APPENDIX I

A.I.1 Power of wind.

First the power that a wind turbine produces from the kinetic energy of the flow will be define:

$$Kinetic\ energy\ (J) = \frac{1}{2} \cdot air\ mass(kg) \cdot \left(air\ velocity\ (m/s)\right)^2 \quad \textbf{Eqn A. I. 1}$$

$$E_O(J) = \frac{1}{2} \cdot m(kg) \cdot \left(v_o\ (m/s)\right)^2 \quad \textbf{Eqn A. I. 2}$$

$$E_{after}(J) = \frac{1}{2} \cdot m(kg) \cdot \left(v_{out}\ (m/s)\right)^2 \quad \textbf{Eqn A. I. 3}$$

$$Total\ energy = E_O - E_{after} \quad \textbf{Eqn A. I. 4}$$

To calculate the power that wind makes the mass flow is needed.

$$P(W) = \frac{1}{2} \cdot \dot{m}\left(\frac{kg}{s}\right) \cdot \left(v_o\ \left(\frac{m}{s}\right)\right)^2 \quad \textbf{Eqn A. I. 5}$$

$$\dot{m} = \rho \cdot A \cdot v \quad \textbf{Eqn A. I. 6}$$

Where A is the area which blades create and ρ is the air density.

$$\boldsymbol{P(W) = \frac{1}{2} \cdot \rho \cdot A \cdot v^3} \quad \textbf{Eqn A. I. 7}$$

Finally multiplying the latter equation (Eqn A.I.7) by the period of time along the wind turbine is running, the energy produced can be got **[17]**. That calculation can be used just when the flow velocity is constant, and that situation hardly happens because the wind changes not only

in speed but also in direction very often. For that reason there are a kind of diagrams, which are called the Wind Rose diagrams **[18]** that represent the wind speed and its direction (Fig A.1). However, as by now almost all of the wind turbines can face the wind at any time thanks to a yaw mechanism, the Weibull wind distribution (Fig A.2), which provides the probable wind speed distribution f(v), are more relevant.

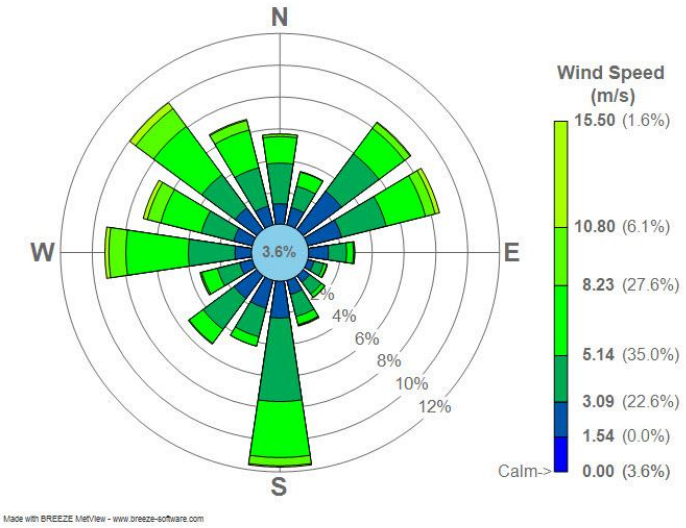


Fig A.I.1 Wind rose diagram [18].

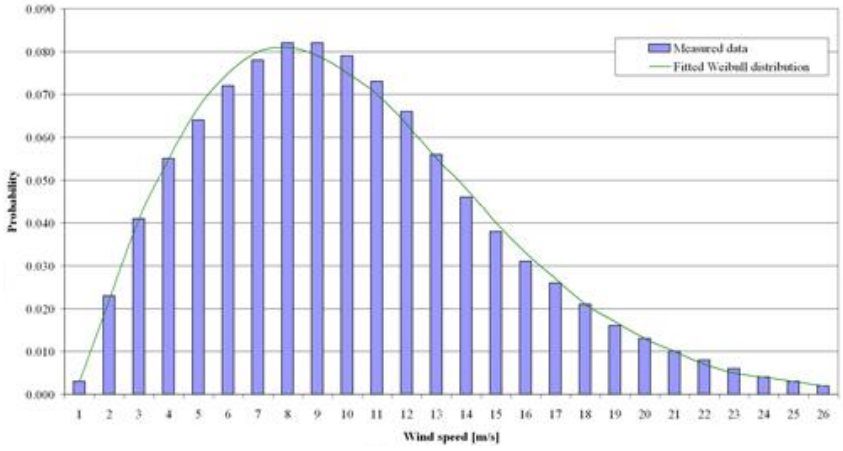


Fig A.I.2 Weibull diagram f(v) [21].

The wind turbines can work from a minimum wind speed which it is called cut-in wind speed (around 3 m/s) to a maximum wind speed called cut-out velocity (around 25 m/s). The power curve (Fig A.3) shows the dependence of the electric power generated by the turbine with the wind speed P(v).

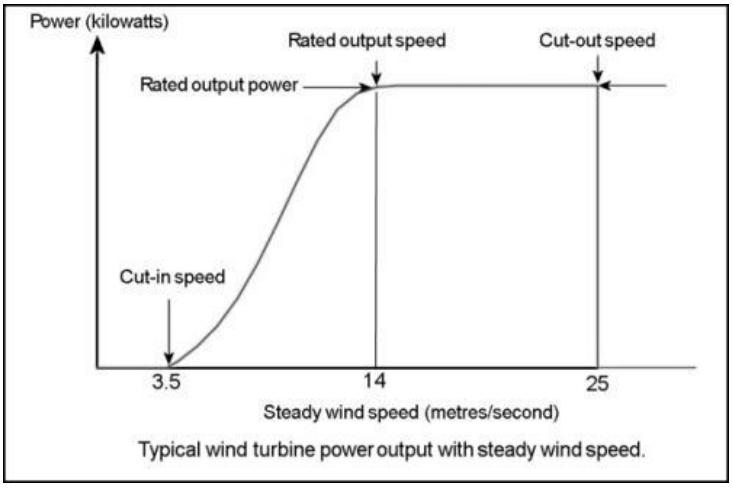


Fig A.I.3 Power curve P(v) [22].

As it was said before the wind speed is rarely constant, so the wind energy production is obtained by integrating the wind distribution times the power curve (f(v) x P(v)).

$$E = 8760 \cdot \eta_T \cdot \eta_\rho \cdot \eta_{avail} \cdot \eta_{wake} \int_{Vci}^{Vco} P(v) \cdot f(v) \cdot dv \qquad \textbf{Eqn A.I.8}$$

Where the coefficients that appear mean :

- η_T efficiency due to the high turbulence level over complex terrain.
- $\eta_\rho = \rho/\rho_o$ air density variation.
- η_{avail} availability of the turbine.
- η_{wake} coefficient which is function of the position of the turbines and the airflow direction.

A.I.2 Betz Law

The energy that a turbine can extract from a fluid running through it is limited to 59,3 % of the kinetic energy of the fluid. That fact happens because if all the kinetic energy were extracted, it would mean that the wind speed at the exit of the turbine would be zero. So if the wind velocity is zero afterwards the flow will be stopped and no more fluid could go in the turbine, blocking it.

$$P(W) = W_1 - W_2 = \frac{1}{2} \cdot \dot{m} \cdot ({v_1}^2 - {v_2}^2) \qquad \textbf{Eqn A. I. 9}$$

$$v_1 = a \quad and \quad v_2 = a \cdot v \qquad \textbf{Eqn A. I. 10}$$

$$P(W) = \frac{1}{2} \cdot \dot{m} \cdot v^2 \cdot (1 - a^2) = \frac{1}{2} \cdot \rho \cdot A \cdot v^3 \cdot (1 - a^2) \cdot \left(\frac{1 + a}{2}\right) \qquad \textbf{Eqn A. I. 11}$$

$$If \quad \frac{dP}{da} = 0 \; when \; the \; maximun \; power \; is \; reached \qquad \textbf{Eqn A. I. 12}$$

$$a = \frac{1}{3} \quad so \quad (1 - a^2) \cdot \left(\frac{1 + a}{2}\right) = \frac{16}{27} = 0{,}593 \qquad \textbf{Eqn A. I. 13}$$

Where ‘a’ is called the Betz´s limit. And finally the maximum power developed by a wind turbine is:

$$P_{max}(W) = \frac{1}{2} \cdot \left(\frac{16}{27}\right) \cdot \rho \cdot A \cdot v^3 \qquad \textbf{Eqn A. I. 14}$$

APPENDIX II

Appendix II.1

Effect of turbulence in Time-Average Transport Equations **[2]**.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \qquad \textbf{Eqn A. II. 1}$$

$$\frac{\partial (\rho u)}{\partial t} + u\big(\nabla \cdot (\rho u)\big) = -\nabla \cdot p + \mu \nabla^2 u + S_M \qquad \textbf{Eqn A. II. 2}$$

$$\frac{\partial (\rho u)}{\partial t} + \nabla \cdot (\rho u.u) = -\frac{\partial p}{\partial x} + \mu \nabla^2 u + S_M \qquad \textbf{Eqn A. II. 2. a}$$

$$\frac{\partial (\rho v)}{\partial t} + \nabla \cdot (\rho v.u) = -\frac{\partial p}{\partial y} + \mu \nabla^2 v + S_M \qquad \textbf{Eqn A. II. 2. b}$$

$$\frac{\partial (\rho w)}{\partial t} + \nabla \cdot (\rho w.u) = -\frac{\partial p}{\partial z} + \mu \nabla^2 w + S_M \qquad \textbf{Eqn A. II. 2. c}$$

To start with, there is some rules which relates the time averages of the fluctuating properties $\varphi = \Phi + \varphi'$ and $\psi = \Psi + \psi'$ how they are combined (φ and ψ represent any flow property e.g. velocity).

$$\overline{\psi'} = \overline{\phi'} = 0 \qquad \textbf{Eqn A. II. 3}$$

Because is the time average of the fluctuating term that is zero by definition:

$$\overline{u'} = \frac{1}{\Delta t} \cdot \int_0^{\Delta t} u'(t)dt = 0 \qquad \textbf{Eqn A. II. 4}$$

$\overline{\phi} = \phi$; $\overline{\partial \psi}/\partial s = \overline{\partial \Psi}/\partial s$; $\overline{\int \psi . ds} = \int \Psi . ds$; $\overline{\phi + \Psi'} = \Phi + \Psi$; $\overline{\psi . \phi} = \Phi\Psi + \overline{\phi'\psi'}$;

$\overline{\phi . \Psi} = \Phi\Psi$; $\overline{\phi' . \Psi} = 0$

APPENDICES

Knowing that “m” is a vector and “ψ” represents a scalar.

