of Heat and Mass Transfer*, vol. 15, pp. 1787–1806, 1971.

[12] R. Issa, A. Gosman, and A. Watkins, "The computation of compressible and incompressible recirculating flows by a non-iterative implicit scheme," *Journal of Computational Physics*, vol. 62, pp. 66–82, 1986.

[13] S. Ali, "Pressure drop correlations for ow through regular helical coil tubes," *Fluid Dynamics Research*, vol. 28, pp. 295–310, 2001.

[14] M. Germano, "On the effect of torsion on a helical pipe flow," *Journal of Fluid Mechanics*, vol. 125, pp. 1–8, 1982.

[15] D. Gammack and P. E. Hydon, "Flow in pipes with non-uniform curvature and torsion," *Journal of Fluid Mechanics*, vol. 433, pp. 357–382, 2001.

[16] O. Limited, "The open source cfd toolbox - openfoam user's guide," tech. rep., OpenCFD Limited, August 2010. Version 1.7.1.

[17] Y. Kulakov, "Salome to openfoam mesh conversion tutorial," tech. rep., University of Malta, 2010.

[18] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*. Springer, Jan 1996.

[19] T. Cebeci, *Turbulence Models and Their Application: Efficient Numerical Methods With Computer Programs*. Springer, Jan 2004.

[20] B. Eckel, *Thinking in C++: Introduction to Standard C++, Volume One*. Prentice Hall, Jan 2000.

[21] B. Eckel and C. Allison, *Thinking in C++, Volume 2: Practical Programming*. Prentice Hall, Jan 2004.

Appendix A

Set-up Files for OpenFOAM Simulation

A.1 Geometry and Mesh script

```
import geompy
import salome
import numpy as np
import smesh

gg = salome.ImportComponentGUI("GEOM")

#Pontos da espiral completa
t = np.arange(0,10*np.pi+0.01/np.pi,0.01/np.pi) # N*np.pi altera numero de volta
r = np.linspace(100,500,t.size) #Altura da Espiral

x1=r*np.sin(t)
y1=r*np.cos(t)
z = r

# criar pontos
pointList=[]
for i in range(0,t.size):
        pointList.append(geompy.MakeVertex(x1[i], y1[i], z[i]))
#pointList.reverse()
print "vertices done"

# criar curvas
interpol = geompy.MakeInterpol(pointList, False)
id_interpol1 = geompy.addToStudy(interpol,"Interpol")
gg.createAndDisplayGO(id_interpol1)
print "curves done"

#create the initial vectors
vector_1 = geompy.MakeVector(pointList[t.size-1], pointList[t.size-2])
id_vector1 = geompy.addToStudy(vector_1,"Vector1")
gg.createAndDisplayGO(id_vector1)
print "vectors done"
```

```
#disk1 = geompy.MakeDisk(pointList[0], vector_1, 0.025)
disk1 = geompy.MakeDiskPntVecR(pointList[0], vector_1, 6.35)
id_disk1 = geompy.addToStudy(disk1, "Disk1")
gg.createAndDisplayGO(id_disk1)
print "main 1 disk done"

pipe_1 = geompy.MakePipe(disk1, interpol)
id_pipe_1 = geompy.addToStudy(pipe_1,  "Main Pipe1")
gg.createAndDisplayGO(id_pipe_1)
print "main 1 pipe done"


# add objects in the study
#id_vector_1 = geompy.addToStudy(vector_1, "Vector1")
#id_vector_2 = geompy.addToStudy(vector_2, "Vector2")
print "main objects added to study"

print "objects displayable"

# define the mesh geometry
#Mesh_1 = smesh.Mesh(interpol,"Espiral")

#Definition of hypotheses and algorithms 1D, 2D and 3D :
#Regular_1D = Mesh_1.Segment()
#Nb_Segments_1 = Regular_1D.NumberOfSegments(10)
#Nb_Segments_1.SetDistrType( 0 )
#Quadrangle_2D = Mesh_1.Quadrangle()
#Hexa_3D = Mesh_1.Hexahedron()

#NETGEN_3D_Parameters = smesh.smesh.CreateHypothesis('NETGEN_Parameters', 'libNETGENE
#NETGEN_3D_Parameters.SetMaxSize( 500 )
#NETGEN_3D_Parameters.SetSecondOrder( 0 )
#NETGEN_3D_Parameters.SetOptimize( 1 )
#NETGEN_3D_Parameters.SetFineness( 3 )
#Mesh_1 = smesh.Mesh(Solid_1)
#status = Mesh_1.GetMesh().AddHypothesis( pipe_1, NETGEN_3D_Parameters )
#Netgen_1D_2D_3D = smesh.smesh.CreateHypothesis('NETGEN_2D3D', 'libNETGENEngine.so')
#status = Mesh_1.GetMesh().AddHypothesis( pipe_1, Netgen_1D_2D_3D )
# Launch of the mesh computation:
#isDone = Mesh_1.Compute()
#if not isDone : print "Mesh is not computed"

# Create a group of faces
#aSmeshGroup1 = Mesh_1.GroupOnGeometry(face,"SmeshGroup1")
#group = geompy.CreateGroup(pipe_1, geompy.ShapeType["FACE"])
#group_name = "Espiral" + "_grp"
#geompy.addToStudy(group, group_name)
```

```
#group.SetName(group_name)
#gg.createAndDisplayGO(group_name)

# Add faces to the group
#faces = geompy.SubShapeAllIDs(pipe_1, geompy.ShapeType["FACE"])
#geompy.UnionIDs(group, faces)
```

