Modelling of wind flow over complex terrain using OpenFoam

Xabier Pedrue1lo Tapia

June 2009

Master’s Thesis in Energy Systems

Examiner: Ulf Larsson
Supervisor: Mathias Cehlin
ACKNOWLEDGMENTS

I thank my supervisor Dr Mathias Cehlin whose technical advice was very valuable and showed strong commitment whenever I needed his help. He has been always willing to listen and give advice thorough the whole process.

Also to my home University (UNAV, Tecnun) and Hogskolan i Gavle who made possible that I could spend a whole year abroad which has been one of the most important experiences in my life.

I would like to thank as well, to all my neighbour students who make the stay very enjoyable and help me out with format final touches that without them sure enough the present project would not look like it does.

And finally, I would like to truly thank my parents and sister who stood by me both in good and bad times in the last year abroad and especially in the time I was carrying out the present project.
ABSTRACT

Vattenfall is pursuing a massive expansion of wind power capacity in the following years. The most suitable emplacements for such purpose are already taken so sites which are not so favorable but still economically feasible are to be considered.

In order to assess the viability of the complex terrain locations onsite measurements are taken as well as a CFD tools employment is convenient since saves time and cost relative to a complete field measurement program.

The current commercial codes are considered somewhat over-priced and not sufficiently flexible when it comes to user-defined models, consequently, it is decided to develop the open source OpenFoam CFD tool.

The present thesis project is inside a larger project with which Vattenfall expects to integrate by 2011 OpenFoam CFD tool and be able to make wind power production accurate predictions over complex terrain.

OpenFoam CFD tool was validated using two different cases: a flat plane and an axi-symmetric plane getting results which strongly resembled to Fluent commercial code results.

New wall functions including roughness were added to OpenFoam’s current version (OF-1.5.) in order to be able to establish non-uniform variable roughness along the domain. In addition, a pre-processing tool was developed (roughnessToFoam) which permits an easy addition of the roughness data arising from map format file. The implemented wall functions were validated and turned out to work even better than the commercial code Fluent.

Coriolis and gravitational forces were successfully implemented being validated with theoretical formulation. It was appreciated that coriolis force can have a considerable influence when we come to domains which can span up to 20 km at high latitude places, therefore, it is considered that is necessary to take it into account.

Eventually, Openfoam’s pre-processing tool SnappyHexMesh was evaluated in order to analyze whether is useful or not to mesh complex terrains. Two cases were tried to be meshed and whereas the first one (bump case) was correctly meshed, when trying to mesh the second (Askervein hill) problems arose if many surface layers were wanted to be added. It was concluded that more research is needed in order to get the most of the meshing tool, since the big amount of variable needed to be set make it quite difficult to control correctly the process.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................................................................. 1
   1.1. Energy background .......................................................................................................................... 1
   1.2. Vattenfall ........................................................................................................................................ 2
   1.3. CFD for wind farm siting ................................................................................................................ 6
   1.4. Aims and tasks ............................................................................................................................... 9
   1.5. Method .......................................................................................................................................... 10
   1.6. Lay out .......................................................................................................................................... 11

2. THEORY ................................................................................................................................................ 13
   2.1. Atmospheric boundary layer (ABL) ................................................................................................ 13
       2.1.1. ABL structure and depth ........................................................................................................ 13
       2.1.2. Stratification and stability .................................................................................................... 14
       2.1.3. Turbulence .......................................................................................................................... 15
       2.1.4. Governing equations ............................................................................................................ 16
           2.1.4.1. Coriolis force ................................................................................................................ 17
       2.1.5. Time-Average Transport Equations ..................................................................................... 18
       2.1.6. Roughness ............................................................................................................................ 19
       2.1.7. Velocity profile approximations in the dynamic sublayer ...................................................... 19
           2.1.7.1. Logarithmic velocity profile ......................................................................................... 20
           2.1.7.2. Power-law expression .................................................................................................. 20
           2.1.7.3. Alexandrou’s formulation ............................................................................................. 21
   2.2. Numerical Modeling ....................................................................................................................... 23
       2.2.1. Turbulence modeling ............................................................................................................. 24
           2.2.1.1. Boussinesq approximation ......................................................................................... 24
           2.2.1.2. K-ε standard model .................................................................................................. 24
           2.2.1.3. K-ε RNG model ......................................................................................................... 25
       2.2.2. Discretization schemes .......................................................................................................... 26
       2.2.3. Near wall treatment: Wall functions ...................................................................................... 27
           2.2.3.1. Smooth surfaces ....................................................................................................... 27
           2.2.3.2. Rough surfaces ....................................................................................................... 28

3. CFD: OpenFoam ................................................................................................................................. 31
   3.1. OpenFoam Introduction ................................................................................................................. 31
3.2. Programming language of OpenFoam ................................................................. 32
  3.2.1. File structure & compiling ........................................................................... 32
  3.2.2. Equation representation ............................................................................ 33
3.3. OpenFoam cases ............................................................................................... 33
  3.3.1. System folder ............................................................................................. 34
  3.3.2. Constant folder .......................................................................................... 34
  3.3.3. 0 folder ....................................................................................................... 35
  3.3.4. Time directories ......................................................................................... 35
3.4. Standard solvers and libraries ........................................................................... 35
3.5. Pre-processing .................................................................................................... 35
  3.5.1. Gambit ........................................................................................................ 36
  3.5.2. SnappyHexMesh ......................................................................................... 36
3.6. Post-processing with Fluent ............................................................................... 37
4. PROCESS AND RESULTS .................................................................................... 39
  4.1. Comparison OpenFoam vs Fluent .................................................................... 39
    4.1.1. Flat plane .................................................................................................. 41
      4.1.1.1. Velocity comparison OpenFoam vs Fluent 1st order ....................... 42
      4.1.1.2. Velocity comparison OpenFoam vs Fluent using 2nd order ........... 43
      4.1.1.3. Velocity profiles ................................................................................ 44
      4.1.2. Bump case .............................................................................................. 46
        4.1.2.1. Comparison Experimental vs OpenFoam and Fluent .................. 48
    4.1.3. Conclusion .................................................................................................. 51
  4.2. Roughness implementation ............................................................................... 53
    4.2.1. Adding roughness as a variable in OF .................................................... 53
      4.2.1.1. Wall function validation .................................................................. 54
    4.2.2. Pre-processing roughness data application ............................................ 58
      4.2.3.1. Roughness map files ...................................................................... 58
      4.2.3.2. Roughness application ................................................................. 59
  4.3. Coriolis force .................................................................................................... 64
    4.3.1. Validation of the implemented coriolis forces ........................................ 64
  4.4. Gravity force ..................................................................................................... 68
  4.5. SnappyHexMesh ............................................................................................... 70
    4.5.1. Bump case with SnappyHexMesh ......................................................... 70
    4.5.2. Askervein ............................................................................................... 75
5. CONCLUSION ........................................................................................................ 79
6. FUTURE WORK .................................................................................................. 80
7. REFERENCES ...................................................................................................... 81
APPENDIX A ........................................................................................................ 83
APPENDIX B ........................................................................................................ 97
APPENDIX C ........................................................................................................ 103
LIST OF FIGURES

Figure 1.1 Vattenfall’s energy sources in 2008 .............................................................. 3
Figure 1.2. Vattenfall’s energy sources prediction by 2030 ........................................... 4
Figure 1.3. Vattenfall’s wind assets in June 2009 .......................................................... 5
Figure 1.4. CFD simulation example for wind farm siting .............................................. 7
Figure 2.1. Atmospheric boundary layer structure ...................................................... 13
Figure 2.2. Influence of the stability on the motion of a particulate. ............................. 14
Figure 2.3. Influence of the stability on the flow over a hill. [9] ..................................... 15
Figure 2.4. Velocity in a turbulent flow. [10] ................................................................ 16
Figure 2.5. Earth’s rotation velocity .............................................................................. 17
Figure 2.6. Roughness classes according to European Wind Atlas ............................ 19
Figure 2.7. Velocity in the near wall region .................................................................. 27
Figure 3.1. OpenFoam structure. [Source: OF User guide] .......................................... 31
Figure 3.2. OpenFoam class structure. ....................................................................... 32
Figure 3.3. OpenFoam case structure. ......................................................................... 33
Figure 3.4 Wall definition. [Source: OF User guide] ..................................................... 34
Figure 3.5. SnappyHexMesh case ............................................................................... 36
Figure 3.6 Unit numbers for the conversion. [Source: OF User guide] ......................... 37
Figure 4.1. Flat plane case. .......................................................................................... 39
Figure 4.2. Cross section of the axisymetric hill .......................................................... 39
Figure 4.3. Bump case ................................................................................................. 40
Figure 4.4. Flat plane case mesh. ................................................................................ 41
Figure 4.5. Velocity computed by OF and Fluent along vertical lines at ....................... 42
Figure 4.6. Velocity computed by OF and Fluent along vertical lines at ....................... 43
Figure 4.7. Velocity profiles computed by OF using 1st and 2nd order discretization .... 44
Figure 4.8. Velocity profile computed by OF vs Power-law expression (on the left) .... 45
Figure 4.9. View of surface mesh and cross section mesh for the bump case ............ 46
Figure 4.10. Inlet velocity profile for the bump case .................................................... 47
Figure 4.11 Experiment results for the separation centerline (x –y plane). [17] ............ 47
Figure 4.12. Separation along the centerline computed by OF 1st order (x –y plane). 48
Figure 4.13. Separation along the centerline computed by Fluent 1st order (x –y plane). 48
Figure 4.14 Separation along the centerline computed by OF 2nd order (x –y plane). 49
Figure 4.15. Separation along the centerline computed by Fluent 2nd order (x –y plane). 49
Figure 4.16 Separation along the centerline using k-ε RNG model with OF. .......... 50
Figure 4.17. Bump cross section (z-y plane) for x=0 .................................................. 50
Figure 4.18. Velocity profiles along five vertical lines .................................................. 51
Figure 4.19. Velocity distribution computed by OF in m/s ........................................... 55
Figure 4.20. Velocity distribution computed by OF in m/s ........................................... 56
Figure 4.21. Velocity distribution computed by OF in m/s ........................................... 57
Figure 4.22. Roughness map-file example .................................................................. 58
Figure 4.23. Flow chart of the procedure ..................................................................... 59
Figure 4.24. Angle calculation method a). ................................................................... 60
Figure 4.25 Angle calculation method b). .................................................................... 61
Figure 4.26. Inconvenience method a) ................................................................. 61
Figure 4.27. roughnessToFoam created roughness map................................. 62
Figure 4.28. Coriolis validation case.............................................................. 65
Figure 4.29. Trajectory of the wind flow affected by coriolis forces .............. 65
Figure 4.30. Trajectory of the wind flow affected by coriolis force ............... 67
Figure 4.31. Gravitational force validation case ........................................... 68
Figure 4.32. Static pressure depending on height (z) ..................................... 69
Figure 4.33. Bump case displayed with Global mapper and background mesh wireframe ................................................................. 71
Figure 4.34. Bump meshed with SnappyHexmesh ........................................ 73
Figure 4.35. Bump mesh Plane (y-z) ................................................................. 73
Figure 4.36. Detailed picture surface layers ................................. 73
Figure 4.37. Velocity profiles using Snappy and former mesh ...................... 74
Figure 4.38. Askervein hill ................................................................. 75
Figure 4.39. Askervein hill: Background mesh location .............................. 76
Figure 4.40. Askervein: Slice plane definition ........................................... 77
Figure 4.41. Sliced by plane e: Three layers case ....................................... 77
Figure 4.42. Sliced by plane e: Seven layers case ....................................... 78
LIST OF TABLES

Table 1.1. Vattenfall’s offshore and onshore power capacity and production.3 .......... 5
Table 2.1. Montavon obtained c values. ................................................................. 22
Table 2.2. Standard k-ε model constants. ............................................................... 25
Table 2.3. RNG k-ε model constants. ................................................................. 26
Table 4.1. K-ε standard turbulence model constants. ......................................... 40
Table 4.2. Boundary conditions. Flat plane. ..................................................... 41
Table 4.3. Velocity profile computed ................................................................. 45
Table 4.4. Boundary conditions bump case ..................................................... 46
Table 4.5. New files in order to implement roughness in OF. ......................... 53
Table 4.6. Boundary conditions coriolis validation case. ................................. 65
Table 4.7. Velocity results calculated by OF and theoretically in m/s ............... 66
Table 4.8. Time comparison using SnappyHexMesh vs Manual meshing .......... 75
NOWADAYS, MOST OF THE PRODUCED ENERGY IN THE WORLD IS OBTAINED FROM COAL, OIL AND OTHER FOSSIL FUELS. EXPERTS PREDICT THAT GLOBAL OIL SUPPLIES WILL ONLY MEET DEMAND UNTIL PRODUCTION PEAKS SOME TIME BETWEEN 2013 AND 2020, THOUGH OTHER EXPERTS ARGUE THAT IT COULD OCCUR EVEN SOONER (SALAMEH, 2003). ON TOP OF THAT, SUCH ENERGY SOURCES HAVE THE ADDITIONAL INCONVENIENCY OF CARBON DIOXIDE EMISSIONS TO THE ATMOSPHERE WHICH IS THE MAIN CONTRIBUTOR TO THE ONGOING GLOBAL WARMING.

THE INTERNATIONAL ENERGY AGENCY (EIA) PREDICTS THAT IN THE PERIOD 2008-2030 THE ENERGY DEMAND WILL INCREASE ABOUT 55% AND THE ELECTRICITY CONSUMPTION WILL BE DOUBLED.

THUS, A SERIOUS PROBLEM IS UPCOMING IN THE FOLLOWING YEARS: HOW TO INCREASE THE ENERGY PRODUCTION IN ORDER TO MEET DEMAND, WHILE DECREASING AT THE SAME TIME CO₂ EMISSIONS.

SEVERAL KIND OF OPINIONS ARE ABOUT THE DECISIONS MUST BE TAKEN, BUT ALL OF THEM AGREE IN ONE THING: SOMETHING MUST BE DONE AND IT MUST BE NOW AS THE TIME IS RUNNING OUT AND THE CONSEQUENCES CAN BE CATASTROPHIC.

IN THE AUTHOR’S OPINION THE ENERGY SOURCE THAT CAN MEET THE DEMAND IS THE NUCLEAR POWER WHICH DOES NOT EMIT CARBON DIOXIDE THEREFORE THE GLOBAL WARMING WOULD NOT BE AFFECTED. HOWEVER, THE ENTAILED RISKS AND THE UNCERTAINTY ABOUT THE NUCLEAR WASTE PROCESSING PROMPTED MOST OF THE GOVERNMENTS IN EUROPE (EXCEPT FROM A FEW EXCEPTIONS E.G. FRANCE) TO INVEST HEAVILY IN RENEWABLE SOURCES SUCH AS WIND, SOLAR OR BIO MASS INSTEAD AND RULE OUT CONTINUE INVESTING IN NEW NUCLEAR POWER PLANTS.

SOLAR ENERGY DEVELOPMENT HAS PRIMARILY LIMITED TO COUNTRIES WITH SUITABLE SPACE AND CLIMATE SUCH AS MEDITERRANEAN COUNTRIES AND USA.

IN THE NORDIC AREA, THE CLIMATE MAKES THE WIND AND WAVE POWER MORE SUITABLE AND THE LAST YEARS COUNTRIES SUCH AS DENMARK, SPAIN AND UK HAVE INVESTED CONSIDERABLY IN THESE FORMS OF ENERGY.

THE TOTAL ENERGY IN THE WINDWORLDWIDE, IS ESTIMATED TO BE ABOUT THREE TO FOUR TIMES THE TOTAL ENERGY NEEDED IN THE PLANET. IN 2008 THE WORLD INSTALLED WIND CAPACITY WAS 121 GW AGAINST 94 GW IN 2007, WHICH MEANS AN INCREASE NEARLY 30%. IF WE LOOK BACK A LITTLE BIT LONGER IN 2002 THE INSTALLED WIND CAPACITY WAS 31 GW. IN LIGHT OF THE ABOVE DATA, IT CAN BE SEEN THE HUGE AND RAPID SPREAD IT IS HAVING SUCH TECHNOLOGY ALL ALONG THE WORLD AND IT SEEMS CLEAR THAT IT WILL BECOME ONE OF THE MOST IMPORTANT ENERGY SOURCES IN THE FUTURE.[1]
1.2. Vattenfall

The birth of Vattenfall is marked by the restructuring of the Trollhättan canal and hydro power plant in 1909. The Swedish government had bought the water rights some years earlier and started to get involved in the rapidly developing electricity generation technology. During the first years of existence, some hydro power plants were built, for example in Olidan or Älvkarleby and in 1951 the world’s largest hydro power was inaugurated in Harsprånget. During these years grid lines also were put in operation being able to provide electricity in most parts of Sweden by the end of the 40s.

In 1975 with the inauguration of the first two nuclear reactors in Sweden, Ringhals 1 and 2 Vattenfall gets involved in the nuclear power generation and will build seven more during the 70s and 80s.

In 1992 Vattenfall suffers the most important change in its hundred-year-old history and is transformed from a state enterprise to the limited liability company Vattenfall AB.

In 1996 starts Vattenfall’s international expansion with the acquisition of the finish electricity distribution company Hämeen Sähkö and through the partnership with VASA which permitted Vattenfall begin working in the Germat market.

In the following years and up to now Vattenfall’s expansion around Europe has not stopped with several acquisitions mostly in Germany and Poland and recently in UK, Netherlands and Belgium.