$a = A + a'$ $\quad \psi = \Psi + \psi'$

$\overline{\nabla . \mathrm{a}} = \nabla . \mathrm{A}$; $\overline{\nabla . (\psi \mathrm{a})} = \nabla . (\overline{\psi \mathrm{a}}) = \nabla . (\Phi \mathrm{A}) + \nabla . (\overline{\psi' + \mathrm{a}'})$; $\nabla^2 \psi = \nabla^2 \Psi$

Using the Cartesians co-ordinates

$\mathrm{u} = (u, v, w)$ $\mathrm{U} = (U, V, W)$ $\mathrm{u}' = (u', v', w')$

$u = U + u'$

$v = V + v'$

$w = W + w'$

$\mathrm{p} = \mathrm{P} + \mathrm{p}'$

Now the time average equations are got thanks to the rules shown at before.

The mass conservation equation:

$$\overline{\frac{\partial \rho}{\partial \mathrm{t}}} + \overline{\nabla \cdot (\rho \mathrm{u})} = 0 \qquad \textbf{Eqn A. II. 5}$$

$$\frac{\partial \rho}{\partial \mathrm{t}} + \nabla \cdot (\rho \mathrm{U}) = 0 \qquad \textbf{Eqn A. II. 6}$$

The time average x-momentum equation:

$$\overline{\frac{\partial (\rho u)}{\partial \mathrm{t}}} + \overline{\nabla \cdot (\rho u . \mathrm{u})} = \overline{-\frac{\partial \mathrm{p}}{\partial \mathrm{x}}} + \overline{\mu \nabla^2 u} + \mathrm{S_M} \qquad \textbf{Eqn A. II. 7. a}$$

$\frac{\partial (\rho u)}{\partial \mathrm{t}} = \frac{\partial U}{\partial t}$; $\overline{-\frac{\partial \mathrm{p}}{\partial \mathrm{x}}} = -\frac{\partial \mathrm{P}}{\partial \mathrm{x}}$; $\overline{\nabla \cdot (\rho u . \mathrm{u})} = \nabla \cdot (\rho U . \mathrm{U}) + \nabla \cdot (\rho \overline{u' \mathrm{u}'})$;

$$\overline{\mu \nabla^2 u} = \mu \nabla^2 U$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho U . \mathrm{U}) + \boldsymbol{\nabla} \cdot (\boldsymbol{\rho \overline{u} \cdot \overline{\mathbf{u}}}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{x}} + \mu \nabla^2 U + \mathrm{S_M} \qquad \textbf{Eqn A. II. 8. a}$$

A new term appears in the momentum equation along the process of averaging. This term is the convective momentum that the velocity fluctuations produce, which usually is represented at the right side of the equation as additional turbulent stresses on the mean velocity components.

$$\nabla \cdot (\overline{u \cdot \mathrm{u}}) = \frac{\partial \overline{u'^2}}{\partial \mathrm{x}} + \frac{\partial \overline{u' v'}}{\partial \mathrm{x}} + \frac{\partial \overline{u' w'}}{\partial \mathrm{x}} \qquad \textbf{Eqn A. II. 9. a}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho U . \mathrm{U}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{x}} + \mu \nabla^2 U - \rho \left[\frac{\partial \overline{u'^2}}{\partial \mathrm{x}} + \frac{\partial \overline{u' v'}}{\partial \mathrm{x}} + \frac{\partial \overline{u' w'}}{\partial \mathrm{x}} \right] + \mathrm{S_M} \qquad \textbf{Eqn A. II. 10. a}$$

Doing the same process on the equations (Eqn A.II.2.b and Eqn A.II.2.c) the time average y- and z- momentum equations are got:

$$\frac{\partial \rho V}{\partial t} + \nabla \cdot (\rho V . \mathrm{U}) + \boldsymbol{\nabla} \cdot (\boldsymbol{\rho \overline{v} \cdot \overline{\mathbf{u}}}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{y}} + \mu \nabla^2 V + \mathrm{S_M} \qquad \textbf{Eqn A. II. 8. b}$$

$$\frac{\partial \rho W}{\partial t} + \nabla \cdot (\rho W . \mathrm{U}) + \boldsymbol{\nabla} \cdot (\boldsymbol{\rho \overline{w} \overline{\mathbf{u}}}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{z}} + \mu \nabla^2 W + \mathrm{S_M} \qquad \textbf{Eqn A. II. 8. c}$$

Finally the time averaged Navier-Stokes momentum equations are:

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U . \mathrm{U}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{x}} + \mu \nabla^2 U - \rho \left[\frac{\partial \overline{u'^2}}{\partial \mathrm{x}} + \frac{\partial \overline{u'v'}}{\partial \mathrm{y}} + \frac{\partial \overline{u'w'}}{\partial \mathrm{z}} \right] + \mathrm{S_M} \qquad \textbf{Eqn A. II. 10. a}$$

$$\frac{\partial(\rho V)}{\partial \mathrm{t}} + \nabla \cdot (\rho V . \mathrm{U}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{y}} + \mu \nabla^2 V - \rho \left[\frac{\partial \overline{v'u'}}{\partial \mathrm{x}} + \frac{\partial \overline{v'^2}}{\partial \mathrm{y}} + \frac{\partial \overline{v'w'}}{\partial \mathrm{z}} \right] + \mathrm{S_M} \qquad \textbf{Eqn A. II. 10. b}$$

$$\frac{\partial(\rho W)}{\partial \mathrm{t}} + \nabla \cdot (\rho W . \mathrm{U}) = -\frac{\partial \mathrm{P}}{\partial \mathrm{z}} + \mu \nabla^2 W - \rho \left[\frac{\partial \overline{w'u'}}{\partial \mathrm{x}} + \frac{\partial \overline{w'v'}}{\partial \mathrm{y}} + \frac{\partial \overline{w'^2}}{\partial \mathrm{z}} \right] + \mathrm{S_M} \qquad \textbf{Eqn A. II. 10. c}$$

These equations are called the Reynolds equations.There are six terms that appears after the time averaging process, these terms are known as the Reynolds stresses. Three of them are normal stresses and the other three are shear ones.

As the Reynolds stresses are unknown the development of a turbulence model is necessary in order to obtain these stresses accurately enough but without forgetting that for getting a very accurate solution the computational power needed will be unavailable.

Appendix II.2 Experimental researches for obtaining the drag coefficient (C_D)

1. Experimental research developed by D. Poggi, A. Porporato, C. Ridolfi, J.D. Albertson and G.G. Katul. *The effect of vegetation density on canopy sub-layer turbulence* **[4].**

   An experimental research was carried out, where the canopies systems were simulates using steel cylinders. The velocity was measured using a two-component Laser Doppler Anemometry (LDA).

   The effect of the canopy is also introduced as a source term into the momentum equation. In that experiment the C_D is related through the Reynolds number:

   - At low Re the C_D follows a classical distribution around a cylinder due to its drag behaviour.

$$C_D \cong Re^{1/2}$$

- If the Reynolds number is increased the C_D value goes down. The sheltering effect due to consecutive obstacles in the wind flow. From this experiment, an empirical relationship was extracted:

$$C_D(Re_d) = -8{,}5 \,.\, 10^{-4} . Re_d + 1{,}5$$

Where Re_d is the local Reynolds number.However, this equation is just an empirical one, so it has to be checked.

2. Experimental research done by A.Cescatti and B.Marcolla. "Drag coefficient and turbulence intensity in conifer canopies" **[6].**

The effect of the drag coefficient on the wind was experimentally studied in two forests at the Italian Alps. From the mean momentum equation C_D was extracted, through the source term that the canopy effect produces. It was tested two different ways of the momentum equation.

- The first one was the standard one where drag forces depend on the squared mean speed (U^2):

Assuming the common form of the momentum equation, the drag coefficient goes down when the wind speed is increased. The C_D values for both places are (0,15 ± 0,14 and 0,34 ± 0,45).