A.2 system/controlDict

```
/*--------------------------------*- C++ -*----------------------*\
| =========                 |                                     |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox |
|  \\    /   O peration     | Version:  1.7.1                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.com          |
|    \\/     M anipulation  |                                     |
\*---------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pisoFoam_mod;

//startFrom       startTime;
startFrom       latestTime;

startTime       0;

stopAt          endTime;
//stopAt          nextWrite;

endTime         60;

deltaT          2e-05;

writeControl    adjustableRunTime;

writeInterval   0.1;

purgeWrite      0;

writeFormat     ascii;
```

```
writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep on;

maxCo 0.4;

maxDeltaT 0.05;

minDeltaT 1e-6;

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

A.3 system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                           |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox |
|  \\    /   O peration     | Version:  1.7.1                           |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                |
|    \\/     M anipulation  |                                           |
\*-------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicholson 0.5;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
```

```
    grad(U)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
    div(phi,k)      Gauss limitedLinear 1;
    div(phi,B)      Gauss limitedLinear 1;
    div(phi,nuTilda) Gauss limitedLinear 1;
    div(B)          Gauss linear;
    div((nuEff*dev(grad(U).T()))) Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nuEff,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DkEff,k) Gauss linear corrected;
    laplacian(DBEff,B) Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(U)  linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}


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

A.4 system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                            |
```

```
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox |
|  \\    /   O peration     | Version:  1.7.1                       |
|   \\  /    A nd           | Web:      www.OpenFOAM.com            |
|    \\/     M anipulation  |                                       |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner
{
preconditioner GAMG;
         tolerance       1e-06;
         relTol          0.05;
smoother GaussSeidel;
agglomerator faceAreaPair;
nPreSweeps 1;
nPostSeeps 2;
nBottomSweeps 2;
cacheAgglomeration off;
nCellsInCoarsestLevel 20;
mergeLevels 1;
}
        tolerance       1e-06;
        relTol          0.05;
    };

    pFinal
    {
        solver          PCG;
        preconditioner
{
preconditioner GAMG;
         tolerance       1e-06;
         relTol          0.05;
smoother GaussSeidel;
agglomerator faceAreaPair;
```

```
nPreSweeps 1;
nPostSeeps 2;
nBottomSweeps 2;
cacheAgglomeration off;
nCellsInCoarsestLevel 20;
mergeLevels 1;
}
        tolerance       1e-06;
        relTol          0.05;
    }

    U
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }

    k
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }

    B
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }

    nuTilda
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 2;
```

```
}


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

A.5 Pressure boundary and initial conditions 0/p

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*-------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];      //  m2/s2

internalField   uniform 0;

boundaryField
{
    wall
    {
        type            zeroGradient;
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
}

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

A.6 Velocity boundary and initial conditions 0/U

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                           |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox |
|  \\    /   O peration     | Version:  1.7.1                           |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                |
|    \\/     M anipulation  |                                           |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];      //  m/s

internalField   uniform (0 0 0);

boundaryField
{
    wall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    inlet
    {
        type            fixedValue;
        value           uniform (-0.06 0 0);
    }

    outlet
    {
        type            zeroGradient;
    }
}

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