Currently Vattenfall is Europe’s fifth largest electricity generator and the largest producer of heat. Vattenfall works in all parts of the electricity value chain, such as generation, distribution, transmission, sales as well as sells of heat. Nowadays, Vattenfall operates in Sweden, Denmark, Finland, Germany, Poland and UK. In addition, with the recent acquisition of the Dutch company Nuon, Vattenfall has also operations in the Netherlands and Belgium.

In the Nordic region Vattenfall is the leading energy group with a share about 20% in electricity production. Hydro and nuclear power are the base of the electricity production, while wind, waste and fossil fuels are also used. In addition, the group is the fourth largest heat producer in the Nordic region which is mainly based on biomass.

In Central Europe the operations are conducted in Germany and Poland. Vattenfall is the third largest electricity generator and the largest heat producer in Germany. In Poland the heat production and sales is the largest part of operations with a market share about 12 % of the heat production.

In the following picture the sources from which the energy was produced by Vattenfall can be seen in the last year 2008.
The net sales reached 15041 million euros\(^1\) in 2008 with an increase of 14% respect to 2007. The Group has approximately 33000 employees of which about 9500 work in the Nordic region and 22500, though, in Central Europe.

**Future**

Vattenfall's central principle for the following years is sum up in the following sentence: "Make electricity clean". The goal is to decrease drastically the carbon emissions in the near future and reach the zero emissions by 2050. Achieving this ambitious goal requires that Vattenfall reduces its CO\(_2\) emissions from the existing power plants and at the same time increases its generation capacity from renewable energy sources, such as wind power, bioenergy, ocean energy and nuclear power\(^2\).

On top of that Vattenfall is developing a new technology called CCS (Carbon Capture and Storage) which could make the power plants driven by fossil-fuel combustion became climate-neutral. Such technology is still under development and it is predicted to be commercially available by 2020.

The transformation will be gradual and step by step Vattenfall's production will shift to clean means of energy production. By 2030, Vattenfall expects to achieve a halving of emissions from existing operations from 1990 (base year for the Kyoto Treaty) that was about 680 gCO\(_2\)/ kWh.

---

\(^1\) Exchange rate SEK 10.94 = EUR 1

\(^2\) Although nuclear energy is not a climate-neutral energy source is usually refer to as because it does not generate CO\(_2\) emissions.
In the following picture the expected sources from which the energy will be produced by Vattenfall in 2030 can be seen.

![Vattenfall's energy sources prediction by 2030.](image)

As it can be seen, still some non-neutral operations are remaining but the expected improvement is incredible. In addition, it is important to take into account that the energy production capacity will increase too (from 163 TWh in 2008 to 390 TWh in 2030), thus, the increase in clean energies is even higher than the reflected by the percentages. The source which suffers the largest increase is clearly the wind power which in the next decades is going to be one of the bases of Vattenfall’s electricity production.

**Wind**

It is a long time Vattenfall started to install wind power capacity. In the early 1970s, the first wind power plant was installed at Älvkarleby in northern Uppland (Sweden), a small 60 kW generator. In the early 1980s, Vattenfall constructed the Näsudden I which years later and after some improvements will lead to the Näsudden II, the first wind power plant in the world to achieve an output of 60GWh (2004).

Currently, Vattenfall has a total about 900 MW installed wind power capacity with an energy production of 2.2 TWh per year. Most of the capacity is located in Denmark and Sweden, including the Nordic largest offshore wind farm, located outside Esbjerg in Denmark, which currently generates about 700 GWh of electricity annually. Recently Vattenfall made some acquisitions in the UK and signed a cooperation agreement with Scottish Power Renewables (owned by Iberdrola, the world’s largest wind power operator) to establish new offshore wind power.

Vattenfall’s goal as far as wind power production is concerned is to increase the production from the current 1.5 TWh to 49 TWh by 2030. The recent acquisitions carried out by Vattenfall in the UK signify a very important step forward toward
achieving this objective. In addition, Vattenfall continues identifying suitable locations (most of them over complex terrain as the best locations are already taken) to build land based wind power capacity in Sweden and Denmark with the aim of installing more than 500 wind power turbines with combined capacity of 1.5 TWh in the near future. The present project is involved in such target in order to make OpenFoam CFD tool be integrated into the based-siting.

In the following picture (Figure 1.3.) Vattenfall's wind power assets can be seen spread out in the Nordic area, UK and central Europe. In table 1.1. the capacity and electricity generation breakdown is represented.

![Figure 1.3. Vattenfall’s wind assets in June 2009.](image)

<table>
<thead>
<tr>
<th></th>
<th>Sweden</th>
<th>Denmark</th>
<th>UK</th>
<th>Belgium</th>
<th>Germany</th>
<th>Poland</th>
<th>Netherlands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onshore</td>
<td>34 MW</td>
<td>215 MW</td>
<td>34 MW</td>
<td>2 MW</td>
<td>13 MW</td>
<td>30 MW</td>
<td>165 MW</td>
</tr>
<tr>
<td>Offshore</td>
<td>130 MW</td>
<td>96 MW</td>
<td>90 MW</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>90 MW</td>
</tr>
<tr>
<td>Production</td>
<td>450 GWh</td>
<td>800 GWh</td>
<td>300 GWh</td>
<td>5 GWh</td>
<td>30 GWh</td>
<td>60 GWh</td>
<td>525 GWh</td>
</tr>
</tbody>
</table>

Table 1.1. Vattenfall’s offshore and onshore power capacity and production.³

³ The assets of the Dutch company Nuon (pending acquisition that will be carry out in the following months by Vattenfall) are included in the graph.
1.3. CFD for wind farm siting

Vattenfall is pursuing a huge expansion of onshore wind power capacity in the Nordic region. The most suitable sites are already taken, so more complex places are to be studied in order to search for new profitable emplacements.

Several factors must me taking into consideration when a site is being analyzed. First of all, and probably the most important factor is the mean velocity since the power production of a wind turbine scales with the cube of wind speed. In addition, it is imperative to analyze the prevailing high wind shear; turbulence levels caused by canopies, hilly surfaces, as well as electrical grid access, noise emission, site accessibility and so on.

The first step in order to assess the wind potential for a site is usually based on long-term onsite measurements. It is not easy to determine how long is necessary to be the data collection in order to get reliable results but usually it varies from a year to three years. For example Cherry (1980) found that a year collection data gave rise to an uncertainty of 12% in wind speeds, whereas if the data collection was extended to four years the uncertainty decreases to 7% in mean wind speed.

In light of above results, it can be seen that even with four year data collection (that is not common to have) the uncertainty is still quite high. On top of that, it must be taken into account that the available power is proportional to the velocity to the power three so the power uncertainty would be about 22%.4

In order to increase the accuracy of the power predictions wind tunnel and numerical models are used.

The most widely used numerical wind engineering tools are WindPRO and Wasp, which obtain resource predictions based on wind statistics and topography. These programs use linear solvers which make them be limited to more straightforward terrain regions and hill flows not predicting correctly recirculation and separated flows. Its reliability in simple terrains and easy use makes them a perfect for offshore potential assess.

Currently more complex fluid flow models are available which provide better accuracy and helps to mitigate the risk of failing in the wind farm emplacement being the computational demands higher than the linear solver case.

The procedure will be the following: Simulations are done using different wind situations (boundary conditions) and then combining results weighed by the likelihood of occurrence the wind statistic over the studied area can be obtained.

4 For more information about how to calculate the wind power look at Appendix A.1.
Ongoing project: Proposed CFD Methodology

The terrain data, orography as well as terrain characteristics such as roughness, canopies and so on, are provided in digitized format from a GIS system, e.g., WindPRO. Such information must be converted into the format needed by the CFD program is going to be used.

Thereafter a surface mesh conforming to the site orography is constructed for a domain that extends some beyond the region of interest. In order to base on steady-state CFD simulations the domain under study should be small enough for air to pass through much faster than the time-scale necessary to notice meteorological conditions changes.\(^5\) This limits the region of interest to no bigger than 20 x 20 km.

The mesh must be finer in special zones such as step changes or steep slopes where the gradients of the variables are higher. On this project two basic meshing approaches are considered: projection of predefined base mesh onto the surface and split hexahedral-cells from STL surface (SnappyHexMesh utility). The typical computational mesh may consist of several millions of cells.

Next, wind speed and direction can be specified in the inlet boundaries. The available onsite data will be inside the domain, accordingly a procedure will be carried out in order to infer the inlet boundary conditions from the onsite measurements.

The wind rose is broken down in several segments and then different wind speeds will be simulated according to Weibull\(^6\) wind distribution. The number of segments the wind rose is split up could be about 12 while the number of wind speed magnitudes could be between 5 to 10, which comes to 60 to 120 simulations. Such simulations, even using several computers in parallel, will take several weeks to finish. In the following picture such case example can be seen.

![Figure 1.4. CFD simulation example for wind farm siting.](image)

---

\(^5\) The time in which the meteorological conditions change is typically one hour according to Prospathopoulos (2007).

\(^6\) For more information about Weibull wind distribution look at Appendix A.1.
Commercial CFD tools are considered to be over-priced, based on antiquated numerics, and not sufficiently flexible when it comes to user-defined models, accordingly OpenFoam open source CFD tool will be developed in order to do the required simulations.
1.4. Aims and tasks

The scope of the present work is to develop CFD tools for sitting of wind turbine farms in complex terrain based on an open source flow modeling software: OpenFoam.

The proposed tasks are the following:

**Task 1**: CFD tool validation

Demonstration and validation of CFD simulations with the OpenFoam toolbox comparing the results with the commercial code Fluent. The toolbox will be applied for a number of cases in order to determine its reliability.

**Task 2**: Development of the current code (OpenFoam 1.5.)

Implementation of proper wall functions and submodels for developing a tool able to get accurate results in predicting the wind flow over complex terrain. This task involves:

- Implementation of a wall function with variable roughness over the terrain in the computational domain getting the data from a map file. Large variation of the surface roughness along the terrain of interest is likely, since the domain can span up to 20 x 20 km. In addition a pre-processing tool is to be configured in order to automate data conversion from the map file to the CFD toolbox.

- Comprise Coriolis force into the momentum equation. Coriolis force can significantly affect the flow due to the large region is going to be studied.

- Comprise Gravity force into the momentum equation. Gravity force can affect the flow field when buoyancy effects are being considered that although on this project are not included in the near future they will.

The new code is to be validated comparing the results with the commercial code Fluent.

**Task 3**: Preprocessing

Test and analyze the convenience in using OpenFoam’s tool SnappyHexMesh for meshing complex terrains.
1.5. Method

In order to develop this project, Linux and Windows operative systems are required as OpenFoam CFD tool is run in Linux whereas the commercial code Fluent is run in Windows. The latter will be used in order to compare and validate the results obtained by OpenFoam.

The preprocessing along most of the work will be done with Gambit (Fluent’s preprocessing tool) as OpenFoam’s pre-processing basic tool “blockMesh” is not developed enough and does not fulfill the requirements in order to create meshes fine enough for the purpose of this project. In the last part of the project SnappyHexMesh (OF’s post-processing tool) will be tried out in order to analyze the convinience of using it to mesh the domains of interest.

The post-processing results will be visualized in Fluent, even the simulations run with OpenFoam, since in Älvkarleby the cluster that is used in order to run Linux does not support Paraview (OF’s post-processing tool). For this purpose, the results obtained by OpenFOAM will need to be converted to Fluent format (.dat) with foamDataToFluent as it is described in Chapter 3.6. of the present project.
1.6. Lay out

This introduction sets out the subject area showing the need for developing the renewable energies and specially the wind power. A brief description of Vattenfall is done relating its evolution from its birth in the early 20th century to nowadays focusing on the wind power development, mostly in the last 30 years. Finally it is explained as well the longer project which the present work is in and its objectives.

The second chapter summaries on the one hand the physical properties of the atmospheric boundary layer (ABL) and on the other hand the numerical modeling of the ABL focusing on the aspects are going to be used later in this work.

The third chapter tries to be a small introduction to the open source CFD tool OpenFoam. Its main characteristics are explained as well as the main file and folders in order to be able to run the simulations were performed in this project. Some conversion data description is required as in this project as it was mentioned before, it is not just going to work with OF, but Gambit (as pre-processing meshing tool) and Fluent (as post-processing tool) will be used.

The fourth chapter is the main chapter of the present work since it resumes all the work that was developed within the 5 months that took to carry out the present project. It comprises four main subchapters: first the validation of OpenFoam comparing it with the commercial code Fluent, secondly the implementation of the roughness variable into the wall functions and the pre-processing tool in order to make easy to add the roughness values into the CFD tool, thirdly the addition of coriolis and gravitational forces into the momentum equation and eventually the analysis of the meshing tool SnappyHexMesh is performed.

Chapter fifth evaluates the results and findings of the project while the sixth chapter the path to follow in the future is set and some implementations are proposed.
2. THEORY

2.1. Atmospheric boundary layer (ABL)

The concept of boundary layer has been studied since the 19th century by scientists such as Freud or Prandtl, who recognized features closed to surfaces and the transition from the free stream conditions to the condition of no-slip at the wall. It has never been easy to define precisely what the boundary layer is but Stull (1988) for example defined it as “the part of the atmosphere that is directly influenced by the presence of the earth’s surface, and responds to surface forcings with a timescale of about an hour or less”. Stull mentions plenty of forcings perturbing the boundary layer, such as frictional drag, pollutants, evaporation, heat and so on. The depth of the atmospheric layer according to Stull could change from one hundred metres to few kilometers depending on the forcings before mentioned.

2.1.1. ABL structure and depth

The boundary layer is usually divided in two main sections. The outer region, widely known as the Ekman layer, in which the flow shows no much dependence on the surface, but it is dominated by pressure and coriolis force due to the rotation of the Earth; and the inner region which is characterized by the nature of the surface. The transition from the outer to the inner region is usually characterized by an overlap region. ABL structure can be seen in the following figure with approximate heights for each region.

![Figure 2.1. Atmospheric boundary layer structure.](image)
The depth and structure of the atmospheric boundary layer, though, can vary with atmospheric conditions and time of day so the values for depth presented in Figure 2.1 are just for guidance.

The lower part of the inner region is usually called dynamic sublayer which is a fully turbulent region close enough from the ground that the buoyancy and coriolis effects are negligible and far enough from the surface that is not affected by it. Under neutral conditions of the atmosphere the whole inner region behaves as a dynamic sublayer.

The region just above the surface where the flow suffers the influence of the surface directly is called the viscous sublayer.

### 2.1.2. Stratification and stability

If the fluid of the atmosphere is considered to be comprised of parcels of different densities, the tendency is to go up the ones that have lower density and go down those of higher density and the fluid is said to be stratified. If heavy fluid parcels are found below light parcels the system is referred as stable. However if the lightest parcels are below the highest the fluid is considered unstable, while if there is no density difference between parcels with height (no temperature difference) the fluid is said to be neutral and there is no stratification.

In order to define the stability, the potential temperature is defined, which is the temperature a parcel of fluid at pressure $P$ would acquire if adiabatically brought to a standard reference pressure $P_0$, usually 1 bar. The potential temperature is denoted $\theta$ and for the air is given by:

$$\theta = T \left(\frac{P_0}{P}\right)^{\frac{R}{C_P}}$$

Equ. 2.1

Static stability is divided into three categories:

- **Unstable** $\frac{\partial \theta}{\partial x} < 0$
- **Neutral** $\frac{\partial \theta}{\partial x} = 0$
- **Stable** $\frac{\partial \theta}{\partial x} > 0$

If a parcel is displaced from its original height its motion can be seen in the picture depending on the stability:
When the atmosphere is unstable a vertical displacement of air will be acted on to accelerate it upward. If the atmosphere is neutral, though, the particle will stay at the place it was displaced and eventually if it is stable the particle will be acted on to return to its original altitude.

This phenomenon can have a considerable influence on the wind flow. For instance, if we imagine the flow over a hill the deattachment just after the summit will be more or less important depending on the stability of the atmosphere. For the stable case the air flow keeps attached to the ground before and after the top, while for neutral and unstable cases separation takes place (for unstable extremely vertical flow can happen)

![Figure 2.3 Influence of the stability on the flow over a hill.][9]

Neutral flow can be approximated to windy conditions with cloud cover, whereas stably stratified flow occurs mostly at night and unstable at very hot sunny days.

Neutral conditions are not very common and are usually the transition between stable and unstable conditions. However, if the wind speed is high enough, the vertical displacement can be neglected, as the velocity in this way is much lower than the main flow. Under neutral conditions the whole surface layer behaves as dynamic sublayer.

In this work, neutral conditions will be considered and density is considered to be constant with height. In the future, the effects of the stability should be taken into account in order to get more accurate results.

### 2.1.3. Turbulence

Most of the flows that are found in engineering practice, even the more simple ones become unstable above a certain Reynolds number and since it was to be expected the ABL is not an exception. Usually three regimes are defined: laminar, transitional and turbulent.
At values above the so-called critical Reynolds number ($Re_{crit}$) the flow is random and chaotic and it is continuously changing with the time, such region is the one that is called turbulent regimen. In the following chart we can see a typical velocity measurement in turbulent flow:

![Figure 2.4. Velocity in a turbulent flow. [10]](image)

The fluctuations (seen in the above picture) have always three-dimensional character even when the velocity is unidimensional or bidimensional and the fluid particles move very fast and in all directions. Therefore mass, heat and momentum exchanges are very effective in turbulent flows.