- The second way will imply that drag forces are defined in terms of the averaged product of longitudinal wind component (*u,v,w*) and the instantaneous wind intensity (u) :

According to the second alternative, no clear dependence can be observed on the wind intensity. The mean drag coefficients are (0.09 ± 0.06 and 0.12 ± 0.12).

Analysing these results the second approach looks more appropriate to get the drag coefficient.

APPENDIX III

In order to make the Canopy writer some files have to be codified with C++: *leafIndexToFoam.C ; createCdAlphaL.H ; Check_inside_cell.H and ReadmapfileCdAlphaL.H* . These files are placed in the *leafIndexToFoam* folder, and after being compiled they will create the *leafIndexToFoam* executable file.

Then just typing "leafIndexToFoam" in the terminal, the application will be executed. The source file *leafIndexToFoam.C* extension file will summon the other files needed by the program, also when the *Readmapfile_CdAlphaL.H* file starts, it will ask the user for the name of the map file.

Appendix III.1 leafIndexToFoam.C

```
/*---------------------------------------------------------------------------*\
========= |
\\ / F ield |
\\ / O peration |
\\ / A nd |
\\/ M anipulation |
-------------------------------------------------------------------------------
\*---------------------------------------------------------------------------*/
#include "wallFvPatch.H"
#include "fvCFD.H"
#include <fstream>
#include <stdlib.h>
#include <string.h>
using namespace std;
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
bool Check_inside(vector, List<vector>);
int main(int argc, char *argv[])
{
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
#include "createCdAlphaL.H"
#include "Readmapfile_CdAlphaL.H"
```

```
Info<<" Writing Cd / alphaL values "<<endl; //writing Cd and alphaL files.
Cd.write();
alphaL.write();
return(0);
}
bool Check_inside(vector point, List<vector> polygon){
int i,j=polygon.size()-2;
bool oddNodes=false;
for (i=0;i<polygon.size()-1;i++){
if(polygon[i][1]<point[1] && polygon[j][1]>=point[1] || polygon[j][1]<point[1] && polygon[i][1]>=point
[1]){
if(polygon[i][0]+(point[1]-polygon[i][1])/(polygon[j][1]-polygon[i][1])*(polygon[j][0]-polygon[i]
[0])<point[0]){
oddNodes=!oddNodes;
}
}
j=i;
}
return oddNodes; //true if the point is inside the polygon

}
// ************************************************************************* //
```

Appendix III.2 CreateCdAlphaL.H

```
Info << "Reading field Cd / alphaL" << endl;
volScalarField Cd
(
IOobject
(
"Cd",
runTime.timeName(),
mesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
mesh
);
volScalarField alphaL
(
IOobject
(
"alphaL",
```

```
runTime.timeName(),
mesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
mesh

);
```

Appendix III.3 Readmapfile_CdAlphaL.H

```
//********** Readmapfile_CdAlphaL.H *****************
fileName tmp;
tmp=args.path();
Info<<"Type the name of the map file..."<<endl;
char file[100];
tmp=tmp+"/";
cin.getline(file,100);
tmp=tmp+file;
char nam[300];
int p=0;
for (int i=0;i<100;i++){
if (file[i]=='.'){
p=i;
i=100;
}
}
int ii=tmp.size();
for (int i=0;i<ii;i++){
nam[i]=tmp[i];
if (tmp[i]==file[p] && tmp[i+1]==file[p+1] && tmp[i+2]==file[p+2] && tmp[i+3]==file[p+3]) {
nam[i+1]=tmp[i+1];
nam[i+2]=tmp[i+2];
nam[i+3]=tmp[i+3];
nam[i+4]='\0';
i=ii+4;
}
}
ii=0;
ifstream inFile;
inFile.open(nam);
if (!inFile){
cerr <<"Unable to open file "<<file<<endl;
```

```
return(0);
}
char c;
int a=0;
char d[100];
fileName cadena;
while (inFile){
if (a<4){ //We skip the first 4 lines of the map file
inFile.getline(d,99,'\n');
}
else{
inFile.get(c);
cadena+=c;
}
a++;
}
Info<<"File "<<file<<" has been read."<<endl;
Info<<tmp<<endl;
Info<<nam<<endl;
Info<<endl;
//Info<<cadena<<endl;
/*Info<<"Now we need to relate the map coordinate system to our mesh coordinate system. For that purpose
we will have to introduce some data"<<endl;*/
// This piece of code it will be used when real and complex cases were studied

scalar offset_x=0;
scalar offset_y=0;
scalar a1=1;
scalar b1=0;
scalar a2=0;
scalar b2=1;
/*Info<<endl<<"* Introduce the offset between both coordinate system origins (x0,y0)mesh=offset+(x0,y0)
map :(the offset must be referred to mesh coordinates) "<<endl;
Info<<"x_value: ";
cin>>offset_x;
Info<<"y_value: ";
cin>>offset_y;
Info<<endl<<"Now introduce the values (x,y) with respect to mesh coordinate system for the point (1,0) in
map file"<<endl;
Info<<"x_value: ";
cin>>a1;
Info<<"y_value: ";
cin>>b1;
Info<<endl<<"To end introduce the values (x,y) with respect to mesh coordinate system for the point (0,1)
```

```
in map files"<<endl;
Info<<"x_value: ";
cin>>a2;
Info<<"y_value: ";
cin>>b2;
Info<<endl;*/
Info<<cadena<<endl;
int cad_size=cadena.size();
int n_polygons=0;
long i=0;
List <scalar> CdAlphaL;
CdAlphaL.setSize(5);
while(i<cad_size-3){
Info<<"Getting Cd, alphaL and number of vertices of each polygon"<<endl;
Info<<"cad_size="<<cad_size<<", i="<<i<<endl;
int ni=0;
char cd[30];
while (cadena[i]!=' '){ //Getting Cd
cd[ni]=cadena[i];
ni++;
i++;
}
scalar cd_s=atof(cd);
Info<<"cd_s="<<cd_s<<", i="<<i<<endl;
i++;
ni=0;
char alpha[30];
while (cadena[i]!=' '){ // Getting alphaL
alpha[ni]=cadena[i];
ni++;
i++;
}
scalar alphaL_s=atof(alpha);
Info<<"alphaL_s="<<alphaL_s<<", i="<<i<<endl;
i++;
ni=0;
char deltaz[30];
while (cadena[i]!=' '){ // Getting deltaz
deltaz[ni]=cadena[i];
ni++;
i++;
}
scalar deltaz_s=atof(deltaz);
Info<<"deltaz_s="<<deltaz_s<<", i="<<i<<endl;
i++;
```

```
ni=0;
char z0[30];
while (cadena[i]!=' '){ // Getting z0
z0[ni]=cadena[i];
ni++;
i++;
}
scalar z0_s=atof(z0);
Info<<"z0_s="<<z0_s<<", i="<<i<<endl;
i++;
ni=0;
char nvertex[30];
while (cadena[i]!='\n'){ //Getting number of vertices
nvertex[ni]=cadena[i];
ni++;
i++;
}
scalar nvertex_s=atoi(nvertex);
Info<<"nvertex_s="<<nvertex_s<<", i="<<i<<endl;
List <vector> p_polygon;
p_polygon.setSize(nvertex_s);
CdAlphaL[0]=cd_s;
//cout<<CdAlphaL[0]<<endl;
CdAlphaL[1]=alphaL_s;
CdAlphaL[2]=deltaz_s;
CdAlphaL[3]=z0_s;
CdAlphaL[4]=nvertex_s;
//cout<<CdAlphaL[2]<<endl;
i++;
if (cadena[i]=='/') i=cad_size;
for (int j=0;j<nvertex_s;j++){
ni=0;
char x1[30];
while (cadena[i]!=' '){
x1[ni]=cadena[i];
ni++;
i++;
}
i++;
x1[ni]='\0';

ni=0;
char y1[30];
while (cadena[i]!='\n'){
y1[ni]=cadena[i];
```

```
ni++;
i++;
}
y1[ni]='\0';
scalar x1_s=offset_x+(a1-offset_x)*atof(x1)+(a2-offset_x)*atof(y1);
scalar y1_s=offset_y+(b1-offset_y)*atof(x1)+(b2-offset_y)*atof(y1);
p_polygon[j][0]=x1_s;
p_polygon[j][1]=y1_s;
//cout<<y1_s<<endl;
p_polygon[j][2]=0; // We dont use it.Cd/alphaL information is 2D
i++;
if (cadena[i]=='/') i=cad_size;
}
Info<<p_polygon<<endl;
#include "Check_inside_cell.H"
n_polygons++;
}
Info<<CdAlphaL<<endl;
inFile.close(); //We close the file
```

Appendix III.4 Check_inside_cell.H

```
//*************** Check_inside_cell.H ****************
forAll(mesh.cells(), celli)
{
Vector <scalar> centre_pointi;
centre_pointi[0]=mesh.C()[celli].component(vector::X);
centre_pointi[1]=mesh.C()[celli].component(vector::Y);
centre_pointi[2]=mesh.C()[celli].component(vector::Z);
bool inside=Check_inside(centre_pointi,p_polygon);
if (inside){
if (centre_pointi[2]>=CdAlphaL[3] && centre_pointi[2]<=CdAlphaL[3]+CdAlphaL[2]) {
cout<<"Inside: "<<centre_pointi[0]<<", "<<centre_pointi[1]<<", "<<centre_pointi[2]<<endl;
Cd.internalField()[celli]=CdAlphaL[0];
alphaL.internalField()[celli]=CdAlphaL[1];
}
}
}
```

Appendix III.5 tree.map (Map file example)

+ DXF-file: Mathematica written
0.0 1.0 0.0 1.0
1.0 0.0 1.0 0.0
1.0 0.0

} These lines will be skipped.