Moreover, in turbulent flows rotational flow structures happened too, the so-called turbulent eddies with a wide range of length scales. The largest eddies are dominated by inertia effect and viscous effects are negligible. They extract energy from the main flow (vortex stretching) which creates motion that makes them interact and fission into smaller and smaller eddies generating the so-called energy-cascade which finishes when the energy contained by the smallest eddies is dissipated by viscosity heating up the flow and surroundings.

### 2.1.4. Governing equations

The main governing equations for the fluid flow are the mass conservation and momentum equation, which together constitute Navier-Stokes equations.

In general the conservations of the mass in Cartesian coordinates can be written as

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0$$  \hspace{1cm} \text{Equ. 2.2}

and the conservation of momentum can be written as

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} = - \frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j} \left( \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right) + S_M$$  \hspace{1cm} \text{Equ. 2.3}

where $S_M$ is referred to external forces which are applied such as gravity, density forces or coriolis.
2.1.4.1. Coriolis force

Moving objects on the Earth’s surface suffer a force which is called Coriolis force and it is caused by the movement of the object at the same time as the earth is rotating around its axis. Its components can be derived from the following diagram where \( \varphi \) is the latitude of the area of study.

The earth’s rotation velocity (\( \vec{\omega} \)) has three components, the East-West component, the North-South component (\( C_h = |\vec{\omega}| \cos \varphi \)) and the vertical component (\( C_v = |\vec{\omega}| \sin \varphi \)). In the figure above, it can be appreciated that the East-West component is zero.

The coriolis acceleration can be computed as:

\[
2 \omega \times u = \begin{vmatrix}
0 & \omega \cos \varphi & \omega \sin \varphi \\
u & v & w \\
\end{vmatrix} = (2\omega \cos \varphi - 2\omega \sin \varphi)i + (2\omega \sin \varphi)j - (2\omega \cos \varphi)k
\]

Equ. 2.4

where \( u, v \) and \( w \) are the velocity components of the wind flow in Cartesian coordinates.

\( \omega \) is the rate of rotation of the Earth, that is, the speed at which the Earths completes one rotation. If we consider that it rotates once every 23 hours 56 minutes and 4 seconds (Sidereal Day) \( \omega \) is equal to \( \frac{2\pi}{(86164 \text{ sec.})} = 7.2921 \times 10^{-5} \text{ s}^{-1} \).

In order to be included in Navier-Stokes equations the above presented acceleration will have to be multiplied by the density so \( SM = -2\rho \omega \times u \).

In order to evaluate the influence of the coriolis force on the wind flow a so-called Rossby number which is defined as follows:

\[
R_o = \frac{U}{L \, 2\omega \sin \varphi}
\]

Equ. 2.5

Where \( L \) is the length scale.
When Rossby number is large enough \((R_o >> 1)\) coriolis force can be neglected since centrifugal force is dominant on the wind flow behaviour. However, if \(R_o\) is around one the wind flow will be strongly affected by coriolis force.

The present project is focused on emplacements situated in the Nordic Region so let’s consider for example Stockholm’s latitude \((59.283^\circ)\), a 15 x 15 km domain and a moderate wind speed of 7.5 m/s. Rossby number for such case would be about 4 a value with which we are not allowed to say either that can be neglected or that its influence is important. Thus, it is decided to check if the coriolis force should be included in the model running a simulation with OpenFoam trying to analyze the impact due to such force.

### 2.1.5. Time-Average Transport Equations

Solving the time-dependent Navier-Stokes of fully turbulent flows with present day computing power is feasible for just very simple cases where the Reynolds number is quite low (laminar regime). Most of the times the engineer face up to cases which would require much more computing power that is available nowadays so some time must await for developments in computer hardware in order to solve directly Navier-Stokes equations.

In the meantime, accurate results are needed but without being necessary to describe each and every eddy in the flow. For this purpose, time-average transport equations (RANs) are used.

Reynolds (1895) proposed that the \(u_i\) velocity as it was presented in figure 2.4. can be decomposed into a mean component \(\bar{U}_i\) and time varying fluctuation with zero mean value \(\bar{u}_i\) representing the turbulent variations.

\[
\bar{u}_i = \bar{U}_i + \bar{u}_i
\]

The Navier Stokes equations (Equation 2.3.) are transformed using the velocity decomposition into mean and fluctuating velocity (Equation 2.5.) and the following is got:

\[
\frac{\partial (\rho \bar{u}_i)}{\partial t} + \frac{\partial}{\partial x_j} \rho (U_j \bar{u}_i) = - \frac{\partial \rho}{\partial x_i} + \frac{\partial}{\partial x_j} \left( \mu \left( \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) - \rho \bar{u}_j \bar{u}_i \right) + S_M
\]

Equ. 2.7

As it can be noticed, six extra unknown terms showed up on the right-hand side of the momentum equation, which are known as the Reynolds stresses. Such unknown source terms are the reason why the turbulence model are needed, in order to predict the Reynolds stresses accurately and not using too much computational resources as the full resolution of the equations would require vast resources which right now are not even available.
2.1.6. Roughness

The roughness of an area is determined by the size and distribution of the roughness elements.

The roughness length, $z_0$, is the height above the ground where the flow velocity is zero. Although $z_0$ is slightly dependent on the flow it is widely accepted to be constant and so it will be assumed.

There is another variable which is used to define the roughness as well, the class which is defined as:

$$\text{Roughness class} = \begin{cases} 
1.6998 + \frac{\ln(z_0)}{\ln(150)} & \text{if } z_0 \leq 0.03 \\
3.9125 + \frac{\ln(z_0)}{\ln(3.333)} & \text{if } z_0 > 0.03 
\end{cases}$$

The following table presents the existing roughness lengths and classes according to the European Wind Atlas.

<table>
<thead>
<tr>
<th>Roughness Class</th>
<th>Roughness height [m]</th>
<th>Landscape Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0002</td>
<td>Water surface</td>
</tr>
<tr>
<td>0.5</td>
<td>0.0024</td>
<td>Completely open terrain with a smooth surface, e.g. concrete airport runways or mowed grass.</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>Open agricultural area without fences and hedgerows and very scattered buildings. Only softly rounded hills</td>
</tr>
<tr>
<td>1.5</td>
<td>0.055</td>
<td>Agricultural land, some houses, and 8m. tall sheltering hedgerows with a distance of approx. 1250 meters</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>Agricultural land, some houses, and 8m. tall sheltering hedgerows with a distance of approx. 500 meters</td>
</tr>
<tr>
<td>2.5</td>
<td>0.2</td>
<td>Agricultural land, many houses, shrubs and plants, or 8m. tall hedgerows with a distance of approx. 250 meters</td>
</tr>
<tr>
<td>3</td>
<td>0.4</td>
<td>Villages, small towns, agricultural land with many or tall hedgerows, forests, or very rough and uneven terrain</td>
</tr>
<tr>
<td>3.5</td>
<td>0.8</td>
<td>Larger cities with tall buildings</td>
</tr>
<tr>
<td>4</td>
<td>1.6</td>
<td>Metropolitan areas with tall buildings and skyscrapers</td>
</tr>
</tbody>
</table>

Figure 2.6. Roughness classes according to European Wind Atlas.

2.1.7. Velocity profile approximations in the dynamic sublayer

The dynamic sublayer as it was said before is a turbulent layer which is close enough to the surface that the Coriolis and buoyancy forces due to density stratification are negligible and at the same time is far enough from the surface in order to be influenced by the surface. Although they have not been proved theoretically there are some velocity
profiles formulation that are known to fit quite well to experimental results and three of them are presented below.

### 2.1.7.1. Logarithmic velocity profile

It is the most well-known and accepted velocity profile approximation in the dynamic sublayer. The law was developed in the early 20th century and from then on it has been widely used.

The wind speed change with elevation was introduced by Prandtl (1932) and is usually expressed as:

\[
\frac{\partial U}{\partial z} = \frac{u_*}{kz}
\]

Where \( u_* \) is the friction velocity and \( k \) von Karman’s constant.

The logarithmic wind profile equation is got directly from the integration of Equ. 2.7 over height from \( z=z_0 \) to any height \( z \).

\[
\frac{U}{u_*} = \frac{1}{k} \ln \frac{z}{z_0}
\]

Equ. 2.9

The effects of the atmospheric stability could be added by including an extra term into the profile:

\[
\frac{U}{u_*} = \frac{1}{k} \left[ \ln \frac{z}{z_0} - \zeta \right]
\]

Equ. 2.10

Where \( \zeta \) is a stability dependent function, positive if the atmosphere is unstable and negative for stable conditions.

### 2.1.7.2. Power-law expression

In some cases serious mathematical difficulties are presented when trying to calculate logarithmic velocity profiles, therefore, sometimes simpler approximations are required in order to facilitate the calculations.

The power-law expression is defined as:

\[
\frac{v}{V_0} = \left( \frac{h}{H_0} \right)^m
\]

Equ. 2.11

where

- \( H_0 \) is the reference height which must be smaller or equal to the height at which the wind speed is maximum.
- \( V_0 \) is the velocity at the reference height \( H_0 \).
m is a constant coefficient.

The expression above can be written as:

\[
\ln \left( \frac{V}{V_0} \right) = m \ln \left( \frac{h}{H_0} \right)
\]

which is a linear function \(y = m \cdot x + b\) with \(y = \ln \left( \frac{V}{V_0} \right)\), \(x = \ln \left( \frac{h}{H_0} \right)\) and \(b = 0\).

Most of turbulence studies point to \(m = 1/7\) as a typical value.

Although there is no theoretical justification of the power law, the truth is that it fits mean wind profiles well when suitable parameter for \(m\) is chosen.

### 2.1.7.3. Alexandrou’s formulation

To end the third and last velocity profile formulation is presented.

Using the boundary layer thickness \((h)\), Alexandrou (1996) proposed the following velocity profile over a flat plate with zero pressure gradient.

\[
\frac{u}{u^*} = \frac{1}{k} \left[ \ln \left( \frac{\delta}{\delta + c} \right) - \left( \frac{c}{1 + c} + a \right) \frac{\delta^2}{2} + a \delta + B - \ln \left( k \right) \right]
\]

where

\[
\delta = z/h
\]

\(k\) is von Karman constant \((=0.4178)\)

\(a, c\) and \(B\) are constants of the model.

In order to ensure that the velocity profile is logarithmic close to the ground, \(a\) and \(c\) must satisfy \(a = 1/c\). In addition, at height \(z = h\) the velocity will be \(U_\infty\) and from this assertion the following relation between \(B, u, h, a\) and \(c\) can be obtained:

\[
B = \frac{u_\infty}{u^*} k - \ln \left( \frac{1}{1+c} \right) + \frac{1}{2} \left( \frac{c}{1+c} + a \right) - a + \ln k
\]

For each case, the \(c\) constant value must be calculated, as though it will be probably quite close from other cases’ values changes can take place.

For a boundary layer developing over a flat smooth terrain, Alexandrou got that the value for \(c\) that best reflect fit with experimental results was 0.166.

Montavon (1998), though, got the following results for \(c\) depending on the roughness of the terrain.
It can be noticed that the $c$ is almost constant whatever $z_0$ is.

Montavon points out that the differences for the $c$ values can be due to the fact that Alexandrou had the friction velocity ($u^*$) as an additional parameter to be fitted (the same as $c$ or $h$), whereas Montavon, computed it with a CFD program (CFX). Considering that a change of 1\% in the friction velocity gives rise to a 3\% change in the value of the constant, the difference of almost 10\% could be easily explained.
2.2. Numerical Modeling

In section 2.1 the theory of the ABL was discussed and now the models and codes used in order to simulate the wind flow, looking at accuracy and ease of implementation will be commented.

A large number of numerical models are available, from simple linear solvers to direct numerical solutions. Some of the most important ones are briefly described:

a) **Linear models:** They contain simple turbulence and roughness models, providing fast and accurate results for simple cases. The code starts not to be accurate when the terrain is hilly and it is known to poorly predict flow separation and recirculation.

b) **RANS:** As it was mentioned before, they involve the solution of the time-averaged Navier-Stokes equations with the whole scales of turbulence being modelled. It is the most well-known and used approach for practical engineering applications.

c) **Large Eddy Simulation (LES):** It was developed based on the paper released by Smagorinsky (1963). The larger scales, which contain most of the energy, are computed while the smaller scales, which are thought to be more predictable are modeled, instead.

d) **Detached Eddy Simulations (DES):** It is a modification of the RANS, where RANS model is used and only separated regions are modelled by LES. This hybrid model between RANS and LES permits the user to have the advantages of both programmes at the same time.

e) **Direct Numerical simulations (DNS):** It involves the numerical solution of the instantaneous equations that govern fluid flows. So far only a few simple cases have been solved using DNS, as there is no enough computational power.

As it can be thought the more turbulence scales are computed instead of modelled the more time it will be needed to solve the problem and the required computational resources will be higher as well.

While DNS is the ideal method, when deciding which kind of model is being used we need to strike a balance between accuracy and computational resources. DNS are unlikely to become feasible in the foreseeable future, and the difficulties in modelling Reynolds stress models (RANS) leaves LES as the best option but even LES requires computational resources that nowadays are not available although in the near future will be.

Thus, while computational resources are not good enough in order to use LES the work must focus on developing RANS models and making them work better day by day as it will benefit not only users of RANS, but the future users of LES too.
2.2.1. Turbulence modeling

When RANS equations were presented in 2.1.5, it was highlighted that after transforming Navier Stokes equations six unknown terms were created, so-called Reynolds stresses. In order to be able to solve the case, new extra models need to be included in order to model such six new terms. A number of models are available but in this section just two will be presented, the most well-known one, the standard k-epsilon model and the k-epsilon RNG, alternative model based on the standard.

2.2.1.1. Boussinesq approximation

Boussinesq (1877) suggested that the Reynolds stresses were related directly to the main stress which is based on the idea that viscous and Reynolds stresses have similar effects on the mean flow and can be written for incompressible flows:

\[ \tau_{ij} = -\rho u_i u_j = \mu_t \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad \text{Equ. 2.15} \]

where \( \mu_t \) is the eddy or turbulent viscosity (dimensions Pa·s) which is used to model the additional viscosity due to the turbulence in the flow being the total turbulence the sum of the viscosity and the eddy viscosity (\( \mu_{TOTAL} = \mu + \mu_t \)).

The modeling can be now completed if this turbulent viscosity can be obtained from other variables.

Although Boussinesq approximation is widely accepted, it is not useful for every case since in when there are sudden changes in mean strain rate are Reynolds stresses are poorly predicted by models using Boussinesq approximation. Such sudden changes makes the Reynolds stresses adjust at a different rate as the mean flow processes do so the approximation fails. This occurs for instance for flow over curved surfaces, three dimensional flows and flow with separation.

2.2.1.2. K-\( \varepsilon \) standard model

K-epsilon model is a two-equation model where two new variables are introduced, the energy dissipation per unit mass and the kinematic energy with which the turbulent viscosity will be calculated. The model was developed originally by Hanjalic and Launder (1972).

The kinetic energy per unit mass can be defined as:

\[ k = \frac{1}{2} \left( \overline{u'^2} + \overline{v'^2} + \overline{w'^2} \right) \quad \text{Equ. 2.16} \]

K and \( \varepsilon \) are used in order to define a velocity and length scale:

\[
\text{Velocity scale } \theta = k^{\frac{1}{2}}
\]
Length scale \( l = \frac{k^{3/2}}{\varepsilon} \)

which leads to the eddy viscosity defined as follows:

\[
\mu_t = C_\mu l \varepsilon = C_\mu \rho \frac{k^2}{\varepsilon}
\]

Equation 2.17

where \( k \) and \( \varepsilon \) are the subject of the transport equations for incompressible flow:

\[
\rho \frac{\partial k}{\partial t} + \rho \frac{\partial (k U_i)}{\partial x_j} = \tau_{ij} \frac{\partial U_i}{\partial x_j} - \rho \varepsilon + \frac{\partial}{\partial x_l} \left[ \left( \mu + \frac{\mu_t}{\sigma_\varepsilon} \right) \frac{\partial k}{\partial x_l} \right]
\]

Equation 2.18

\[
\rho \frac{\partial \varepsilon}{\partial t} + \rho \frac{\partial (\varepsilon U_i)}{\partial x_j} = C_{\varepsilon_1} \frac{\varepsilon}{k} \tau_{ij} \frac{\partial U_i}{\partial x_j} - C_{\varepsilon_2} \rho \frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_l} \left[ \left( \mu + \frac{\mu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_l} \right]
\]

Equation 2.19

The values for the model constants are the following:

<table>
<thead>
<tr>
<th>( C_\mu )</th>
<th>( C_{\varepsilon_1} )</th>
<th>( C_{\varepsilon_2} )</th>
<th>( \sigma_k )</th>
<th>( \sigma_\varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.09</td>
<td>1.44</td>
<td>1.92</td>
<td>1.0</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 2.2. Standard k-\( \varepsilon \) model constants.

After the K-epsilon model many other models based on it were developed in order to try to improve it and increase in accuracy such as the K-\( \varepsilon \) RNG model or the low Reynolds numer k-\( \varepsilon \) model.