0.15 0.919 10 15 6 C_D; α; Δz; z_o values and the number of vertex which make the first polygon.

0 20
-40 20
-40 0
-40 -20
0 -20
0 0

} X-Y coordinates which set the first polygon´s vertexes.

0.45 0.148 15 15 6 C_D; α; Δz; z_o values and the number of vertex which make the second polygon

50 20
40 20
40 0
40 -20
50 -20
50 0

} X-Y coordinates which set the second polygon´s vertexes.

Appendix III.6 Cd and alphaL initial files.

Cd initial file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| ========= | |
| \\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\ / O peration | Version: 2.1.0 |
| \\ / A nd | Web: www.OpenFOAM.org |
| \\/ M anipulation | |
\*---------------------------------------------------------------------------*/
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
location "0";
object Cd;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 0 0 0 0 0 0];
internalField uniform 0;
boundaryField
{
ground
{
type fixedValue;
value uniform 0;
}
top
{
type fixedValue;
value uniform 0;
}
inlet
{
type fixedValue;
value uniform 0;
}
outlet
{
type fixedValue;
value uniform 0;
}
```

```
bolund05.stl_bolund05
{
type fixedValue;
value uniform 0;
}
}
// ************************************************************************* //
```

alphaL initial file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| ========= | |
| \\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\ / O peration | Version: 2.1.0 |
| \\ / A nd | Web: www.OpenFOAM.org |
| \\/ M anipulation | |
\*---------------------------------------------------------------------------*/
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
location "0";
object alphaL;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 -1 0 0 0 0 0];
internalField uniform 0;
boundaryField
{
ground
{
type fixedValue;
value uniform 0;
}
top
{
type fixedValue;
value uniform 0;
}
inlet
{
type fixedValue;
value uniform 0;
```

```
}
outlet
{
type fixedValue;
value uniform 0;
}
bolund05.stl_bolund05
{
type fixedValue;
value uniform 0;
}
}
// ************************************************************************* //
```

APPENDIX IV

Appendix IV.1 ControlDict.

```
// -*- C++ -*-
// File generated by PyFoam - sorry for the ugliness

FoamFile
{
 version 2.0;
 format ascii;
 class dictionary;
 location "system";
 object controlDict;
}

application windFoam;

startFrom latestTime;

startTime 0;

stopAt endTime;

endTime 1000;

deltaT 1;

writeControl timeStep;

writeInterval 100;

purgeWrite 0;

writeFormat ascii;

writePrecision 6;

writeCompression uncompressed;

timeFormat general;

timePrecision 6;

runTimeModifiable yes;

libs
  (
    "libgroovyBC.so"
  );

functions
```

```
{
  probes
  {
    type probes;
    functionObjectLibs
      (
        "libsampling.so"
      );
    enabled true;
    outputControl timeStep;
    outputInterval 1;
    probeLocations
      (
        (0.0254 0.0253 20)
        (0.0508 0.0253 100)
      );
    fields
      (
        p
        k
        U
      );
  }
}     //
************************************************************ //
```

Appendix IV.2 fvSchemes

```
/*--------------------------------*- C++ -*---------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.6
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*--------------------------------------------------------------
-------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

```
ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwindV cellLimited Gauss linear
1;
    div(phi,k)      Gauss linearUpwind cellLimited Gauss linear
1;
    div(phi,epsilon) Gauss linearUpwind cellLimited Gauss linear
1;
    div(phi,R)      Gauss linearUpwind cellLimited Gauss linear
1;
    div(R)          Gauss linear;
    div(phi,nuTilda) Gauss upwind;
    div((nuEff*dev(grad(U).T()))) Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nuEff,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DkEff,k) Gauss linear corrected;
    laplacian(DepsilonEff,epsilon) Gauss linear corrected;
    laplacian(DREff,R) Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
}

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

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

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

Appendix IV.3 fvSolution

```
/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.6
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*-----------------------------------------------------------------
-------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.1;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
```

```
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }

    k
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }
   epsilon
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }


    nuTilda
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

relaxationFactors
{
    default         0;
    p               0.3;
    U               0.6;
    k               0.6;
    epsilon         0.6;
    nuTilda         0.5;
}


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

Appendix IV.4 transportProperties

```
// -*- C++ -*-
// File generated by PyFoam - sorry for the ugliness

FoamFile
{
 version 2.0;
 format ascii;
 class dictionary;
 location "constant";
 object transportProperties;
}

transportModel Newtonian;

rho rho     [ 1 -3 0 0 0 0 0 ]     1.229 ;

nu nu     [ 0 2 -1 0 0 0 0 ]     0.000125 ;

latitude latitude     [ 0 0 0 0 0 0 0 ]     0 ;

gravity gravity     [ 0 0 0 0 0 0 0 ]     0 ;

CrossPowerLawCoeffs
{
  nu0 nu0       [ 0 2 -1 0 0 0 0 ]       1e-06 ;
  nuInf nuInf       [ 0 2 -1 0 0 0 0 ]       1e-06 ;
  m m       [ 0 0 1 0 0 0 0 ]       1 ;
  n n       [ 0 0 0 0 0 0 0 ]       1 ;
}

BirdCarreauCoeffs
{
  nu0 nu0       [ 0 2 -1 0 0 0 0 ]       1e-06 ;
  nuInf nuInf       [ 0 2 -1 0 0 0 0 ]       1e-06 ;
  k k       [ 0 0 1 0 0 0 0 ]       0 ;
  n n       [ 0 0 0 0 0 0 0 ]       1 ;
}    //
************************************************************* //
```

Appendix IV.5 turbulenceProperties

```
// -*- C++ -*-
// File generated by PyFoam - sorry for the ugliness

FoamFile
{
```

```
 version 2.0;
 format ascii;
 class dictionary;
 location "constant";
 object RASProperties;
}

RASModel kEpsilon;

turbulence on;

printCoeffs on;

laminarCoeffs
{
}

kEpsilonhigCoeffs
{
  Cmu 0.033;
  C1 1.2081852929;
  C2 1.92;
  alphaEps 0.76923;
  C4 0;
  C5 0;
  beta_p 0;
  beta_d 0;
}

wallFunctionCoeffs
{
  kappa 0.41;
  E 9;
}    //
********************************************************** //
```

Appendix IV.6 *points*

```
/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "constant/polyMesh";
    object      points;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


746241
(
(-160 -160 0)
(-160 160 0)
(160 160 0)
(160 -160 0)
(-282.8427136 -282.8427114 0)
(282.8427195 -282.8427054 0)
(282.8427147 282.8427102 0)
….

(-276.7543588 -268.3972462 185.2542663)
)


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

Appendix IV.7 *faces*.

```
/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*-------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       faceList;
    location    "constant/polyMesh";
    object      faces;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


2186000
```

```
(
4(45178 15776 68800 87891)
4(72860 87891 68800 52735)
4(15875 45178 87891 72860)
….

4(812 18 11975 13895)
)


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

Appendix IV.8 *boundary*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

4
(
    pressure_outlet.6
    {
        type            patch;
        nFaces          8000;
        startFace       2134000;
    }
    inlet
    {
        type            patch;
        nFaces          8000;
        startFace       2142000;
    }
    ground
```

```
    {
        type            wall;
        nFaces          18000;
        startFace       2150000;
    }
    top
    {
        type            patch;
        nFaces          18000;
        startFace       2168000;
    }
)

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

Appendix IV.9 *owner*

```
/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*-------------------------------------------------------------------
-------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       labelList;
    note        "nPoints:746241  nCells:720000  nFaces:2186000
nInternalFaces:2134000";
    location    "constant/polyMesh";
    object      owner;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

2186000
(
0
0
0
0
0
0
1
```

```
1
1
1
1
2
2
2
2
2
3
3
3
3
3

…

719980
719981
719982
719983
719984
719985
719986
719987
719988
719989
719990
719991
719992
719993
719994
719995
719996
719997
719998
719999
)


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

APPENDIX V

Mesh generation using the blockMesh utility.

In the constant/polyMesh directory of a case a dictionary file named *blockMeshDict* is placed. This file is searched by blockMesh tool and read it, generating the mesh and writing out the mesh data to points, faces, cells and boundary files in the directory where the *blockMeshDict* is.

The blockMesh tool decomposes the domain geometry into several hexahedral 3-D blocks. The edges of each block can be straight lines, arcs or splines. Every block is defined by 8 vertices, and the vertices are in a list so they can be accessed by its label.

To define the mesh using blockMesh the *blockMeshDict* file has to be made. An example can be seen in below.

Keyword	Description
convertToMeters	Scaling factor for the vertex coordinates
vertices	List of vertex coordinates
edges	Used to describe arc or spline edges
block	Ordered list of vertex labels and mesh size
patches	List of patches
mergePatchPairs	List of patches to be merged

Table A.V.1 BlockMesh keywords [14].

The next example belongs to the cavity case in the IcoFoam folder. It is a really easy case and it is fully developed in the tutorials of OpenFOAM **[14]**.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 | |
| \\      / F ield          | OpenFOAM: The Open Source CFD Toolbox |
| \\    / O peration        | Version: 2.1.0 |
| \\  / A nd                | Web: www.OpenFOAM.org |
| \\/ M anipulation         | |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
version 2.0;
format ascii;
class dictionary;
object blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.1;
vertices
(
(0 0 0)     //vertex 0
(1 0 0)     //vertex 1
(1 1 0)     //vertex 2
(0 1 0)     //vertex 3
(0 0 0.1)  //vertex 4
(1 0 0.1)  //vertex 5
(1 1 0.1)  //vertex 6
(0 1 0.1)  //vertex 7
);
blocks
(
hex
 (0 1 2 3 4 5 6 7)          // vertex numbers
 (20 20 1)                  // number of cells in each direction
 simpleGrading (1 1 1)    // cell expansion ratios

);
edges
(
);
boundary
(
movingWall
{
type wall;
faces
(
(3 7 6 2)
);
}
fixedWalls
{
type wall;
faces
(
```

```
(0 4 7 3)
(2 6 5 1)
(1 5 4 0)
);
}
frontAndBack
{
type empty;
faces
(
(0 3 2 1)
(4 5 6 7)
);

}
);
mergePatchPairs
(
);
// ************************************************************************* //
```

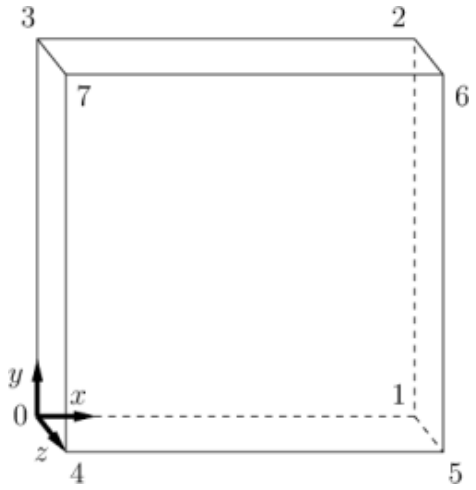


Fig A.V.1 Mesh generated with blockMesh utility (cavity case) [14].

APPENDIX VI

APPENDIX VI.1 *U* file

```
/*------------------*- C++ -*-------------------*\

FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "2";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   nonuniform List<vector>
136612
(
(3.81952 3.81952 0)
(3.77461 3.77461 0)
(3.77227 3.77227 0)
(3.77092 3.77092 0)
(3.76965 3.76965 0)
(3.76842 3.76842 0)
(3.76727 3.76727 0)
(3.76614 3.76614 0)
(3.76504 3.76504 0)
…

(3.80091 3.80091 0)
(4.66752 4.66752 0)
(4.66723 4.66723 0)
(3.80094 3.80094 0)
(4.66679 4.66679 0)
(4.66698 4.66698 0)
(3.80294 3.80294 0)
)
;
    }
    outlet
    {
        type            zeroGradient;
    }
    bolund05.stl_bolund05
    {
        type            fixedValue;
        value           uniform (0 0 0);
```

```
    }
}


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

APPENDIX VI.2 *p* file

```
// -*- C++ -*-

FoamFile
{
 version 2.0;
 format ascii;
 class volScalarField;
 location "2";
 object p;
}

dimensions [ 0 2 -2 0 0 0 0 ];

internalField uniform 0;

boundaryField
{
  inlet
  {
    type zeroGradient;
  }
  top
  {
    type fixedValue;
    value uniform 0;
  }
  ground
  {
    type zeroGradient;
  }
  outlet
  {
    type fixedValue;
    value uniform 0;
  }
  bolund05.stl_bolund05
  {
    type zeroGradient;
  }
}    //
************************************************************* //
```

APPENDIX VI.3 *Cd* and *alphaL* files.