### 2.2.1.3. K-\( \varepsilon \) RNG model

The k-\( \varepsilon \) RNG model is an alternative turbulence model which was proposed in 1986 by Yakhot and Orzag. The transport equations for turbulence generation and dissipation are the same as those for the standard model, but the model constants are different.

\[
\rho \frac{\partial \varepsilon}{\partial t} + \rho \frac{\partial (\varepsilon U_i)}{\partial x_j} = (C_{\varepsilon_1} - C_{\varepsilon_{1RNG}}) \frac{\varepsilon}{k} \tau_{ij} \frac{\partial U_i}{\partial x_j} - C_{\varepsilon_2} \rho \frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_l} \left[ \left( \mu + \frac{\mu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_l} \right]
\]

Equation 2.20

where

\[
C_{\varepsilon_{1RNG}} = \frac{\eta (1 - \eta / \eta_0)}{1 + \beta \eta^3}
\]
\[
\eta_0 = 4.377 \\
\beta = 0.0012
\]

The values for the model constants are the following:

<table>
<thead>
<tr>
<th>(C_\mu)</th>
<th>(C_{c1})</th>
<th>(C_{c2})</th>
<th>(\sigma_k)</th>
<th>(\sigma_\varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.085</td>
<td>1.42 - (C_{c1})RNG</td>
<td>1.68</td>
<td>0.7179</td>
<td>0.7179</td>
</tr>
</tbody>
</table>

Table 2.3. RNG k-\(\varepsilon\) model constants.

The extra term in the dissipation transport equation improves the performance for separating flow and recirculation regions.

The simulations take a little longer time and although it may provide improved results in some applications, it may also reduce accuracy in others.

Some studies such as Kim and Patel (2000) or Jeong (2002) were made some years ago comparing turbulence models in regions of complex terrain and indicated that the k-\(\varepsilon\) RNG model was the 2-equation turbulence model which best agreed with experimental results.

### 2.2.2. Discretization schemes

The value for the variables is usually stored at the cell center so if the value at the face is needed (e.g. convection term), it must be computed from the interpolation of cell center values. There are several schemes that can be used in order to interpolate, but in this section only the two which are used on this project are described: First - order and Second-Order Upwind discretization. As the order of the scheme is increased, the accuracy is increased but being increased the required computational resources as well.

**First order upwind discretization:** It is considered that the variable is constant inside the cells, thereby, the value at the face is the same as the value at the cell center.

**Second order upwind discretization:** It is considered that the variable changes linearly inside the cell, thereby, the following formula is used in order to get the value at the face:

\[
\theta_{face} = \theta_{center} + \nabla\theta \cdot \hat{n}
\]  
Equ. 2.21
As it can be seen such method requires a previous gradient calculation ($\nabla \theta$). $\vec{n}$ is referred to the displacement vector from the upstream cell centroid to the face centroid.

2.2.3. Near wall treatment: Wall functions

It is widely accepted that the near-wall region can be subdivided into three main layers. There is a sublayer so-called “Viscous sublayer” where the flow is almost laminar and the viscosity determines the flow behaviour. In the outer layer, the flow is fully turbulent and in this case the turbulence is the one that plays a dominant role in momentum and heat transfer. Eventually, there is an intermediate layer (Buffer layer) where the effects of viscosity and turbulence are both important. In the following figure can be seen the above presented subdivision of the near-wall region.

![Figure 2.7. Velocity in the near wall region.](image)

The natural way to treat wall boundaries is to mesh fine enough so that the gradient of the flow variables is not big. Often, when computing three-dimensional flow, that requires too much computer resources, thus, a way must be found in order not to lose too much accuracy while saving computational resources.

The solution is to use the so-called wall functions with which the viscous sublayer is bridged using empirical formulae. However, the formulations give rise to inaccuracy and becomes less adequate when the problems starts to increase in complexity such as cases of separation, reattachment and so on.

2.2.3.1. Smooth surfaces

In OpenFoam the following functions are used for smooth surfaces (very similar to Fluent):

\[
U^+ = \frac{1}{\kappa} \ln (E y^+) \quad \text{Equation 2.22}
\]

\[
y^+ = y \left( \frac{\rho C_p^{1/4}}{\mu^{1/2}} \right) \quad \text{Equation 2.23}
\]
The log-law is employed when $y^* > y_{\text{lam}}$ which is the $y^*$ value for which the log-law and laminar flow curves cross each other. In OpenFoam $y_{\text{lam}}$ is equal to 10.97 and in Fluent, though, is equal to 11.225.\(^7\)

The turbulence viscosity is recalculated using the following formula:

$$\mu_t = \mu \left( \frac{y^*}{\ln (E y^*)} - 1 \right)$$  \hspace{1cm} \text{Equ. 2.24}

The $k$ transport equation is solved in the whole domain including the wall-adjacent cells, while the $\varepsilon$ transport equation is not solved at the wall-adjacent cells, but instead is computed using the following equation:

$$\varepsilon = \frac{C_\mu^{3/4} k^{3/2}}{k_y}$$  \hspace{1cm} \text{Equ. 2.25}

However if the $y^* < y_{\text{lam}}$ the flow is considered to be laminar and $U^*$ is equal to $y^*$.

### 2.2.3.2. Rough surfaces

For rough surfaces some modifications are to be made in order to considerate the roughness. In the current version of OpenFoam (OF-1.5), the roughness is not included so it is decided to include them trying to base on formulation used in Fluent as it has been already validated and proved to work properly.

The surface roughness is defined by means of two variables: the roughness height ($K_s$) and the roughness constant ($C_s$). Roughness heights were presented in table 2.6. and as far as the roughness constant is concerned there are no guidance values. In Fluent’s manual is said that it varies from 0.5 to 1 and the 0.5 corresponds to sand-grain roughness. It is recommended that if the model departs much from the sand-grain roughness the constant roughness should be adjust. The adjustment should be done comparing the results with experimental data.

The implemented functions are the following:

$$U^* = \frac{1}{\kappa} \ln (E y^*) - \Delta B$$  \hspace{1cm} \text{Equ. 2.26}

$$y^* = y \frac{C_\mu^{1/4} k^{1/2}}{\mu}$$  \hspace{1cm} \text{Equ. 2.27}

Where $\Delta B$ depends on the type and size of the roughness and it is computed by the formulas proposed by Cebeci and Bradshaw. The regime is subdivided into three regimes. In order to define the regimes a non-dimensional roughness height is defined as follows:

---

\(^7\) Look at Appendix A.2. for more information about how to calculate $y^*$ laminar.
2. THEORY

\[ K_s^+ = \frac{\rho K_s C_{\mu}^{1/4} k^{1/2}}{\mu} \quad \text{Equ. 2.28} \]

And the regime is broken up in three:

- **Hydrodynamically smooth surface (\( K_s^+ > 2.5 \))**:

  \[ \Delta B = 0 \quad \text{Equ. 2.29} \]

- **Transitional regime (2.5 < \( K_s^+ < 90 \))**:

  \[ \Delta B = \frac{1}{\kappa} \ln \left[ \frac{K_s^+ - 2.25}{87.75} + C_s K_s^+ \right] \sin\{0.4258(\ln K_s^+ - 0.811)\} \quad \text{Equ. 2.30} \]

- **Fully rough regime (\( K_s^+ > 90 \))**:

  \[ \Delta B = \frac{1}{\kappa} \ln (1 + C_s K_s^+) \quad \text{Equ. 2.31} \]

The turbulence viscosity is recalculated using the following formula:

\[ \mu_t = \mu \left( \frac{\gamma K}{\ln \left( E_{v^*} / e^{k\Delta B} \right) - 1} \right) \quad \text{Equ. 2.32} \]
3. CFD: OpenFoam

Computational fluid dynamics is the analysis of systems involving fluid flow, heat transfer, chemical reactions, etc... by means of computer based simulations. Computers are used to perform the high amount of calculations required for solving the fluid dynamics problems, however nowadays even with the best super-computers in some cases just approximations can be got.

Before the 90s CFD programs were not widely used amongst the industrial community but the last 20 years due to the availability of high performing software and powerful computers the use of such programs has increased rapidly.

CFD packages usually comprise three main elements: a pre-processor, a solver and a post-processor. In the pre-processing step the mesh is created as well as the boundary conditions and fluid properties are set up. Thereafter, the solver will make the flow analysis and to end in the post-processing part the calculated results will be displayed with versatile data visualization tools.

3.1. OpenFoam Introduction

OpenFoam (Open Field Operation and Manipulation) is an open source computational fluid dynamics program produced by OpenCFD Ltd. Its origins come from the late 1980s at Imperial College (London) when a group of people decided to develop a powerful tool to be able to run flexible general simulations. It was released open source in 2004 under the GNU General Public License.

OpenFoam is written in C++ and it is used an object oriented approach which makes the code to be understandable and permits the user to implement its own files in order to adapt it to each specific case. Basically, OpenFoam is a library used to create executables, known as applications, in order to solve the case the user is working with. There are two kind of applications: solvers, that are to solve cases; and utilities, that are designed for data transformation. The overall structure of OpenFoam is shown in the following figure.

![Figure 3.1. OpenFoam structure. [Source: OF User guide]](image-url)
3.2. Programming language of OpenFoam

In order to understand and be able to work and modify OpenFoam libraries, some knowledge of C++ is required. As it was said before, in OpenFoam object oriented approach is used which makes the code quite easy to understand in order to be able to make it suitable for the user cases.

3.2.1. File structure & compiling

OpenFoam as every library designed with C++ is comprised by classes. The definition of such class is through a set of object construction and class member functions. Each class will contain two files, one for the definition (.C extension) and the other one for developing the class (.H extension). A class can use previous created classes which will be recognized by the compiler including the line: #include “name_of_the_class.H”.

In Linux systems standard “make” utility is currently used as compiler. However, OpenFoam is supplied with an improved version called “wmake” which is based on “make” but is easier to use and considerably more versatile. Every piece of code we include or change need to be compiled in order to be recognized by OpenFoam. In order to do it we will need to include a folder called “Make” in the class directory containing two files, “options” and “files”. In the following figure we can see an example of a class called App.

![OpenFoam class structure diagram](image)

The Make/option file contains the full path to locate header files (other classes) and library names. The syntax for headers is –I<path>, while for libraries is –L<libraryPath> and the Make/files file instead contains the name of the class is being compiled and its full path.

If an already existing class after having made some modifications it is wanted to be recompiled, old dependencies must be removed first. In order to do that wclean is typed in the command line and once all the old dependencies are being deleted it can be recompiled.
3.2.2. Equation representation

One of the most important strengths of OpenFoam is that the syntax of the equations in the solver applications resembles strongly the differential equations. For example the Momentum equation for a simple incompressible laminar flow, in vector form would be:

\[
\frac{\partial \mathbf{U}}{\partial t} + (\mathbf{U} \cdot \nabla) \mathbf{U} - \nabla \cdot (\nu \nabla \mathbf{U}) = - \frac{1}{\rho} \nabla \mathbf{p}
\]

Is represented in icoFoam solver by the code

```
Solve
(
    fvm::ddt(U)
    + fvm::div(phi,U)
    - fvm::laplacian(nu,U)
    ==
    - fvc::grad(p)
);
```

where phi is the volume velocity flux defined on the faces of each cell.

3.3. OpenFoam cases

OpenFoam is provided with a set of example cases which are located in $FOAM_TUTORIALS and are divided in folders depending on the solver is used in the example. Usually when a user wants to run a case, he/she will copy the source case most resembles the case he/she wants to run and later on will make the modifications that are needed. The basic structure for a OpenFoam case is the following:

![OpenFoam case structure diagram]

Figure 3.3. OpenFoam case structure.
3.3.1. System folder

Three files are placed in this folder: controlDict, fvSchemes and fvSolution.

In controlDict I/O data and running time parameters are set, including start/end time and time step. (See an example in the enclosed DVD in directory “cases/Coriolis/system”)

In fvSchemes file the numerical schemes for terms are set. In this file it will be said how the laplacian, divergence and so on are solved numerically. On this work, first and second upwind discretization orders will be used. (See an example in the enclosed DVD in directory “cases/Coriolis/system”).

In fvSolution file equation solvers, convergence tolerances, relaxation factors and other algorithm controls will be set. (See an example in the enclosed DVD in directory “cases/Coriolis/system”).

3.3.2. Constant folder

Two files (transportProperties, turbulenceProperties) and a folder (polyMesh) are placed in this folder.

In transportProperties the properties of the fluid we are working with are included, e.g. viscosity of the fluid. On this project air will be the using fluid. (See an example in the enclosed DVD in directory “cases/Coriolis/constant”).

In turbulenceProperties the turbulence model is going to be used and its parameters are set. (See an example in the enclosed DVD in directory “cases/Coriolis/constant”).

In polyMesh folder all the data for the mesh is located (points, edges, faces …). This folder can be created in different ways: by hand, transforming the data from other software to OF or using OF Pre-applications (snappyHexMesh, blockMesh…). There is a so-called ‘ boundary’ file where boundary walls are defined and the number of faces which are divided in is written. The following wall definitions are permitted:

<table>
<thead>
<tr>
<th>Selection Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch</td>
<td>generic patch</td>
</tr>
<tr>
<td>symmetryPlane</td>
<td>plane of symmetry</td>
</tr>
<tr>
<td>empty</td>
<td>front and back planes of 2D geometry</td>
</tr>
<tr>
<td>wedge</td>
<td>wedge front and back</td>
</tr>
<tr>
<td>cyclic</td>
<td>cyclic plane</td>
</tr>
<tr>
<td>wall</td>
<td>wall (used for wall functions in turbulent flows)</td>
</tr>
<tr>
<td>processor</td>
<td>inter-processor boundary</td>
</tr>
</tbody>
</table>

Figure 3.4 Wall definition. [Source: OF User guide]
3.3.3. 0 folder

The boundary conditions are included in this folder. Depending on the turbulence model we are working with, different boundary condition files will need to be included in addition to typical boundary conditions such as velocity, pressure and so on. For instance if we are using simpleFoam (steady-state, turbulent flow) turbulence boundary conditions will need to be included too (in our case \( k \) and \( \varepsilon \)).

The boundary conditions will be set for each wall and the structure will be the following:

\[
\text{Name of the wall} \\
\{ \\
type \quad \text{Type of wall} \\
value \quad \text{Value} \\
\text{Wall, patch, symmetryPlane…} \\
\text{In case it is necessary} \\
\text{(depending on the type)}
\]

Depending on the wall definition at constant/polyMesh/boundary the boundary condition type will be set. For each wall definition there are some values available that can be found in OpenFoam User Guide Table 5.2.

3.3.4. Time directories

It contains files of data for particular fields such as initial values and boundary conditions or results. In this project as we said before we will work with steady-state models so ‘time directories’ folder will not be included in the case folder.

3.4. Standard solvers and libraries

The solvers are placed in the $FOAM_APP/solvers directory and are divided into directories by category of continuum mechanics, combustion and solid body stress analysis. The list of solvers distributed with OpenFoam can be found in OF User Guide Table 3.5.

As far as the standard libraries are concerned, they are separated into General libraries (provide general classes) and Model libraries (specify models for the continuum mechanics). Both of them are placed in the $FOAM_LIB/$WM_OPTIONS directory. The list of the standard libraries can be found in OF User Guide Tables 3.7, 3.8, 3.9, 3.10.

3.5. Pre-processing

On this work two main Pre-processing tools will be used. On the one hand Gambit, Fluent’s Preprocessing tool, and on the other hand snappyMesh, a very powerful Pre-processing tool provided with the current version of OpenFoam (1.5). The later will be analyzed in order to determine if it is suitable for meshing complex terrain.
3. CFD: OpenFOAM

3.5.1. Gambit

After having created the mesh with Gambit and having exported it to “.msh” format, it will be placed on the main directory of the case and the following command will be typed:

`fluentMeshToFoam “mesh_name”.msh`

Before typing the command it is needed to create the controlDict dictionary in the system subdirectory, otherwise the command will not work.

After typing the command the mesh is created and stored in polymesh folder.

3.5.2. SnappyHexMesh

When a snappyHexMesh is wanted to be run the following directories must be created:

![Diagram of directories]

In the constant/polymesh folder the background must be placed. Such mesh must be comprised of hexahedral cells with an aspect ratio close to one.\(^8\)

The stl file is a surface file containing the data of the surface it is wanted to be meshed.

In the system folder, the SnappyHexMesh dictionary will be located, where all the variables relating to mesh characteristics and quality will be set.

The meshing process is comprised of three steps which will be stored in the main directory as 1, 2 and 3 files respectively. First the refinement is done according to refinement levels set in SnappyHexMesh dictionary. In the second step cells are snapped in order to get smooth surfaces and eventually in the third steps surfaces layers are created according to the values set in the snappyHexMesh dictionary by the user.

\(^8\) Aspect ratio is defined as the width divided by the height.
In order to run the application as the rest of applications in OpenFoam just its name must be typed (in this case snappyHexMesh) in the command line in the case main directory and the meshing will start.

An example of a SnappyHexMesh case can be found in the enclosed DVD in directory “Cases\SnappyHexMesh\Bump\Meshing”). Here all the files needed in order to run SnappyHexMesh can be found such as the stl file, background mesh and so on.

3.6. Post-processing with Fluent

As it was said in Method section (1.5) the visualization of the results will be done using Fluent as in Älvkarleby the cluster has been used in order to carry out the present work is not able to run Paraview (post-processing tool in OF).

Two converters are supplied for this purpose: foamMeshToFluent which converts the OpenFoam mesh into Fluent format (.msh); and foamDataToFluent which converts the OpenFoam results into Fluent results format (.dat). Such converted files will be placed in a so-called fluentInterface subdirectory of the case directory.

In order to convert OpenFoam results we need to include a file called “foamDataToFluentDict” into the “system” subdirectory of our case (see Figure 3.3). Such file can be copied from the following path:

SWM_PROJECT_DIR/applications/utilities/postProcessing/dataConversion/foamDataTo Fluent/foamDataToFluentDict

and pasted into our system subdirectory. In such file the relation between Fluent variables and OpenFoam variables is set. We can see the relation for the most used variables between both programs:

<table>
<thead>
<tr>
<th>Fluent name</th>
<th>Unit number</th>
<th>Common OpenFOAM name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRESSURE</td>
<td>1</td>
<td>p</td>
</tr>
<tr>
<td>MOMENTUM</td>
<td>2</td>
<td>U</td>
</tr>
<tr>
<td>TEMPERATURE</td>
<td>3</td>
<td>T</td>
</tr>
<tr>
<td>ENTHALFY</td>
<td>4</td>
<td>h</td>
</tr>
<tr>
<td>TKE</td>
<td>5</td>
<td>k</td>
</tr>
<tr>
<td>TED</td>
<td>6</td>
<td>epsilon</td>
</tr>
<tr>
<td>XF_RF_DATA_VOF</td>
<td>150</td>
<td>gamma</td>
</tr>
</tbody>
</table>

Figure 3.6 Unit numbers for the conversion. [Source: OF User guide]

The rest of the unit numbers can be found in the following path:

SWM_PROJECT_DIR/applications/utilities/postProcessing/dataConversion/foamDataTo Fluent/fluentUnitNumbers.txt

An example of the foamDataToFluentDict file can be found in the enclosed CD in directory “cases/Coriolis/system”).
4. PROCESS AND RESULTS

4.1. Comparison OpenFoam vs Fluent

Nowadays the most well known CFD program and the one that is supposed to work better is Fluent. In order to know whether OpenFoam results are reliable or not, they will be compared with results provided by Fluent. It is important to keep in mind that although Fluent’s results are usually rather close to the reality it is not perfect so this first section will be just a comparison between the two programs with the aim to see what the differences are. Later, a study should be made comparing both programs and experimental results to finish the validation part.

The following two cases will be considered:

- **1. Flat plane**: It is a simple 2 dimensional plane 500 meters long and 60 meters high. The third dimension (z) is not considered. The flat plane case can be seen in the following figure.

![Figure 4.1. Flat plane case.](image)

- **2. Bump**: The shape of the hill, shown in figure 4.2, is defined analytically as

\[
y/h = -\beta [J_0(\frac{\beta}{a})I_0(\frac{\beta}{a})-I_0(\frac{\beta}{a})J_0(\frac{\beta}{a})]
\]

where \( \beta = 1/6.048 \), \( \beta = 3.193 \) and \( a = 2h \) is the radius of the circular base of the hill, \( J_0 \) and \( I_0 \) are Bessel functions.

The bump height, \( H \), is 78 mm and is mounted on a 0.78 m wide, 0.25 m high and 1.56 m long flat plane (see Figure 4.3.).

![Figure 4.2. Cross section of the axisymmetric hill.](image)
In every simulation is going to be run on this project, unless otherwise indicated, the convergence will be set to 1e-5.

K-epsilon turbulence model will be used with the following standard model constant values:

<table>
<thead>
<tr>
<th>Cμ</th>
<th>0.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>1.44</td>
</tr>
<tr>
<td>C₂</td>
<td>1.92</td>
</tr>
<tr>
<td>TKE</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.1. K-ε standard turbulence model constants.

The solver is pressure based, using a linear discretization for the pressure and first and second upwind discretization orders (depending on the required accuracy) for the momentum and turbulent model equations. The relaxation factors will be set to 0.3 for pressure equation and 0.7 for momentum equation as well as for turbulent model equations.

Steady state model and incompressible flow is considered with a constant density set to 1.225 kg/m³ (air) and a kinematic viscosity of 1.460735 m²/s (Fluent predefined value).

In OpenFoam the so-called SimpleFoam solver will be used, which is recommended for steady-state and turbulent flow simulations.
4.1.1. Flat plane

The computational domain consists of 140,000 quadrilateral cells. The area is meshed trying to keep a good quality of the cells (90% of the cells’ aspect ratio is under 20) and the y’ value on the walls for the used inlet velocity profile is between 40 and 75. The mesh can be seen in the following figure. In order to be sure about the reliability of the mesh the non-grid dependency was checked.

![Figure 4.4. Flat plane case mesh.](image)

The boundary conditions for the flat plane case are the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inlet</strong></td>
<td>Velocity inlet</td>
</tr>
<tr>
<td></td>
<td>Constant inlet velocity</td>
</tr>
<tr>
<td><strong>Outlet</strong></td>
<td>Pressure outlet</td>
</tr>
<tr>
<td></td>
<td>Gauge pressure = 101325 Pa</td>
</tr>
<tr>
<td><strong>Floor and ceiling</strong></td>
<td>Wall</td>
</tr>
<tr>
<td></td>
<td>Smooth surfaces</td>
</tr>
</tbody>
</table>

Table 4.2. Boundary conditions. Flat plane.

The inlet velocity profile is constant and equal to 10 m/s.

Two different discretization orders (1\textsuperscript{st} and 2\textsuperscript{nd} order) will be used for momentum, k and epsilon equations.
4.1.1.1. Velocity comparison OpenFoam vs Fluent 1\textsuperscript{st} order

The velocity profiles will be compared using the 1\textsuperscript{st} order and 2\textsuperscript{nd} order discretization for the momentum, K and epsilon equations in order to be sure of the reliability of the results.

The comparison will be done along 5 vertical lines situated to 5, 100, 200, 300 and 400 meters far from the inlet wall. We will just compare the velocity results for the first 5 meters height, as from this height up the velocity is almost constant and equal to 10 m/s. In the following figure the velocity profiles can be seen for the 5 vertical lines.

As it can be appreciated the results computed by both programs are very similar. However, slight differences can be noticed at 100 and 200 meters velocity profiles but before and after such distances the results are almost identical.

Although the accuracy seems to be good still a comparison with higher order discretization is required. Most of the times the results are not accurate enough using first order discretization so the order is switch to 2\textsuperscript{nd} order and the results comparison along the five vertical lines will be carried out.
4. PROCESS AND RESULTS

4.1.1.2. Velocity comparison OpenFoam vs Fluent using 2\textsuperscript{nd} order

The same figure as for the 1\textsuperscript{st} order is depicted comparing the velocity profiles at different distances from the inlet wall for both programs using the 2\textsuperscript{nd} order discretization order. The obtained results are the following:

As we could have predicted, the results resemble stronger now the 2\textsuperscript{nd} order of discretization is being used. In this case the curves are exactly the same for every distances, what means that including the higher discretization order it is managed to avoid differences that took place at 100 and 200 meters far from the inlet wall. In other words, the higher the discretization order the higher the accuracy of the results as well as the computational costs.

To end, let’s compare the results for 1\textsuperscript{st} and 2\textsuperscript{nd} order obtained by OF just to see the improvement in terms of accuracy when we change to higher discretization orders. In the following chart the velocity profiles for 1\textsuperscript{st} and 2\textsuperscript{nd} order are represented at the previous computed vertical lines.
4. PROCESS AND RESULTS

In the light of the above, it is recommended to use higher order discretizations than the first one as quite big differences are noticed in the velocity profiles. The biggest differences take place at 400 meters, when the wind flow over the smooth flat plane is already stabilized.

4.1.1.3. Velocity profiles

The velocity profile at 400 meters far from the inlet resulting from the OF simulation using 2\textsuperscript{nd} order discretization (same results for Fluent) is going to be compared to the power law expression and the formulation presented by Alexandrou (1996) which although have not been proven theoretically they do have been proven to match with experimental results. Both formulations were explained earlier in the theory in sections 2.1.7.2. and 2.1.7.3.

a) Power law expression:

The value \( m \) is calculated for a linear regression with the previously obtained velocity results by OF at the vertical line 400 meters far from the inlet from the ground up to 4 meters high. The values for the \( m \) and the coefficient of determination \( (R^2) \) calculated with Microsoft Excel are the following:

\[
\begin{align*}
    m &= 0.10255. \\
    R^2 &= 0.9986.
\end{align*}
\]

As it can be seen the coefficient determination is nearly 1, which means that the results are very close to follow a power law expression. In Figure 4.8. on the left, the velocity profile computed by OF and the power law expression for the previously obtained \( m = 0.10255 \) can be seen.
b) Alexandrou’s formulation:

The results obtained in order to fit the velocity profile at 400 meters far from the inlet, are sum up in table 4.3. First the wall shear stress was taken from OF results in order to calculate the $u^*$ and be able to make the formulation.

<table>
<thead>
<tr>
<th>c (varied during fit)</th>
<th>0.1851</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = 1/c$</td>
<td>5.4</td>
</tr>
<tr>
<td>$h$ (boundary layer thickness)</td>
<td>5.42</td>
</tr>
<tr>
<td>$B$ (by equ. 2.13.)</td>
<td>11.40</td>
</tr>
<tr>
<td>$u^*$ (from CFD)</td>
<td>0.287</td>
</tr>
</tbody>
</table>

Table 4.3. Velocity profile computed

The obtained c value strongly resembles to the value got by Montavon (1998), 0.1851 against 0.1835. (Table 2.1.)

In the following chart, the comparison between OF velocity profile and the power-law expression (on the left) and Alexandrou’s formulation (on the right) curves can be seen.

![Graph comparing OF and Alexandrou's formulation](image)

Figure 4.8. Velocity profile computed by OF vs Power-law expression (on the left) and Alexandrou’s formulation (on the right).

It can be appreciated that both formulations fit very well with the results obtained with OpenFoam. Thus, we consider the results got by OpenFoam for the flat plane are correct being proved the accuracy with theoretical formulation as well as comparing them to the results obtained by commercial CFD tools (Fluent).
4.1.2. Bump case

The computational domain consists of about 666,000 hexahedral cells. The volume is meshed trying to keep a good quality of the cells (85% of the cells' aspect ratio is under 20) and the $y^*$ value near the walls for the used inlet velocity profiles is controlled to be between 30 and 60 [26]. The mesh on the bump and its surroundings can be seen in the following figure.

![View of surface mesh and cross section mesh for the bump case.](image)

The boundary conditions for the bump case are sum up in the following table.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inlet</td>
<td>Velocity inlet</td>
</tr>
<tr>
<td></td>
<td>Logarithmic velocity profile with no turbulence at the inlet</td>
</tr>
<tr>
<td>Outlet</td>
<td>Pressure outlet</td>
</tr>
<tr>
<td></td>
<td>Gauge pressure = 101325 Pa</td>
</tr>
<tr>
<td>Floor and ceiling</td>
<td>Wall</td>
</tr>
<tr>
<td></td>
<td>Smooth surfaces</td>
</tr>
<tr>
<td>Lateral walls</td>
<td>Symmetryplane</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.4. Boundary conditions bump case.

The inlet velocity profile is defined as (in m/s):

\[
U = 27.5 \left( \frac{y}{0.039} \right)^{\frac{1}{7}} \quad \text{if } y < 0.039
\]

\[
U = 27.5 \quad \text{if } y > 0.039
\]
The results obtained with OpenFoam and Fluent will be compared close to the bump, analyzing the separation along the centerline with 1\textsuperscript{st} and 2\textsuperscript{nd} order discretization orders.

Detailed three-dimesional laser-Doppler velocimeter measurements are available for the validation of the CFD models (Simpson 2002; Byun and Simpson 2005) for this case. The measurements were conducted in a Low Speed boundary Layer Wind Tunnel, which has been used in many previous studies and is described by Devenport and Simpson (1990).

In the following picture the experimental results obtained by Dadvison (2006) are presented. The red line represents the line where the velocity magnitude is equal to zero, above it the velocity will be counter the main flow, that is to say, recirculation will take place.

The recirculation bubble along the centerline will be compared qualitatively first with 1\textsuperscript{st} discretization order and thereafter with 2\textsuperscript{nd} discretization order using OpenFoam and Fluent.
4.1.2.1. Comparison Experimental vs OpenFoam and Fluent

In the following pictures the separation along the centerline using 1\textsuperscript{st} discretization order can be seen.

![Figure 4.12. Separation along the centerline computed by OF 1\textsuperscript{st} order (x–y plane).](image1)

![Figure 4.13. Separation along the centerline computed by Fluent 1\textsuperscript{st} order (x–y plane).](image2)

It can be noticed that the results obtained by both programs are very close to results obtained by Davidson (2006) (presented in Figure 4.11.).

However, as it was concluded in the Flat plane case it is recommended to use higher 2\textsuperscript{nd} order discretization as many times 1\textsuperscript{st} order’s predictions are not accurate enough.

So another two simulations are done using this time the 2\textsuperscript{nd} discretization order instead of the 1\textsuperscript{st} one. The results are presented in the following pictures.
Surprisingly OpenFoam’s results are now much further from the experimental results than they were when the 1st order was used. The separation is widely over-predicted and the solver is not able to provide accurate results in the wake region. The results obtained with Fluent, though, are quite similar to those obtained using the 1st order and the accuracy remains being rather good. Many previous research works have showed that turbulence models such as k-epsilon are likely to over-predict the separation and consequently they do not work properly in separation cases such as bump case. Fluent, though, is able to predict really accurate results even using k-epsilon model.

A final simulation is done in order to try to improve the results obtained with OpenFoam in the recirculation region. The turbulence model is changed and instead of using the standard k-ε, the k-ε RNG model will be employed, which adds an extra term in the dissipation transport equation in order to improve the performance in separation flow cases. The results using the new turbulence model computed by OF are presented in the following picture.

Figure 4.14 Separation along the centerline computed by OF 2nd order (x–y plane).

Figure 4.15. Separation along the centerline computed by Fluent 2nd order (x–y plane).
The results are quite close to those obtained with the standard model (Figure 4.14.) and too far from the experimental results (Figure 4.11). In this case, it is not worth using this more complex model since no increase in accuracy is got and computations will consume more time.

Anyway, the predictions in the wake are not currently so useful and interesting as the wind resources are minimum in this regions. The highest velocities will take place at the top of the bump and consequently it will be there where most of the aero generators will be placed.

Five vertical lines are drawn along the bump (plane y-z) 0.05 meter far one from each other (see Figure 4.17.) and the velocity profiles are computed in such lines by OpenFoam and Fluent using the 2\textsuperscript{nd} order discretization order (see Figure 4.18.)
4. PROCESS AND RESULTS

Figure 4.18. Velocity profiles along five vertical lines.

As it can be seen the velocity profiles calculated by both programs along the five lines are exactly the same and in case a prediction of the power production should be performed for such sites the calculations will be rather accurate according to the results obtained.

4.1.3. Conclusion

In this section two cases were used in order to try to prove the reliability of the open source CFD OpenFoam tool.

In the first case, the flat plane, no differences were appreciated between the velocity profiles calculated by both programs.

In the second case (the bump), though, big differences took place mostly after the bump in the wake region. While Fluent was able to predict the difficult phenomena of deattachment, OpenFoam over-predicted the separation and not reliable results were got in this zone. It is worth highlighting that much previous work such as Simpson (2002) or Davidson (2006) concluded that CFD RANs models were not able to predict...
correctly the complex phenomena of separation and reattachment. Thus, although Fluent is able to do it, OpenFoam can be considered to be working well as its results are in keeping with previous research work.

It is important to highlight as well, that OpenFoam is quite sensible to discretization order and it is strongly recommended to use the 2nd upwind discretization order if accurate results are required. The first order discretization can give us an idea about the wind flow behaviour but it cannot be considered useful when looking for accuracy.

In the future, when computational available resources are enough in order to use DES even LES models the predictions in wake regions will be much better and reliable results will be got even in such complicate areas. Up until then, RANS models are to be used and OpenFoam as it was shown in this section is a reliable tool which can be useful in wind farm siting as long as the user bears in mind the limitations in separation cases and using it together with onsite measurement in order to provide a good assessment of wind conditions.
4.2. Roughness implementation

The current OpenFOAM version (OpenFOAM 1.5.) is not provided with an option to include the roughness as a calculation variable so the purpose of this section is to implement in OpenFoam a wall function which will take into account the roughness. In addition to that, a pre-processing application must be designed which will permit the user to convert automatically the provided data (in map format) to OpenFOAM files. On top of that, for this project it is required to be able to set variable roughness over our terrain, thus, whatever approach is taken, it must permit the user to set non-constant roughness.

4.2.1. Adding roughness as a variable in OF

For starters, in order not to modify the OF source code, some new files are created; where the necessary code in order to implement the roughness will be added. With this view, the following new files are created which will be placed in a new folder called “src” in the user directory $WM_PROJECT_USER_DIR. The new files which are needed to define the roughness and not to change the source code are the following (they are divided in three folders):

<table>
<thead>
<tr>
<th>Folder</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kepsilonvfall</td>
<td>kEpsilonvfall.C, kEpsilonvfall.H</td>
</tr>
</tbody>
</table>

Table 4.5. New files in order to implement roughness in OF.

Small changes are made in Rasmodelvfall and KEpsilonvfall folders in order to work together. However, bigger modifications will be done in the wall function folder where roughness is added as a variable in order to be used when solving the Navier-Stokes equations near the wall. In Rasmodelvfall.H and Rasmodelvfall.C also the roughness variables’ (Cs, Ks) definition takes place.

In addition to that a new solver “windFoam” will be created, which will call the new Rasmodel, Kepsilon, wall function and so forth and so on. This new solver will be placed also in the user directory in the folder “solvers”.

All the files were mentioned before can be seen in the enclosed DVD path “/code”.