The code below shows the *Cd* and *alphaL* files after the application "leafIndexToFoam" was run. So it has to be compared against the code shown on the appendix III.6.

```
/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------
-------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      Cd;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   nonuniform List<scalar>
136612
(
0
0
0
…

0
0
0.15
0
0
0.15
0.15
0
0.15
0.15
0
0
0.15
0.15
0
0.15
0.15
```

```
0
0
0.15
0.15
0
0
0
0.15
0.15
0.15
0.15
0
0
0
0.15
0.15
0.15
0.15
0
0
0

…

0
0
0
)
;

boundaryField
{
    ground
    {
        type            fixedValue;
        value           uniform 0;
    }
    top
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    bolund05.stl_bolund05
    {
        type            fixedValue;
        value           uniform 0;
```

```
    }
}


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



/*--------------------------------*- C++ -*----------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  2.1.0
|
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|
|    \\/     M anipulation  |
|
\*-------------------------------------------------------------
-------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alphaL;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * //

dimensions      [0 -1 0 0 0 0 0];

internalField   nonuniform List<scalar>
136612
(
0
0
0
…

0
0
0
0.919
0
0
0.919
0.919
0
0.919
0.919
0
0
```

```
0.919
0.919
0
0.919
0.919
0
0
0.919
0.919
0
0
0
0.919
0.919
0.919
0.919
0
0
0
0.919
0.919
0.919
0.919
0
0
0
…

0
0
0
)
;

boundaryField
{
    ground
    {
        type            fixedValue;
        value           uniform 0;
    }
    top
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
```

```
    bolund05.stl_bolund05
    {
        type            fixedValue;
        value           uniform 0;
    }
}


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

APPENDIX VII

Memoria original en lengua inglesa.

FACULTY OF ENGINEERING AND SUSTAINABLE DEVELOPMENT

MODELLING WIND FLOW THROUGH CANOPY SYSTEMS USING OPENFOAM.

Jose Miguel Maldonado

June 2012

Master's Thesis

Master Programme in Energy Systems
Examiner: Taghi Karimipanah
Supervisor: Mathias Cehlin

PREFACE

I would like to thank everybody who has supported me along this time working in my thesis. Especially my supervisor Mathias who not only helped me when I was stuck but also shared with time talking about several energy topics and some sportive ones, showing me that your supervisor can become more than just a teacher. He taught me that a better world, energetically speaking, can be reached if people have faith on it.

Also my classmates and friends have represented a trustful support when I was overload or worried while I was working.

For that reason I am very grateful for being given the chance of studying here in Sweden specifically in Gävle.

ABSTRACT

The most proper emplacements for set a wind farm are already taken or cannot be used for environmental causes. So in order to check the viability of the complex terrain locations which are still available Computational Fluid Dynamics tools are used. As the commercial codes are not flexible enough and very expensive, an open software will be used OpenFOAM.

OpenFOAM needs a code for accomplish the simulation; this code is programmed in C++. The terrain roughness, the Coriolis force and the gravity force were developed, so the next step will be to include the effect of canopies systems in the flow simulations.

Although it could be considered as roughness, it is suggested to add a forest canopy model in order to forecast the behaviour of the wind flow over the forests.

Along this document it will be shown the process followed in order to insert the canopies systems in the CFD software. This achievement has two mains goals:

- Pre-processing tool which will insert the canopy parameters in the mesh of the domain. This application will situate the forest along the studied case.
- The second goal is to develop a solver which take into account the effect caused by the canopy.

Once both of them are made, as there is no software which includes this kind of obstacles in the airflow, the results just can be checked by an experimental research but that experiment is suggested as future work because it is out of this thesis. So it will be checked that the canopy parameters are uploaded to the case, and that the airflow is disturbed in a consequently way by any forest.

Table of Contents

List of figures

List of Tables

Introduction

1.1. Wind Power

Wind power is an abundant, renewable and clean energy source. It takes energy out from the wind. The drawback of this energy id that is not a reliable way to supply electricity because if it is not windy enough or it is too much windy, the wind-farms will be stopped and an alternative power plant will have to provide the electricity demand. So even if the whole electricity demand is covered by wind power, another kind of power plant, which could produce electricity when it is commanded, must be installed just in case the wind-farms could not work.

The wind power is one of the most famous renewable energies, because the society relates it with the green energy. When the project that a company wants to carry out is environmentally friendly and socially accepted, it will be easier to develop. Once the windmills are installed, no fuel is needed for running the wind mill which generates the electricity and no contaminants will be emitted by them, which allow you to reduce the emissions of greenhouse gasses if wind-farms are settled replacing thermal power stations which run with fossil fuels.

In Sweden wind power is not a really relevant electricity source but as it is a renewable energy wind farm are still being settled and studies in order to improve their efficiency are still being developed. The energy extracted from the windmills is limited by the Betz´s law (see APPENDIX I), that energy is function of the cube of the wind speed, also it depends on the windmill size and the efficiency of the electric generator, but mainly it depends on the wind speed. Hence to find a good spot where the wind speed is as high as possible and also stable along the time is rather important. For that reason the CFD programs are used in order to study possible sites for setting new wind farms.

The best place for setting a wind farm usually is offshore, because the water surface has less roughness than land and the wind speed is higher and more stable and no obstacles disturb the wind. Denmark was the first country that started installing offshore wind farms in 1991 since that day 39 offshore wind farm has been arisen in Europe around Denmark, Belgium, Finland, Germany, Ireland, Holland, Norway, Great Britain and Sweden. The problem is that the whole coast cannot be used for that purpose, so wind farms are still being installed onshore like at the beginning.

Due to that almost all the proper settlements for install wind farms are already taken or can not be used due to environmental reasons; several wind mills for electric generation have to be settled near to a forest edge. The forest density affects the airflow when it is flowing through the canopy, the air speed goes down and its turbulence increases. That is the reason why some studies have been developed in order to know how the airflow behaves in that situation and how the aero generator set near forest is affected.

Nowadays the wind power is not very relevant in the electricity production along the year, especially in Sweden where just around 1, 8 % of the total electricity production comes from wind farms **[1]** but for instance in Spain or Denmark the wind power produces around 13% and 18% respectively of the total electricity production, which means that this kind of energy is taken into account. However, as the Swedish government is trying to achieve the electric demand without nuclear power plants, **[16]** because it was supposed that by 2010 they would be phased out, the number of wind farms will increase along the next years in order to supply the electricity that nuclear power is producing by now.

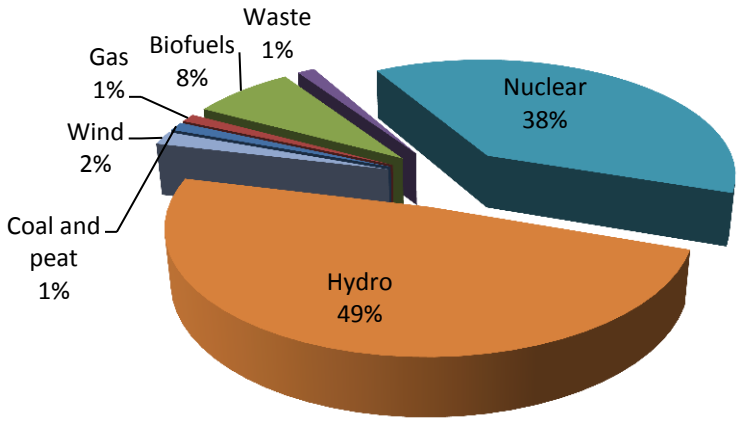


Fig 1.1 Electricity production in Sweden along 2009.

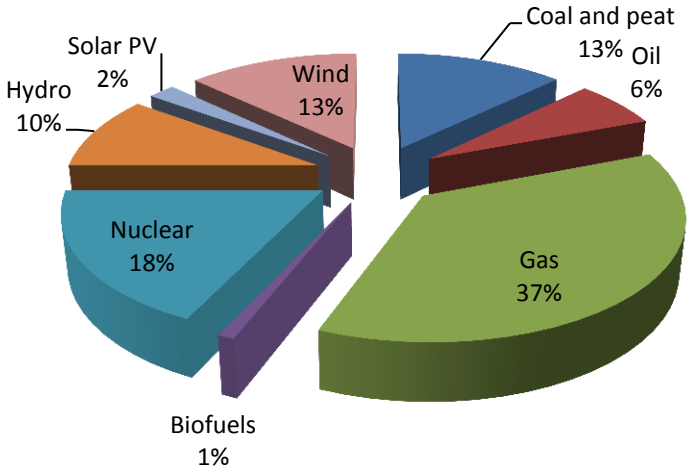


Fig 1.2 Electricity production in Spain along 2009.

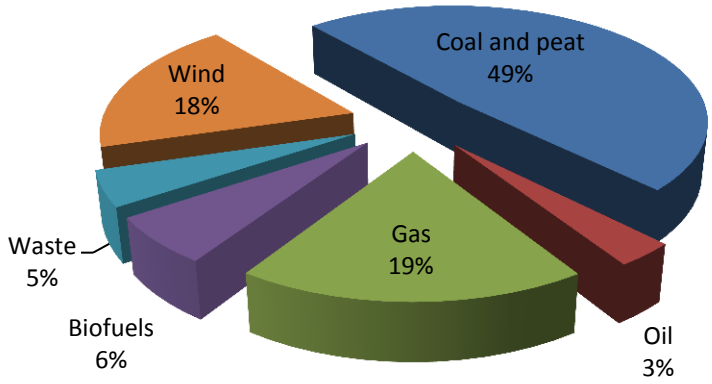


Fig 1.3 Electricity production in Denmark along 2009.

1.2. Aims and tasks

The main task is to develop CFD tools which allow you to determine the best spot for setting a wind farm over complex terrain, and how the wind mills have to be installed. Complex terrain means that roughness, canopy and also the other wind mills have to be taken into account. In this project the main achievement is to add the drag coefficient (C_D) and the leaf area density (α) as calculation variables in the OpenFOAM solver.

Task 1: Pre-processing. Develop the current OpenFOAM code in order to insert the canopy. This application will provide a tool for adding automatically the C_D and α from a given data (in map file format) to OpenFOAM files.