As a first approach the roughness is implemented as a constant variable. It is added as a wall function coefficient in the RASProperties file (see Figure 3.3), just below the
values for the von Karman constant and the empirical constant E. This way, we are not able to set variable roughness yet, even we are not able to set a different roughness value for each different wall but it is a first step forward the target. In order to try it out and see that it is working well the bump case is run with the new solver (windFoam) and the rest of modifications, including the roughness set to zero. Indeed the results are the same as in previous simulations.

After thinking about some possible solutions, roughness is eventually decided to add it as a boundary condition in order to be able to set variable roughness along the terrain of interest. As the rest of the boundary conditions, it will be added in a file which will be located in the '0' folder inside the case we are running (see Figure 3.3), but in this case since two variables are needed to define the roughness (Cs and Ks), two new files will have to be added instead of one. Unlike the other boundary conditions roughness will be just considered close to the walls defined as ‘wall’ and not in the whole domain. Moreover it will not be modified throughout the simulation, the values that are set by the user at the beginning of the simulation will remain constant (it does not happen the same with other boundary conditions such as pressure or velocity that change their value throughout the simulation).

In order to test it the bump case is run again with the new solver (windFoam) and its new wall functions, setting the values for roughness constant (Cs) and roughness height (Ks) to zero in previously created Cs, Ks boundary condition files in ‘0’ folder. Indeed, the results are once again the same as in previous simulations so the research continues.

**4.2.1.1. Wall function validation.**

In order to definitely test the new wall functions (including the roughness), a new simulation is performed using the Flat plane case, but this time some roughness will be included and the results will be compared to the commercial CFD tool Fluent.

According to Fluent, in order to get accurate results the roughness can be never such that the wall-adjacent cell is smaller than it. In the present case the wall-adjacent cell is situated 5.12 millimetres above the ground so such value is the cap for the roughness height if reliable results are required. Two roughness values will be tested, 0.0024 m and 0.005 m and the velocity profiles will be compared at 400 meters far from the inlet using OpenFoam as well as Fluent.

The value for Cs (roughness constant) will be set to 0.5 (predefined value for Fluent) as there is no clear guideline for choosing it.

**a) Roughness 0.0024 m.**

In the following chart the velocity profile at 400 meters is presented for Fluent and OF using 1st and 2nd order discretization as well as the velocity profile got in section 4.1.1. for smooth surface, the later just to have an idea how the roughness affects to the velocity profile. The mentioned data is represented in the following chart.
Several things can be noticed from the above figure. First of all, big differences take place when switching from 1\textsuperscript{st} to 2\textsuperscript{nd} order discretization for OpenFoam, however, it is not the same for Fluent as with 1\textsuperscript{st} order discretization order is able to predict correctly the velocity profile. Thus, it is confirmed that when working with OpenFoam it is advisable (even mandatory) to use at least the 2\textsuperscript{nd} order discretization, otherwise it is likely not to get accurate results.

Secondly, the results for both programs, OpenFoam and Fluent, using the 2\textsuperscript{nd} discretization order are exactly the same. Some differences are noticed when the stabilization is being reached and the boundary layer thickness is slightly different, about 6.15 meters for Fluent against 6.3 meters for OpenFoam (=2.4\% difference).

**b) Roughness 0.005 m**

For this roughness it must be kept in mind that maybe not good accuracy is got, since as it was said before Fluent recommends not to have roughness heights bigger than the height of the first node and although in our case we fulfill the requirement (the first node is placed at 0.0051 meters height) we are not too far not to do it.
In this case the 2nd order discretization will be used as in previous cases it was noticed that the 1st discretization order (at least for OpenFoam) was not accurate enough. In the following figure the velocity profile computed by OpenFoam and Fluent applying 0.005 meters roughness are represented. Moreover, the velocity profiles for the smooth case and roughness equal to 0.0024 meters are drawn too in order to see how the roughness affects the velocity profile.

![Velocity distribution computed by OF in m/s.](image)

We can see that the results are not the same for both programs. While Fluent predicted a velocity profile quite similar to the previous roughness case (0.0024 m), OpenFoam computed a velocity profile where the roughness would affect much more, being the velocity considerably lower than Fluent’s.

The differences obtained between both programs might be due to the lack of accuracy caused by the height of the roughness (similar to the first node height). In order to know which velocity profile is correct, it is decided to change the mesh, make it coarser so that the first node is considerably higher than the roughness height value which is being working with (0.005 m).

The new mesh instead of 200 nodes in vertical edges will have 100 nodes and the growth ratio is kept at 6e-5, so the first node will be situated 0.0102 meters above the ground. The y+ in previous simulations at 400 meters far from the inlet was about 70 so
with the new mesh the value will be doubled becoming about 140. Although it is usually recommended to be around 60 the results will be still very accurate.

In the following figure the velocity profiles for the new mesh can be seen, using OpenFoam and Fluent. As in Figure 4.18, the velocity profile for smooth case and roughness equal to 0.0024 is represented.

![Figure 4.21. Velocity distribution computed by OF in m/s.](image)

As we can see in the above graph the velocity profiles computed by OpenFoam and Fluent using the new mesh match perfectly. According to the simulations made using above mentioned two different meshes, it can be said that Fluent is strongly dependent on the roughness height in order to get good accuracy and if the roughness height is close to the height of the first cell, we will not get good results. OpenFoam, though, works much better in such situations and accuracy is not lost. However, it is not recommended to work with cases where the roughness height is much bigger than the height of the first cell.

To end, we can see that the implementation of the roughness has two main consequences: first the velocities decrease while the roughness is increased (something predictable) and secondly that the boundary layer thickness also increases as the roughness is increased. In the figure above it can be seen how for the smooth case the boundary layer thickness is about 4.5 meters, for the roughness equal to 0.0024, 6.2 meters and for the last roughness 0.005, 6.5 meters.
4.2.2. Pre-processing roughness data application

That is well and good but in real cases it will not be so easy to add the roughness values into the boundary conditions files. Let’s say we have a case with 300 hundred thousand cells where for instance 3000 cells are owned by a wall defined as “wall” and the roughness is not constant. In such case the user should write 3000 values for Cs and Ks by hand, something that it is not feasible. Thus, for big cases it would be convenient to make easier to write the boundary files, that is to say, try to automate as much as possible the pre-processing of the roughness data.

4.2.3.1. Roughness map files

The roughness data is provided in “.map” extension files and an example can be seen in the Appendix A.4. The format of such files is quite simple:

The first five lines of the files are defining the name of the file we are reading, the units for the values of the points as well as for the roughness and so on. We will skip them and this data will be asked when the user is running the application. Thereafter, the roughness is set for certain areas of the terrain. Each area contains a contour line (basically a polygon) of constant roughness. The area has a simple header that contains three values: the roughness value on either side of contour line (Cs and Ks values) and the number of contour line points whose 2D coordinates are written just below, one after another ordered clock wisely. In the following figure we can see the graphical representation of the example map file presented in Appendix A.4.

![Figure 4.22. Roughness map-file example.](http://waspengineering.dk/WebHelp/FileFormatofMAP.htm)

9 For more information follow the link: [http://waspengineering.dk/WebHelp/FileFormatofMAP.htm](http://waspengineering.dk/WebHelp/FileFormatofMAP.htm)

10 Point A (1490002 ; 7048033) and grid size 2000x2000 m
If we look at the map file (Appendix A.4) it can be appreciated that the roughness values for the first area are 0.8 and 0.0005 respectively; for the second area 0.7 and 0.0002 and so on and so forth.

4.2.3.2. Roughness application

Once it is known how the roughness data is delivered, we start thinking about how to transform the data from the map file we are provided with, to OpenFoam roughness boundary condition files.

Each boundary wall is divided in small faces (cells) which are characterized by its central point. After the data transformation the values for Cs and Ks will have to be written in the OF boundary condition files (in ‘0’ folder, look at Figure 3.3) for the central point of each small face.

The procedure is represented in the following flow chart:

![Flow chart of the procedure.](image)

The central points of the small faces are checked whether they are or not inside a polygon. If they are inside any polygon, the Cs and Ks values of such polygon will be copied. When the last polygon and the last central point are reached the Cs and Ks OF boundary condition files are rewritten with the new values of the roughness.

In order to do the process described above some pieces of C++ code must be written. The following files are created:
- roughnessToFoam: Main application. It will call different files in order to carry out the process.

- CreateCsKs: Cs, Ks variables are created and read from the ‘0’ folder. Such step is necessary in order to make OpenFoam know that the user is working and changing these two boundary files.

- Check_inside: It will check if a point is inside a polygon and if it is the Cs/Ks values will be copied.

- Read_mapfile: Reads the map file.

The code of the above described files can be seen in Appendix A.3.

In order to know whether a central point is inside a polygon or not, the following two approaches were considered:

a. The angle between the lines that join each couple of two consecutive vertexes and the point is computed. If the sum of all computed angles is 360°, then the point is inside the polygon, else the point is outside.

The angle is calculated using the cross product: 
\[
\cos \varphi = \frac{(v_i \times v_{i+1})}{||v_i||v_{i+1}||}
\]

![Figure 4.24. Angle calculation method a).](image)

If \( \sum \varphi_i < 360 \) the point is outside.

If \( \sum \varphi_i \Rightarrow 360 \) the point is inside.

b. The central point coordinates and the coordinates of each couple of consecutive vertexes of the polygon are compared. If the central point fulfills either the conditions 1 or conditions 2 for all the couples of consecutive vertexes, it will be inside, else it will be outside:

Conditions:

\[
\begin{align*}
1 \quad & \begin{cases}
    \text{a. } \text{point}_x < \text{vertex1}_x \text{ & point}_x > \text{vertex2}_x \\
    \text{b. } \text{point}_y < \text{vertex1}_y \text{ & point}_y > \text{vertex2}_y
    \end{cases}
\end{align*}
\]
a. \( \text{point}_x > \text{vertex1}_x \ \& \ \text{point}_x < \text{vertex2}_x \)

b. \( \text{point}_y > \text{vertex1}_y \ \& \ \text{point}_y < \text{vertex2}_y \)

where \( \text{point}_x \) refers to the coordinate \( x \) of the point and \( \text{point}_y \) refers to the coordinate \( y \). For the vertexes the same nomenclature was used.

The conditions above presented, check if the point is inside the area \( A \). If the point meets the requirements for all the couple of consecutive vertexes, it means that the point is inside the polygon, hence, the roughness values of the polygon will be set for such point.

The above explained two methods are analyzed in order to determine which one is the most suitable for the case is being studied. The main advantage of the first method is that it can be used in 3 dimensional systems. However, the roughness data is going to be delivered in 2D (plane \( x-y \)) so this advantage is useless. The main drawback of this method is that it is not possible to use it with concave polygons. In the following picture can be seen that the method will not work with points located in concave zones since the sum of the angles will be higher than 360 degrees although the point is outside.

\[ \sum \phi_i > 360 \] and it is seen that the point is outside.

In Figure 4.22, it can be appreciated that most of the polygons have concave zones. Thus, it is decided to rule out using the first method as it is not useful for our case.

The second method will be used in order to know whether the point is inside or not. The only limitation of this method is that it can only be used in 2D, but since the
roughness data is going to be delivered in 2D as well, the method is fully suitable for our case. The code for this method can be seen in the function Check_inside() in the file roughnessToFoam. H from line 48 on (Look at Appendix A.3).

The last thing must be done is related to the coordinate systems. The coordinate systems used for the mesh and in the map file can be different, hence, after reading the map file a coordinate transformation is to be done in order to work with the same coordinate system.

For that purpose the following data will be asked to the user while the application is being run:

- Offset: distance from the mesh coordinate system origin to the map file coordinate system origin.
- Coordinates referred to the mesh coordinate system of the point (1,0) in map file.
- Coordinates referred to the mesh coordinate system of the point (0,1) in map file.

Thereafter, the coordinates of the map file points are recalculated.

**RoughnessToFoam validation**

In order to try it out, a new case is run it will be checked if the roughness data is correctly transferred to OpenFoam’s boundary condition files.

The roughness data presented in Figure 4.22. is going to be transferred. Such data can be seen in Appendix A.4. roughnesstoFoam application is run and the following is got for the values of the Ks displayed by ParaView (OF’s post-processing tool).

![Figure 4.27. roughnessToFoam created roughness map](image)
As it can be seen the Figure 4.27 is the same as it was the figure 4.22. Thus, we conclude that the application works properly and is able to convert the data coming from the map file to OpenFoam’s boundary conditions. In addition, the working time is really small, of course dependent on the computer we are working with but it is estimated that a big case (6 millions cells) run in a current computer should not take more than one minute which is considered a reasonable working time for the application.

A manual for the application was written in order to make easier to use it. It can be seen at Appendix B.

Thus, now the user is able to include easily non-uniform roughness in the simulation cases. First, the pre-processing roughnessToFoam application will be run which will transform and write the roughness data from map file to OpenFoam boundary condition files (Cs and Ks) and thereafter the user will run the simulations using the new solver “windFoam” which includes the roughness as a computational variable.
4.3. Coriolis force

Most of the cases, coriolis force is not considered since the influence over the wind flow is usually neglectable. However, if the case we are considering is big enough the influence can be noticed. In our case the domain might be quite big (can span up to 20x20 km), so coriolis force will be included in the momentum equation.

The acceleration term due to coriolis forces is the following:

\[ 2*(\Omega \times U) \]

where \( \Omega \) is the rotational velocity of the earth equal to 7.29e-05 rad/s and \( U \) is the wind flow velocity.

The coriolis component will be dependent on the latitude. The more at the north, the more strong will be its influence, thus, in places where the latitude is quite high is particularly advisable to add such term.

The vertical component is usually neglected as is much lower than the gravitational acceleration.

Some code is added in “windFoam.H” file, where the latitude (in radians) will be asked to the user, and in “eqUn.H”, where the coriolis term is included into the momentum equation, e.i., \( S_M \) (source term) in Eq.2.3. of theory section will be equal to \( 2*(\Omega \times U) \).

Coriolis force is directional, that is why when such force is included the axis must be set as follows:

- X axis pointing towards the east.
- Y axis pointing towards the north.
- Z axis must be vertical.

If the user does not take into account the orientation of the axis when working with the mesh, as well as when simulating, the results will not be correct since coriolis force will be applied in a direction that is not correct.

4.3.1. Validation of the implemented coriolis forces.

A new 2d case is created to try out the new implemented term in the momentum equation. The considered case is a flat plane 20 km long and 2 km wide which is located in Älvkarleby (where Vattenfall’s Research & Development Center is located, Sweden) with a latitude of 1.057013 radians (60.56º). The rotational earth velocity is set to 7.2931e-5 rad/s. In the following figure the case is seen.
As it was said before care must be taken when the mesh is being created and the y axis must be pointing towards the north as well as the x axis must point towards the east. Such restrictions were taken into consideration in the above case so the inlet is in the west, the outlet in the east, the upper wall is in the north while the bottom is in the south.

The boundary conditions are the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inlet</td>
<td>Velocity inlet $U = 10 \text{ m/s}$ no turbulence at the inlet.</td>
</tr>
<tr>
<td>Outlet</td>
<td>Pressure outlet $\text{Gauge pressure } = 101325 \text{ Pa}$</td>
</tr>
<tr>
<td>Upper and bottom walls</td>
<td>Pressure outlet $\text{Gauge pressure } = 101325 \text{ Pa}$</td>
</tr>
</tbody>
</table>

The simulation will be run with laminar flow model and the roughness is set to zero.

In the following figure the wind trajectory can be seen, the particles entering through the inlet wall and going out from the bottom wall.

Due to coriolis force, it can be seen that the wind flow is not going out through the outlet but through the bottom wall. If the case instead of being situated at the north hemisphere, would be at the south hemisphere coriolis force would affect it in the opposite direction and the wind flow would tend to go out through the top wall instead.
The results seem promising but in order to be sure that the obtained results are correct it is decided to calculate the velocities using Excel. We will compare the velocity in x and y direction of the particle drawn in red in Figure 4.27, the one that goes into the domain at the top of the inlet wall. Also the distance that the particle covers (the red one in figure 4.27) before it leaves is calculated. The forces due to pressure divergence are neglected since in x direction pressure is almost constant and in y direction the gradients are very small. Hence, considering that the coriolis force is the only force applied on the wind flow the velocity for the particulate is computed.

It will be considered that within time intervals of 10 seconds (period of time short enough comparing to the time needed in order to occur variations in acceleration and velocity) the velocity and acceleration of the wind flow is constant, thus, in such intervals uniform accelerated motion formulas can be used. Those formulas are the following:

\[ a_y = a_{\text{coriolis, y}} = -2V_y \Omega \sin \phi \]
\[ v_y = v_{y-1} + a_y \Delta t \]
\[ a_x = a_{\text{coriolis, x}} = 2V_x \Omega \sin \phi \]
\[ v_x = v_{x-1} + a_x \Delta t. \]

where \( \phi \) is the latitude.

\( V_x \) is the velocity in x direction at this moment.
\( V_{x-1} \) is the velocity in x direction \( \Delta t \) seconds before the current iteration.
\( V_y \) is the velocity in y direction at this moment.
\( V_{y-1} \) is the velocity in y direction \( \Delta t \) seconds before the current iteration.

The values for the velocity at various distances far from the inlet computed with theoretical formulas as well as the velocity results obtained with OpenFoam are presented in the following table:

<table>
<thead>
<tr>
<th>Distance</th>
<th>( V_x ) - OF</th>
<th>( V_x ) - formulas</th>
<th>( V_y ) - OF</th>
<th>( V_y ) - formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 km</td>
<td>9.979</td>
<td>9.981</td>
<td>-0.629</td>
<td>-0.635</td>
</tr>
<tr>
<td>10 km</td>
<td>9.9197</td>
<td>9.921</td>
<td>-1.259</td>
<td>-1.27</td>
</tr>
<tr>
<td>15 km</td>
<td>9.8182</td>
<td>9.8205</td>
<td>-1.895</td>
<td>-1.9052</td>
</tr>
<tr>
<td>Time the particulate leaves the domain</td>
<td>9.748</td>
<td>9.75</td>
<td>-2.232</td>
<td>-2.241</td>
</tr>
</tbody>
</table>

Table 4.7. Velocity results calculated by OF and theoretically in m/s.

\( ^{11} \) Such acceleration term comes from Equation 2.4 (section 2.1.4.1.)
And the trajectory of the particulate along the domain according to the theoretical formulae and by OpenFoam can be seen in the following figure.

![Figure 4.30. Trajectory of the wind flow affected by coriolis force.](image)

It can be noticed that the velocities as well as the trajectory of the particle are exactly the same so it is considered that coriolis force formulation is correct and the solver is working correctly. The particle would leave the domain after covering 17800 meters in about 30 minutes.
4.4. Gravity force

Gravity force is not usually important if buoyancy effects due to density variations in the atmosphere are not considered. Although on this work density is considered constant, it is decided to include gravity force since in the future buoyancy effects will be added and then it will be necessary to consider the effect of the gravitational force.

Gravity force is included in the Momentum equation just adding an acceleration term equal to:

\[ \mathbf{a}_g = (0, 0, -9.81) \]

Without considering buoyancy effects it will be quite hard to appreciate the influence of the gravity on the velocity profile. However, the pressure in a vertical line should be influenced by the gravity and fulfill the following expression:

\[ \Delta p = \Delta h \rho g \]  \hspace{1cm} \text{Equ. 4.1} \]

Where \( h \) is the height, \( \rho \) is the density (1.225 kg/m\(^3\) for the air) and \( g \) is the gravity term equal to 9.81.

In order to test it, a simple case is run (see figure 4.29). A vertical flow is studied which goes up 20 meters high. Pressure will be computed setting a zero static pressure at the outlet (20 meters high).

![Figure 4.31. Gravitational force validation case.](image)

In the next picture the results for the pressure computed by OpenFoam are presented.
As it could have been predicted pressure changes linearly respect to the vertical and reaches the highest value at the inlet. The pressure difference between the inlet and the outlet according to Equation 4.1. should be:

\[ \Delta p = \Delta h \rho g = 20 \cdot 1.225 \cdot 9.81 = 240.35 \text{ Pa} \]

As it can be noticed in the Figure 4.32. OpenFoam’s results are correct and it is considered that the implementation of the gravity has been correctly done.
4.5. **SnappyHexMesh**

Meshing process is usually quite manual and is not easy to automate. Most of current commercial meshing programs need a experienced and skilled person who is able to get the most out of it. However, the latest version of OpenFoam comes with a new tool called SnappyHexMesh which creates automatically quite good meshes over complex geometries. In 2008 in Milan, when it was presented by the company Icon several complex cases were shown that have been meshed with such tool and the results where surprisingly good, e.g. the tool was able to mesh the city of Chicago creating 12.6 million cells in 267 min using 8 processors and it was got a 71% layer coverage.

In the light of the above it is decided to test the SnappyHexMesh and see if it would be interesting for meshing complex terrain as it could save a lot of time and consequently money.

As it was mentioned in section 3.6 in order to use SnappyHexMesh is necessary to have the surface is wanted to mesh in stl format (stereolithography format). The terrain data is commonly provided in xyz format, consequently, a converter is needed in order to transform the data from xyz to stl. Such conversion is not very common and it was not easy to find a suitable software for this purpose but eventually a program was found so called “Global mapper” which is able to do it. Global mapper is a terrain data viewer capable of displaying and converting the most popular elevation and vector datasets.\(^\text{12}\)

In order to analyze the performance of SnappyHexMesh two cases will be analyzed. First the bump case will be meshed and thereafter the same simuation as in 4.1.2. will be run. If the results with the new generated mesh comprise of the same number of cells or less is the same the tool will be consider to be useful. Second the well-known Arkveskevin hill is going to be meshed but in this case just a qualitative analysis of the mesh will be done.

**4.5.1. Bump case with SnappyHexMesh**

The bump case, as it was presented in section 4.1.2., is an axisymetric hill 78 milimetres high mounted on a 0.78 m wide, 0.25 m high and 1.56 m long flat plane(see Figure 4.2.).

SnappyHexMesh tool was found to have problems in meshing next to the walls so it is decided, instead of simulate just a halve of the bump (being necessary to set a symmetry plane), simulate the whole case so that problems do not give rise next to the symmetry plane.

In order to use SnappyHexMesh we need to get the bump case in stl and it will have to be converted from the source data which is in Gambit (got from 4.1.2.).

---

\(^\text{12}\) For more information look at the following webpage: [http://www.globalmapper.com/](http://www.globalmapper.com/)
The data in Gambit will be exported to msh in order to thereafter open it with Fluent. Once in Fluent the surface of interest (bump and floor) is exported in xyz format which will be opened with Global mapper terrain data viewer. While working on this section a weakness of Global mapper was noticed. Global mapper is a software which is mainly for big surfaces and when working with small cases (bump case is small) problems might rise due to that Global mapper’s output data has only two decimals accuracy (at least stl format output data).

In order to get around such inconvenience the output data in stl format before being converted was scaled and made 1000 times bigger. Thereby, if before from millimeters on were lost now from micromillimetres on are lost, an acceptable accuracy in our case. Consequently, the background mesh will have to be 1000 times bigger as well. Just a simple box is created with Gambit slightly smaller in width and length than the bump case surface and a mesh is defined with 120 cells in length, 60 in width and 50 in height.

The bump case displayed with Global mapper and the background mesh outline look as follows:

![Figure 4.33. Bump case displayed with Global mapper and background mesh wireframe.](image)

It is important to highlight that the background mesh must be divided by the stl surface in two volumes (volume A and B in Figure 4.33.) above and below the surface. Thereafter, in the SnappyHexMeshDict the value of the variable “locationInMesh” will determine the meshing volume which will be the one that contains such point. If the user creates a background mesh which is not divided into two volumes by the stl surface the meshing tool will not work as desires and even the process might crash. In our case the locationInMesh point will be located in the upper volume, that is to say, volume A.
4. PROCESS AND RESULTS

After optimizing the values for SnappyHexMesh the application is run and it is worthy to comment some of the variables were set:

- Refinement level: It is not recommended to use a very high value since with values over 3-4 sometimes the application crashes. It is closely related to the background mesh next to the slt surface. The cells next to the surface will be meshed according to the refinement level and the will be divided by 2 over the selected refinement level.

- nCellsBetweenLevels: Number of buffer layers between different levels. It is set to 3 in order to make smooth the change from one level of refinement to the next.

- Layer controls: There are many settings to control the addition of the layers but we will focus on 4 of them: the number of added layers (nSurfaceLayers), the expansion ratio between one layer and the next one (expansionRatio), the final layer thickness refer to the background mesh (finalLayerRatio) and the minimum thickness of the layer so that a layer is included (minThickness). If the later is not fulfilled the layer will not be added (in our case it is set to a very low value in order to always be added). The mentioned variables are set to:
  
  - nSurfaceLayers = 7;
  - expansionRatio = 1.2;
  - finalLayerRatio = 0.9;
  - minThickness = 0.000001;

With the above parameters the height of the first cell can be calculated:

The background mesh cell height was 5 milimetres.

The height of the first layer will be \( h = \frac{5 \times 0.9}{(1.2)^7} = 1.26 \) milimetres.

Thereafter, the obtained mesh must be rescaled and make it 1000 times smaller typing the following command in the command line of the main case:

\texttt{transformPoints --scale \{1e-3 1e-3 1e-3\}\}}

In the following figures the obtained mesh is presented. In figure 4.34. the mesh at the bump and surroundings in 3D can be seen whereas in Figure 4.35 and Figure 4.36. the mesh close to the surface at the bump can be appreciated in 2 dimensions.

\textbf{13} In order to check the values of all the variables used for meshing look at the enclosed DVD path: Cases\SnappyHexMesh\Bump\Meshing\bump\system
4. PROCESS AND RESULTS

Figure 4.34. Bump meshed with SnappyHexmesh.

Figure 4.35. Bump mesh Plane (y-z)

Figure 4.36. Detailed picture surface layers.
About 1,267,000 cells were created and taking into account that this time the whole bump is being considered, the number of cells is nearly the same as in the mesh which was done by hand in 4.1.2. (about 660,000 cells).

As it can be appreciated in the figures above, the mesh looks very good and the seven surface layers were correctly created. Although in SnappyHexMesh was set that the refinement level should be range between 1 and 3, it seems like the first refinement level was enough since it is the highest refinement level was used by the application. The strange features that appear in the above figures are due to that the mesh was sliced (plane x-y) and some cells were cut giving rise to strange features.

In order to be sure that the generated mesh is good enough, two simulations are run, first with the former mesh and thereafter with the mesh generated by SnappyHexMesh. All the settings for the case (boundary conditions, convergence criteria, etc...) will be the same as 4.1.2. except from the inlet velocity that will be constant and set to 20 m/s just to make it more simple. The inlet velocity profiles along the vertical lines defined in Figure 4.1. are computed and the results for the two different meshes are presented in the following picture.

![Velocity profiles using Snappy and former mesh](image)

Figure 4.37. Velocity profiles using Snappy and former mesh.
As it can be appreciated the velocity profiles are almost identical. The slight differences that take place, can be due to that in the former mesh a symmetry plane exists whereas in the second one does not, thereby the boundary could be slightly affecting the velocity profile (mostly close to the wall).

Eventually and to conclude with this case it is important to compare the time it took to do the mesh with SnappyHexMesh tool and the time used when doing by hand using a more common tool such as gambit or icem.

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SnappyHexMesh</td>
<td></td>
</tr>
<tr>
<td>Background Mesh (15 min.)</td>
<td>21 min.</td>
</tr>
<tr>
<td>STL conversion (10 min.)</td>
<td></td>
</tr>
<tr>
<td>SnappyHexMesh process (311 sec.)</td>
<td></td>
</tr>
<tr>
<td>Mesh done with gambit</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>8 hours</td>
</tr>
</tbody>
</table>

Table 4.8. Time comparison using SnappyHexMesh vs Manual meshing.

The benefits of using SnappyHexMesh are clearly reflected on the time reduction got by using the tool. While by hand meshing the bump case can take about eight hours (by an expert), using SnappyHexMesh could be done just in twenty minutes getting a high quality mesh.

4.5.2. Askervein

Askervein hill is a 116 m high hill located on the west coast of South Uist, an island towards the North West coast of Scotland. Its elliptical shape can be appreciated in the following picture.

![Askervein hill](image)

Figure 4.38. Askervein hill.

---

14 The meshing time with SnappyHexMesh tool depends on the computer is being working with.
Many research works has been carried out on such hill due to the extent measures were taken in the 80s. Many of current commercial CFD codes have been validated using such case.

Askervein hill will be meshed using SnappyHexMesh tool and the resulting mesh will be qualitatively analyzed in order to see if the meshing is successful or not.

A quite coarse background mesh is created (about 800000 cells), fulfilling the requirement of being divided in two volumes by the stl surface as it was commented in the previous case. Such mesh is composed by 20 meters height, long and wide cells and its emplacement respect to the hill can be seen in the following floor plan picture.

![Figure 4.39. Askervein hill: Background mesh location.](image)

Two simulations will be run, one for setting the number of surface layers to three and other one for seven in SnappyHexMeshDict. The rest of the values will be very similar to the ones were set for the bump case but a main change is made: it is disabled the concave surface checking parameter since Askervein hill has some regions which are concave and if it was enabled, such areas would not be meshed.

The simulation took about 20 minutes for 3 added layers and 25 for the one with 7 layers and the final mesh has about 1.2 and 1.5 million cells respectively.
The mesh is analyzed in five planes (a, b, c, d and e) which are defined in the following picture.

![Diagram showing five planes: a, b, c, d, and e.]

The mesh looks to be correct for both simulations (three and seven layers) in all the regions except from the one between the Planes b and c where problems arise when it is tried to add 7 layers. However, the layers are added nicely in the case of 3 layers. Such region can be seen and compared in the following figures sliced by the Plane e for 3 and 7 added layers respectively.

![Sliced by plane e: Three layers case.]

Figure 4.40. Askervin: Slice plane definition.

Figure 4.41. Sliced by plane e: Three layers case.
It can be noticed that for the case of 7 added layers suddenly the layers disappear (red circle in Figure 4.42) and there is a large area without any surface layer. If just 3 layers are added, though, such 3 layers are correctly added although strange things take place in the same region as well (blue circle in Figure 4.41).

Thus, whereas in most of the regions the surface layer was correctly created in others it was not. It was tried to change the SnappyHexMeshDict variables in order to improve the resulting mesh for the case it was wanted to add 7 layers but no success was accomplished. Even it was tried to use a refinement box at the top of the hill with the same target but the results were not good either.

It is concluded that SnappyHexMesh can be a very useful tool in the future but the lack of information\textsuperscript{15} as well as its poor performance in some specific regions, mostly when a number of layers is wanted to be added, does not make it a feasible tool by now. However, the author encourages to keep on working on it and trying to increase the knowledge of the tool in order to get the most out of it.

\textsuperscript{15} In the OpenFoam User Guide it is barely explained how to use it and no more tutorials are available so far.
5. CONCLUSION

OpenFoam is an open source code and like most of them is slightly more complicate to use it than commercial codes. It requires some Linux and C++ knowledge which it makes a little harder the beginning for a new user. However, since the code is open the user is allowed to change it and adapt it to his/her own necessities which makes it a very interesting tool for the future.

In light of the present project’s results, OpenFoam open source CFD tool turn out to be a reliable tool in order to use it in wind farm sitting as a complementary tool to onsite measurements. With the current available computational resources it is not possible to solve the instantaneous equations that govern fluid flows (Navier-Stokes equations) which is got around using turbulence models. Consequently, the results in some complex cases such as when separation takes places are not so accurate as we wanted. Thus, when RANs are being used it is vital to keep in mind the limitations of the assumptions and no to take for granted the results obtained by the CFD tool.

OpenFoam was found to be quite sensible to the discretization scheme is used and it was noticed that even for not very complicate cases it is advisable to use 2nd discretization order if accurate results are required.

The roughness variable was successfully added and it was proved to work properly comparing the results with the commercial CFD code Fluent. The new implemented wall function even turn out to work better than Fluent when the first cell height is similar to the roughness height since Fluent is not able to provide accurate results in such case whereas OpenFoam with the new wall function predicted correct velocity profiles.

It was managed as well to create a new application in order to transfer the data from the map file to OpenFoam boundary conditions. It was proved to work well even with big cases and consuming little computational resources. A manual was written in order to make easier to use it, which can be seen in Appendix B.

The implementation of the coriolis and gravity forces was successfully performed too and the results were validated comparing them with theoretical formulation. Coriolis force was verified that can have considerable influence on the wind flow when the studied domain is big enough and the latitude of the site under study is high. As it was expected, the influence of the gravity force is not big if there are not buoyancy effects considered, but it is useful to have it already implemented into the model for the future.

Eventually, SnappyHexMesh meshing tool was analyzed and although it worked well for a quite complicate case such as the bump case, it was not possible to do it for a real case, Askervein hill. More research is needed in order to fully understand the influence of each variable in the tool in order to adjust them to each specific case. It is recommended to continue working on it as the results for the bump look to be so promising and maybe with a better understanding it could be possible to mesh real complex terrains as well, such as Askervein hill.
6. FUTURE WORK

The atmospheric boundary layer contains numerous characteristics which have not been considered in this study. Amongst them, the buoyancy effect due to different densities along the atmosphere is a major factor which can strongly affect the behaviour of the wind flow, so it is recommended to add such factor including the energy equation as well.

Secondly, it is suggested to implement forest canopy models in order to correctly predict the wind flow over areas with forest. Although they can be considered as a roughness height it is much more accurate to consider them as a porous media.

Thirdly, it is necessary to automate the simulation process in order to be able in the future to perform several simulations one after another one changing the inlet wind direction and its magnitude as it was explained in section 1.3. This requires a new application which will be able to do it.

Eventually, it is recommended to consider the transient model instead of steady-state. This change would erase the limitation of the dimension of the case up to 20 x 20 km as it was explained before.
7. REFERENCES

[12] Introduction to Physical Oceanography, John A. Knauss
[23] Vestas V82-1.65 MW technical brochure.


A1. Power of wind

The power of the wind at a specific height is usually calculated in the following way:

\[ P = \frac{1}{2} \rho A v^3 \]


where \( A \) is the area of the blades of a wind generator.

The energy contained in the flowing wind during a certain period could be calculated multiplying the above equation by the considered period of time. Such calculation is valid as long as the wind velocity is constant which rarely is since changes in speed as well as in direction are very common.