Task 2: Develop an application which simulates the wind flow which is affected by the drag force due to canopies. The effect of the canopy will be added by a source term in the momentum equations in the Navier-Stokes equations.

1.3. Method

OpenFOAM is the CFD program used in this project, this tool works over Linux operative system (O.S), so first of all the users has to get used to work with these kind of O.S.

A mesh will be given in order to test the new code; this mesh simulates the Bolund hill situated at the coast, at the north of Risø DTU. The mesh is provided by FLUENT and by the application *FluentToFoam*, the mesh will be converted so that OpenFOAM can work with it.

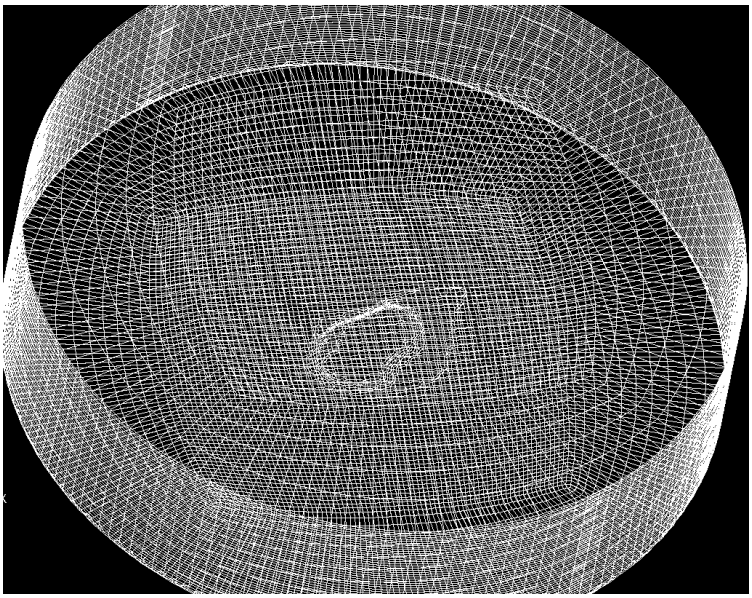


Fig 1.4 Mesh of Bolund hill in OpenFOAM.

Once the mesh is done, the next step will be created a map file which includes the canopy along the domain.

Lastly, it has to be tested if both applications work properly. The first one which will be called “leafIndexToFoam”, after you run it, it has to be checked if the canopies systems are well loaded according to the map file. The “treeFoam” solver will be tested by analysing the wind flow behaviour is several simulations:

Simulation 1: Solve the case using “simpleFoam”, a standard steady-state solver for incompressible, turbulent flow, provided by OpenFOAM. The data obtained in this simulation are valid.

Simulation 2: Solve the case using “treeFoam”, but the α value is null. This way the first simulation can be compared against the second one.

Simulation 3: First of all, the forest/s are added to the domain by “leafIndexToFoam”, then the solver “treeFoam” is run in order to check if the effect of the canopy in the airflow works in a proper way. Finally, the α value will increase to see how the canopies systems disturb the wind flow when they become denser.

2. THEORY

2.1. Atmospheric boundary layer (ABL)

Near the boundary layer of Earth there are several forces that disturb the airflow, as frictional drag, temperature, moisture, pollutants and so on. The lowest part of the atmosphere is directly affected by the planetary surface; this piece of atmosphere is called atmospheric boundary layer. The changes that Earth´s surface produces on the atmosphere take around an hour. The atmospheric boundary layer can extend from hundred metres to some kilometres, it depends on the latter forces.

There are two main parts in the ABL, the outer region also known as Ekman layer and the inner region (known as wall or surface layer). The inner region mainly depends on the surface geometry; however the outer one, where the surface is hardly important, the airflow is ruled by the pressure and the Coriolis force. Once the wind speed does not depend on the height, the outer region is over and the free atmosphere is reached.

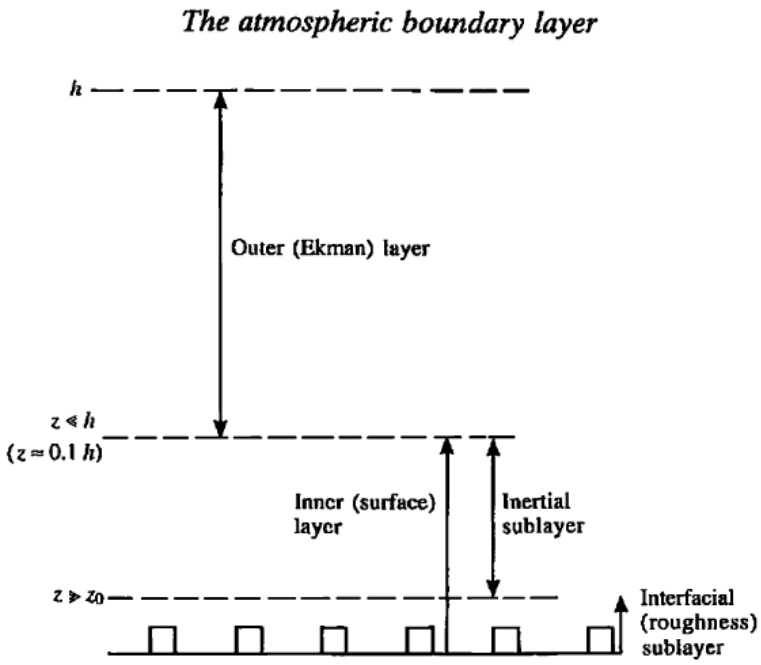


Fig 2.1 Atmospheric boundary layer structure [2].

An overlap region is produce along the transition between the outer and inner regions.

The surface effects can be felt at the Interfacial sub layer. In this layer as heat and mass are exchanged between the airflow and the surface, the molecular diffusion process is important. Close to the surface, the diurnal cycle and the clouds are really relevant on the structure of ABL turbulence, especially over the land where the heat exchanging with the air is higher.

The stability of the atmosphere can be divided in three:

- **Stable:** the heavy particles are placed under the light ones. If the particle is disturbed by the surface, it will be back to its original position.
- **Neutral:** the density does not change along the particles with the height, so there is no temperature difference (no stratified fluid). Once the height of the particle is changed, the particle will remain at that height.
- **Unstable:** the lightest particles are found below the heaviest ones. The particle will be displaced upwards.

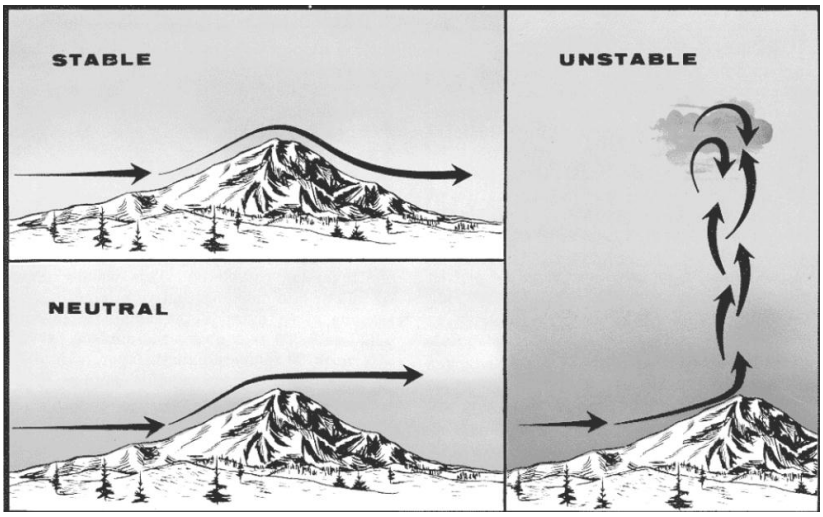


Fig 2.2 Stability of the atmosphere [9].

These three situations depend on the temperature of the airflow, because assuming that the atmospheric fluid is made with particles of different densities, the ones that have higher density will tend to go down and go up those of lower density (the fluid is said to be stratified).

Along this thesis, neutral conditions are considered because density is assumed constant. A really accurate solver would consider how density varies with height and temperature.

2.2. Turbulence

2.2.1. Definition of turbulence

The inertial forces, which are associated with convective effects, and viscous forces are related with the Reynolds number of the flow (Re). There are three regimes of flow: laminar, transient and turbulent.

The first one is the easiest situation and it takes place when the Reynolds number of the flow is below the so called critical Reynolds number (Re_{crit}). The flow motion is smooth, stratified and adjacent layers of fluid slide past each other in an orderly fashion. The flow goes in parallel lines without lateral mixing and each particle of the flow follows a path called, streamline.

However in most of the engineering situations the flow becomes unstable and the laminar regimen can not be used. If the value of the Re is above the Re_{crit} the flow will behave in a random and chaotic way, the flow properties as velocity and pressure are changing constantly along the time, these kinds of regimes are called turbulent flows.

The last one, the transient regime is just the step between the latter two.

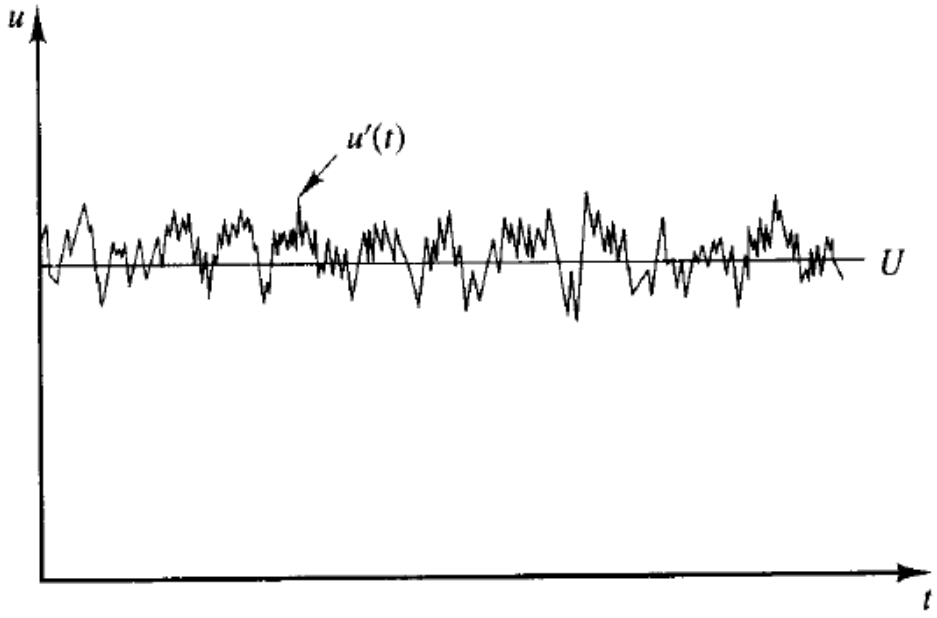


Fig 2.3 Velocity measurements in a turbulent flow [9].

At the latter picture (Fig 2.2) it can be observed the velocity measurement for a typical turbulence flow. In order to deal with turbulent flows, the velocity (u) is decomposed in two components, one steady mean value (U) and the fluctuating value (u´). The fluctuating values are always a three dimensional parameter even if the mean values varies in just one or two dimensions.

Besides, in turbulent regimes rotational flows structures, which are called turbulent eddies, with a wide range of length scales can be found. The eddying motions can join two particles which were separated by a long distance; hence heat, momentum and mass can be exchanged very effectively.

The largest eddies neglect the viscous effects and they are just governed by inertia effects. These eddies are stretched by the mean flow because their scale is as high as the mean flow; and the energy that is taken out during the vortex stretching keeps the turbulence. However the largest eddies are unstable by them so they split up into smaller eddies due to the interaction between them. The new smaller eddies will be divide as well making smaller eddies and so on, this effect is called energy cascade.

The smallest eddies which have a Re around 1 will be dissipated by the viscous stresses because at these scales (length on the order of 01, to 0,01 mm) the viscous effects are becoming relevant. The energy that made the eddy motion is converted into thermal internal energy, heating the flow up.

2.2.2. Effect of turbulence in Time-Average Transport Equations

To know how eddies behave in a turbulence flow is not an easy task; moreover there is no such a computational power which could solve the time dependant Navier-Stokes equations of turbulent flows completely detailed at high Reynolds numbers.

In order to study the turbulent processes, computational procedures are needed and these ones have to avoid the need to predict each eddy. Usually the time-averaged properties of the flow, as mean velocity or mean stresses, provide good enough information for the CFD users.

The flow properties as velocity (u) or pressure (p) can be settled as the sum of a steady mean component (U or P) plus a time-varying fluctuating one (u´ or p´), so:

$$u = U + u' \qquad \textbf{Eqn 2.1}$$

$$p = P + p' \qquad \textbf{Eqn 2.2}$$

The time average of the fluctuations components are zero by definition.

$$\overline{u'} = \frac{1}{\Delta t} \cdot \int_0^{\Delta t} u'(t)dt = 0 \qquad \textbf{Eqn 2.3}$$

From the root-mean-square (rms) of the fluctuations, the fluctuating component can be got

$$u_{rms} = \sqrt{\overline{(u')^2}} = \left[\frac{1}{\Delta t} \cdot \int_0^{\Delta t} (u')^2 dt = 0\right]^{1/2} \qquad \textbf{Eqn 2.4}$$

Thanks to a velocity probe sensitive to the turbulent fluctuations (for instance a hot-wire anemometer) and a simple electrical circuit, the rms of the velocity can be measured and after that the fluctuating part can be extracted.

Combining the Navier-Stokes equations with the fluctuating part of the flow, the effect of the turbulence over the flow can be studied (see APPENDIX II for the whole demonstration).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \qquad \textbf{Eqn 2.5}$$

$$\frac{\partial (\rho u)}{\partial t} + u\big(\nabla \cdot (\rho u)\big) = -\nabla \cdot p + \mu \nabla^2 u + S_M \qquad \textbf{Eqn 2.6}$$

Where ρ means the density of the fluid and S_M stands for source terms.

Using the Cartesians co-ordinates

$u = (u, v, w)$ $U = (U, V, W)$ $u' = (u', v', w')$

$u = U + u'$

$v = V + v'$

$w = W + w'$

$p = P + p'$