Thus, in order to calculate the energy content of the wind at a site is not sufficient to know the wind speed, but you need to know also the different wind speeds that occur and their duration, in other words the frequency distribution of the wind speeds. Such information can be provided by the so-called wind rose diagram, see figure A.1.b. The data can be rearranged without considering wind direction (nowadays most of the turbines have a yaw mechanism in order to keep the rotor facing the wind anytime) and the typical Weibull wind distribution \( f(v) \) is got (see figure A.1.a).

\[ \text{Figure A.1. (a) Weibull wind distribution } f(v) \text{ and (b) Wind rose.} \]

A wind turbine produces power at wind speeds between the cut-in wind speed (about 3 m/s) and the cut-out speed (about 25 m/s). An example power curve is presented in Figure A.2. which corresponds to Vestas V-82 1.65 MW generator.

---

16 The diagrams correspond to measures perfomed at Rio Grande du Sol, Brazil. [22]
And finally the estimated wind turbine production is calculated by integrating the product of the wind distribution \( f(v) \) and the wind turbine power curve \( P(v) \) over the working wind spectrum. In the equation several efficiencies must be included which reduce the obtained amount of energy.

\[
E = 8760 \cdot \eta_T \cdot \eta_p \cdot \eta_{avail} \cdot \eta_{wake} \int_{V_{cl}}^{V_{co}} P(v) \cdot f(v) \cdot dv \quad \text{Equ. A1.2.}
\]

Due to a number of reasons the estimated production is reduced:

- High turbulence levels which can be relevant in complex terrain. \( \eta_T \).
- Variation in expected air density. \( \eta_p = \rho/\rho_0 \)
- \textit{Turbine availability} \( \eta_{avail} \)
- \textit{Wake} \( \eta_{wake} \) interaction that depends on wind direction and the position of wind turbines.

It is important to bear in mind that the maximum energy converted by a wind turbine is limited to called Betz’s limit which is \( 16/27 = 0.59 \).
Y plus is a non-dimensional distance which is commonly used in boundary layer theory and when defining the law of the wall. It is defined as follows:

\[ y^+ = \frac{u*y}{v} \]  

Equ. A2.1.

Where \( y \) is the distance to the nearest wall, \( v \) is the kinematic viscosity and \( u^* \) is the friction velocity, which is defined as follows

\[ u_* = \frac{\tau_w}{\rho} \]  

Equ. A2.2.

And \( \tau_w \) is the wall shear stress defined as \( \tau_w = \mu \left( \frac{\partial u}{\partial y} \right)_{y=0} \)

As it was commented in the theory section from a \( y^+ \) value on \( (y_{lam}) \) there is a change in the flow behaviour from laminar flow (viscous sublayer) to fully turbulent (turbulent sublayer). The change is not abrupt, but there is an intermediate zone, so-called buffer layer where the turbulent and viscous effects are both important.

The laminar and turbulent layers are characterized by different velocity profile curves and the switch from one layer to the other one is computed from the intersection between the two curves. It can be seen in the following picture:

![Figure A.3. Velocity in the near wall region.](image)

Thus the \( y^+ \) for which the flow changes from laminar to turbulent will be computed solving the next equation:

\[ y^+ = \frac{1}{\kappa} \ln(Ey^+) \]  

Equ. A2.3.
As we can see the equation depends on two constants. $\kappa$ is von karman’s constant which most of the scientist agree to set to 0.4781 and $E$ is the log-law constant which value diverge from research to research and scientists do not come to a clear agreement. Depending on the source the $E$ value can range from 9 to 10.

The same happens with CFD tools, that each one uses a different value for $E$. On this project OpenFoam and Fluent are used and the predefined $E$ values are 9 and 9.793 respectively.

In the following table the $y^+$ values for which the turbulent layer starts (depending on the $E$ value) computed from the Equ. A2.3 are presented:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
E & 9 & 9.3 & 9.6 & 9.793 & 9.9 & 10 \\
\hline
\hline
\end{array}
\]

Figure A1.1. $y_{lam}$ value depending on $E$.

In red the value for OpenFoam; whereas in blue that for Fluent.

Depending on the $y^+$ the fully turbulent layer equation will be employed further or closer from the wall.
A3. Roughness application code

The Roughness writer application is made up of four files: roughnessToFoam.C (main application), Readmapfile.H, Check_inside.H and CreateCsKs.H.

After being typed roughnessToFoam in the command line, the main file (roughnessToFoam.C) will call the other files in order to read the mesh first, read the map file and eventually write roughness values according to the map file in boundary condition files.

The code for the four files is presented in this section.

A3.1: roughnessToFoam.C

```c++
// ------------------------------*\
//
// Roughness writer
//
// Roughness writer from .map file (*.map) to Cs and Ks boundary conditions
//
// ------------------------------------------^/
//
// include "wallPvPatch.H"
//
// include "Readmapfile.H"
//
// include "Check_inside.H"
//
// include "CreateCsKs.H"
//
// using namespace std;
//
// --- ------------------------------------------ */
//
bool Check_inside (vector , Listvector);

int main(int argc, char *argv[])
{
    //
    # include "GetCoefFace.H"
    # include "CreateMap.H"
    # include "CreateCsKs.H"
    # include "Readmapfile.H"

    vector fvPatchlist patches = mesh.boundary();
    scalar startFace=mesh.neighbor().size();

    # include "Readmapfile.H"

    InfoC " Writing Cs / Cs " << endl; //writing Cs and Ks boundary files.
    Rs.write();
    Cs.write();

    return (0);
}
```
bool Check_inside(vector point, list<vector> polygon)
{
    int i=0; polygon.size()>= 2;
    bool oddNodes=false;
    for (i=0;i<polygon.size(); i++)
    {
        if(polygon[i][1]-point[1] == polygon[(i+1)%polygon.size()][1]-point[1] && polygon[i][0]-point[0] == polygon[(i+1)%polygon.size()][0]-point[0])
            oddNodes=!oddNodes;
    }
    return oddNodes; //if the point inside the polygon returns true
}

//........................................................................................................

A3.II: CreateCsKs.H

.isHidden "{Declaration CsKs} " << endl;

volScalarField Kn
{
    IObject
    {
        "Ks",
        runTime.timeName(),
        mesh,
        1000object:MUST_READ,
        100object:AUTO_WRITE
    },
    mesh
};

volScalarField Cs
{
    IObject
    {
        "Cs",
        runTime.timeName(),
        mesh,
        100object:MUST_READ,
        100object:AUTO_WRITE
    },
    mesh
};
A3.III: Readmapfile.H

```c
//************ readmapfile.h ************

filebase tap;
tap->scp.path();
Info << "Type the name of the Map file..." << endl;
char file[100];
tap->scp = "/";
cin >> file, 100);
tap->scp = file;
char ns[300];
int p = 0;
for (int i=0;i<100;i++)
{
    if (file[i] == '.'){
        p = i;
    }
    i = 100;
}
for (int i=0;i<100;i++)
{
    ns[i]=tap[i];
}
    ns[i+1]=tap[i+1];
    ns[i+2]=tap[i+2];
    ns[i+3]=tap[i+3];
    i = i+4;
}

fstream infile;
infile.open(nam);
if (!infile){
    Info << "Unable to open file " << endl;
    return (0);
}
char c;
int a = 0;
char d[100];
takeinto camera;
while (infile){
    if (a<4){ //We skip the first 4 lines of the map file
        infile.getline(6, 59, '\n');
    }
    else {
        infile.get(c);
        cadena += c;
    }
    ++a;
}
Info << "File " << file << " has been read" << endl;
Info<<endl;
Info<<endl;
Info<<endl;
Info<<endl;
scalar offset_x = 0;
scalar offset_y = 0;
scalar a1 = 1;
scalar b1 = 0;
```
75.  scalar a2 = 0;
76.  scalar b2 = 1;
77.  
78.  include <<< "Introduce the offset between both coordinate system origins (x0,y0)
79.  mesh-effect=(x0,y0)\exp (the offset must be in mesh coordinates)" <<<endl;
80.  include <<< "x_value: "
81.  csv>offset.x;
82.  include <<< "y_value: "
83.  csv>offset.y;
84.  
85.  include <<< "You introduce the values (x,y) for the point [1,0]\exp in mesh
86.  coordinate" <<<endl;
87.  include <<< "x_value: "
88.  csv>1;
89.  include <<< "y_value: "
90.  csv>0;
91.  
92.  include <<< "To end introduce the values (x,y) for the point [0,1]\exp in mesh
93.  coordinates" <<<endl;
94.  include <<< "x_value: "
95.  csv>0;
96.  include <<< "y_value: "
97.  csv>1;
98.  
99.  include;
100. int ced_size=cadens.size();
101. int n_polygons = 0;
102. jmp = 0;
103. Vecotr scalar Cc[2];
104. 
105. while (jmp < 3){
106. //Getting Cc,ks and number of vertex of each polygon
107.  int ni = 0;
108.  char cs[15];
109.  while (' ' | cadens[i]) == ' ')
110.  c[i]=cadens[i];
111.  ni++;
112.  i++;
113.  
114.  scalar cs_s=stat(cs);
115.  i++;
116.  ni = 0;
117.  char ks[15];
118.  while (' ' | cadens[i]) == ' ')
119.  k[i]=cadens[i];
120.  ni++;
121.  i++;
122.  
123.  scalar ks_s=stat(ks);
124.  i++;
125.  ni = 0;
126.  char nvertex[15];
127.  while (' ' | cadens[i]) == 'n')
128.  v[i]=cadens[i];
129.  ni++;
130.  i++;
131.  
132.  scalar nvertex_s=stat(nvertex);
133.  list cevector p_polygons;
134.  p_polygons.setsize(nvertex_s);
135.  Cc[0]=cs_s;
136.  Cc[1]=ks_s;
137.  Cc[2]=nvertex_s;
138.  
139.  i++;
140. //We are in the position of the first
141. //char of the first vertex of the polygon and we get the
142. //polygon points data;
143. 
144. if (cadens[i] == ' ') idxed_size;
145. for (int j=0; j<cedned.size(); j++){
APPENDICES

A3.IV: Check_inside

1. //*************** Check_inside.M ***************
2. 3. scalar side-startFace;
4. 5. forAll(patches, patchi)
6. 7. {
7. 8. const fvPatch& curPatch = patches[patchi];
9. 10. if (isType<allPolyPatch>(curPatch))
11. 12. {
13. forAll(curPatch, facei)
14. 15. Vector<scalar> centre_points=mesh.faces||[face+1].centre.mesh.points||;
16. 17. //April-09. The C2Xs data is provided in 3D so we dont consider
18. //the third dimension (z)
19. 20. bool inside=Check_inside(centre_points, p_polygon);
if (inside)
{
    label inletPatchId1 = mesh.boundaryMesh().findPatchID(currPatch.name);
    Cs.boundaryField()[inletPatchId1][face1] = Cs0f[1];
}

patch=patch-patches[patch1].size||
}
A4. Roughness .map file example

+ROUGHNESSLINE Görvik_0.map, Rikets Net (SE)

0.0 1.0 0.0 1.0
1.0 0.0 1.0 0.0
1.0 0.0

0.8000 0.0005 36
1500898.73 7050833.55
1500266.99 7051055.52
1499703.84 7051413.76
1499157.71 7051737.91
1499072.61 7051908.38
1497502.09 7052625.18
1496802.00 7052847.20
1496084.67 7052984.03
1495674.36 7052847.99
1495400.93 7052814.10
1495059.03 7052712.10
1494358.46 7052678.51
1493829.35 7052968.57
1493932.41 7053258.19
1494428.08 7053360.09
1494718.31 7053240.60
1495059.76 7053104.03
1495315.92 7053035.69
1495435.51 7053035.61
1495180.19 7053547.00
1495471.47 7053989.85
1495933.01 7054108.81
1497198.45 7054704.34
1498002.14 7055061.63
1498924.75 7055060.98
1499419.94 7054907.27
1499470.52 7054549.38
1499332.72 7053953.06
1499007.78 7053782.88
1498529.29 7053732.10
1498289.52 7053425.54
1498869.59 7052982.07
1500234.64 7052026.85
1500814.29 7051361.86
1500984.41 7050969.81
1500898.73 7050833.55
0.7000 0.0002 10
1502299.34 7050354.95
1502520.95 7050355.28
1502197.00 7050713.36
1501873.24 7051173.68
1502113.13 7051548.40
1502557.35 7051548.09
1502966.44 7051036.59
1503204.91 7050644.49
1503221.58 7050422.95
1502999.34 7050354.95
0.8000 0.0004 17
1502362.11 7047645.96
1501320.67 7048055.66
1500808.49 7048260.50

Header

This data is skip

Cs / Ks values and the number of vertexes of the polygon 1

First polygon’s vertex coordinates

Cs / Ks values and the number of vertexes of the polygon 2
<table>
<thead>
<tr>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500331.50</td>
<td>7049010.61</td>
<td>0.8000</td>
<td>0.0001</td>
</tr>
<tr>
<td>1500265.20</td>
<td>7049025.91</td>
<td>1494978.79</td>
<td>7046356.02</td>
</tr>
<tr>
<td>1500964.71</td>
<td>7049572.51</td>
<td>1493727.92</td>
<td>7047202.23</td>
</tr>
<tr>
<td>1501954.93</td>
<td>7049179.88</td>
<td>1494978.79</td>
<td>7046356.02</td>
</tr>
<tr>
<td>1502826.00</td>
<td>7049025.91</td>
<td>1493727.92</td>
<td>7047202.23</td>
</tr>
<tr>
<td>1503424.08</td>
<td>7049076.62</td>
<td>1495766.25</td>
<td>7047559.27</td>
</tr>
<tr>
<td>1503491.62</td>
<td>7048650.56</td>
<td>1494497.62</td>
<td>7047559.27</td>
</tr>
<tr>
<td>1503432.89</td>
<td>7047048.16</td>
<td>1495766.25</td>
<td>7047559.27</td>
</tr>
<tr>
<td>1503335.15</td>
<td>7047099.71</td>
<td>1494497.62</td>
<td>7047559.27</td>
</tr>
<tr>
<td>1502362.11</td>
<td>7047645.96</td>
<td>1494497.62</td>
<td>7047559.27</td>
</tr>
</tbody>
</table>
APPENDIX B
April 2009

by

Xabier Pedruelo

TABLE OF CONTENTS

1. APPLICATION.
2. HOW IT WORKS.
3. LIMITATIONS.
Input Data Example.
1. **APPLICATION**

`roughnessToFoam` application is done with the view of making easier to add roughness as a boundary condition when we are working with OpenFOAM. The roughness data will be provided in a map file format (.map).

2. **HOW IT WORKS**

The application name (`roughnessToFoam`) will be typed in the command line into the main folder of the case that we want to add the roughness to. In this main folder we need to have:

- Map file (*.map). In this file, data for the roughness will be provided.

- Cs and Ks files inside ‘0’ subfolder. It is recommended to set every roughness values to zero, since the application will just write the values for the areas which are defined in the map file, so the rest of the cells will not be modified and if before running the application a different value from zero was set it will remain.

- Mesh files. We need to have the points, faces and so forth and so on in the `constant/polyMesh` subfolder.

- We need to have a ‘system’ subfolder with the following files: `controlDict`, `fvSchemes` and `fvSolution`. The values set in such files does not matter when we are running the application, except for the value “StartFrom” in `controlDict` that is recommended to set to `startTime`.

After typing the application name in the main folder the following input data will be asked:

- We will be asked for the name of the file containing the roughness data. We will have to type it, including the file type. (Example: “Roughnessdata.map”)

- Offset: distance (x,y) from the mesh coordinate’s origin to the map file coordinate’s origin. The x value will be asked first and thereafter the y value. (x₀ and y₀ at Figure B.1, respectively).

- Coordinates with respect to mesh coordinate system for the point (1,0) in the map file coordinate system (a₁ and b₁ at Figure B.1., respectively).

- Coordinates for the point (1,0) which belongs to map file in mesh coordinate system (a₂ and b₂ at the figure 1, respectively).
3. **LIMITATIONS**

- It is necessary to create Cs and Ks files by hand, setting some initial values for the walls defined as “wall” (zero value settings are recommended). These files must be placed in the ‘0’ subfolder of the case.

- It is necessary to have the following files in ‘system’ subfolder: controlDict, fvSchemes and fvSolution.

- The input roughness data must be in “.map” format.

- The path for the main folder must be smaller than 300 characters long. If it is needed this value can be modified changing the value “file” in the ReadmapFile.H line 6.

- The name of the map file must be smaller than 100 characters long. If it is needed this value can be modified changing the value “nam” in the ReadmapFile.H line 12.

- Mesh cells must be considerably smaller than the polygons defined in the map file. Otherwise accuracy lose can take place.

- Since Cs and Ks values are provided in 2D (x and y axis) the calculations in order to determine the Cs and Ks for each point at the mesh walls are computed in 2D. Such fact has two main consequences:
The z axis must have the same direction for both coordinate systems and must be vertical.

If two surfaces are defined as “wall” and are located one above the other (same x and y, different z) the roughness values will be set at both of them and we will not be able to avoid it, or even define different roughness values for each one.

**Input data Example**

For instance, if the roughness data for the position \((x',y')\) in map file is given in km and the mesh is defined in meters \((x,y)\), the input data for the application would be:

- Offset: \(x_0\) and \(y_0\) respect to mesh coordinates in meters.
- The point \((x'=1, y'=0)\) respect to mesh coordinates would be \((-x_0+1/1000,-y_0)\).
- The point \((x'=0, y'=1)\) respect to mesh coordinates would be \((-x_0,-y_0+1/1000)\).

Figure B.2. Input data example.
Enclosed DVD

With the project it is enclosed a DVD where every piece of implemented code in OpenFoam as well as the cases which were carried out in order to validate the code are included. In the following figure the included folders can be seen.