The time averaged Navier-Stokes equations are:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \qquad \textbf{Eqn 2.7}$$

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U . U) = -\frac{\partial P}{\partial x} + \mu \nabla^2 U - \left[\frac{\partial \rho \overline{u'^2}}{\partial x} + \frac{\partial \rho \overline{u'v'}}{\partial y} + \frac{\partial \rho \overline{u'w'}}{\partial z}\right] + S_M \quad \textbf{Eqn 2.8.a}$$

$$\frac{\partial(\rho V)}{\partial t} + \nabla \cdot (\rho V . U) = -\frac{\partial P}{\partial y} + \mu \nabla^2 V - \left[\frac{\partial \rho \overline{v'u'}}{\partial x} + \frac{\partial \rho \overline{v'^2}}{\partial y} + \frac{\partial \rho \overline{v'w'}}{\partial z}\right] + S_M \quad \textbf{Eqn 2.8.b}$$

$$\frac{\partial(\rho W)}{\partial t} + \nabla \cdot (\rho W . U) = -\frac{\partial P}{\partial z} + \mu \nabla^2 W - \left[\frac{\partial \rho \overline{w'u'}}{\partial x} + \frac{\partial \rho \overline{w'v'}}{\partial y} + \frac{\partial \rho \overline{w'^2}}{\partial z}\right] + S_M \quad \textbf{Eqn 2.8.c}$$

These equations are called the Reynolds equations. There are six terms that appears after the time averaging process, these terms are known as the Reynolds stresses. Three of them are normal stresses and the other three are shear ones.

As the Reynolds stresses are unknown the development of a turbulence model is necessary in order to obtain these stresses accurately enough but without forgetting that for getting a very accurate solution the computational power needed will be unavailable.There are several models for turbulence flows.

2.3. Numerical Modelling

2.3.1. Turbulence Models

Solve the turbulent fluctuations in a pretty detailed way is unnecessary. So, just the turbulence effects that disturb the mean flow are usually looked for. For make a useful CFD code in a general purpose, the turbulence model that the code is based on has to be accurate, simple and economical to run, and it must provide a wide applicability.

There are several turbulence models but here just the models which are offered by OpenFOAM will be shown.

- Reynolds-Average Simulation (RAS) or Reynolds-Averaged Navier-Stokes (RANS):

It is the most used model for engineering applications. This solves the time average Navier-Stokes equations that were shown before. All of the turbulent scales are modelled.

- Large Eddy Simulation (LES):
  The larger turbulent structures have almost all the energy so they are solved by the governing equations, whereas the smaller ones are modelled because they are more foreseeable.

- Detached Eddy Simulation (DES):
  This is a hybrid model made with RANS and LES models, in which the RANS models is mainly used but there are regions simulated by LES.

- Direct Numerical Simulations (DNS):
  It solves the Navier-Stokes equations numerically with no turbulence model. It requires a really high computational power.

As it was said the RANS models are the most common so they will be explained more detailed. Before introduce the turbulence into the system there were four unknowns *u,v,w* and *p*, however once the turbulence has been taken into account through the time-averaging operation six new unknowns have appeared. So in order to solve the equation system that is made by the Reynolds equations (Eqn 2.7, Eqn 2.8.a, Eqn 2.8.b, Eqn 2.8.c), a turbulence model is needed to close the system.

The OpenFOAM provide the user with the next RANS turbulence model for incompressible fluids, some of them can be seen in the table below (Table 2.1).

Laminar	Dummy turbulence model for laminar flow
kEpsilon	Standard k-ε model
kOmegaSST	k-ε-SST model
RNGkEpsilon	RNG k-ε model
LaunderSharmaKE	Launder-Sharma low-Re k-ε model
LRR	Launder-Reece-Rodi RSTM
LaunderGibsonRSTM	Launder-Gibson RSTM
realizableKE	Realizable k-ε model
SpalartAllmaras	Spalart-Allmaras 1-eqn mixing-length model

Table 2.1 RANS models supplied by OpenFOAM

From all the latter models there are two of them that are usually get for engineering tasks, the standard k-epsilon model and the k-epsilon RNG, that is based on the first one.

2.3.2. Boussinesq approximation

The viscous stresses and the Reynolds stresses can be supposed to be linked between them and both of them can have almost the same effect on the mean flow. So the turbulence stresses can be written as:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \qquad \textbf{Eqn 2.9}$$

According to Boussinesq the Reynolds stresses can be related with the mean rate of deformation since the turbulent stresses increase as the mean rate of deformation increases.

$$\tau_{ij} = -\rho \overline{u_i' u_j'} = \mu_t \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) \qquad \textbf{Eqn 2.10}$$

Where μ_t is the turbulent or eddy viscosity (dimensions Pa.s), it is used to model the turbulence forces as viscosities forces. Thanks to that viscosity the system will be solved if μ_t can be found. Total viscosity (μ_{Total}) is the sum of the viscosity (μ) and the turbulent viscosity (μ_t).

2.3.3. k-epsilon model

The k-epsilon model is a two-equation model, so two new terms will be included also two new transport equations which will represent the turbulence properties of the mean flow will appear.

These two new variables are the kinetic energy (k) and the dissipation of turbulent kinetic energy (ε). The latter determines the scale of the turbulence while the first one represents the energy in the turbulence.

The kinetic energy per unit mass can be written downs as:

$$k = \frac{1}{2}\left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}\right) \qquad \textbf{Eqn 2.11}$$

According to the standard k-epsilon model (Launder and Spalding, 1974) k and ε are used to define the velocity scale (v) and the length scale (*l*).

$$\vartheta = k^{1/2} \qquad \textbf{Eqn 2.12}$$

$$l = \frac{k^{3/2}}{\varepsilon} \qquad \textbf{Eqn 2.13}$$

The velocity scale and the scale length are used to define the eddy viscosity

$$\mu_t = C_\mu \rho\, l\, \vartheta = C_\mu \rho \frac{k^2}{\varepsilon} \qquad \textbf{Eqn 2.14}$$

C_μ is a constant with no dimensions. Finally the transport equations that are used by the standard k-epsilon model are:

Kinetic energy equation:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho k U_i)}{\partial x_j} = \tau_{ij}\frac{\partial U_i}{\partial x_j} - \rho\varepsilon + \frac{\partial}{\partial x_i}\left[\left(\mu + {\mu_t}/{\sigma_t}\right)\frac{\partial k}{\partial x_i}\right] \qquad \textbf{Eqn 2.15}$$

Dissipation kinetic energy:

$$\frac{\partial(\rho\varepsilon)}{\partial t} + \frac{\partial(\rho\varepsilon U_i)}{\partial x_j} = C_{\varepsilon 1}\frac{\varepsilon}{k}\tau_{ij}\frac{\partial U_i}{\partial x_j} - C_{\varepsilon 2}\rho\frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_i}\left[\left(\mu + {\mu_t}/{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_i}\right] \qquad \textbf{Eqn 2.16}$$

The constants that appear in the equations (Eqn 2.15 and 2.16) have the following values:

C_μ	$C_{\varepsilon 1}$	$C_{\varepsilon 2}$	σ_k	σ_ε
0,09	1,44	1,92	1,0	1,3

Table 2.2 coefficients used in k-epsilon transport equations

2.3.4. k-epsilon RNG model

Using Re-Normalization Group (RNG) methods the standard k-epsilon is improved to RNG k-epsilon model (Yakhot, 1986). In the standard k-epsilon model a single turbulence length scale determines the eddy viscosity, so the turbulent diffusion is calculated just at that scale, unlike the RNG approach attempts to take into account different scales.

$$\rho\frac{\partial\varepsilon}{\partial t}+\rho\frac{\partial(\varepsilon U_i)}{\partial x_j}=(C_{\varepsilon 1}-C_{\varepsilon 1RNG})\frac{\varepsilon}{k}\tau_{ij}\frac{\partial U_i}{\partial x_j}-C_{\varepsilon 2}\rho\frac{\varepsilon^2}{k}+\frac{\partial}{\partial x_i}\left[\left(\mu+{\mu_t}/{\sigma_\varepsilon}\right)\frac{\partial\varepsilon}{\partial x_i}\right] \qquad \textbf{Eqn 2.17}$$

Where:

$$C_{\varepsilon 1RNG}=\frac{\eta\left(1-{\eta}/{\eta_0}\right)}{1+\beta\eta^3} \qquad \eta_o=4{,}377 \quad \beta=0{,}0012$$

C_μ	$C_{\varepsilon 1}$	$C_{\varepsilon 2}$	σ_k	σ_ε
0,085	1,42 - $C_{\varepsilon 1RNG}$	1,68	0,7179	0,7179

Table 2.3 coefficients used in RNG k-epsilon transport equations

This model improves the accuracy in rotating flows however in some situations the standard model is still better. For instance, predicting vortex evolution.

2.4. Canopy implementation

2.4.1. Drag coefficient (C_D)

The airflow behaviour through a forest and the canopy architecture are linked with the drag coefficient, that is the reason why is an important parameter that has to be calculated in order to achieve the map file which will be used for the simulation. The problem is that there is no such a table which shows the C_D for each kind of tree or forest. There are several studies that give a way to obtain that parameter but there is no a settled equation for that because some studies has developed opposite results (see APPENDIX II.2).

Although the drag coefficient is not easy to calculate, the drag force the canopy produce in the airflow can be added to the momentum equations as an external force. The momentum absorbed by the canopy can be represented with the source term (Dalpe and Masson 2008 **[7 and 8]**):

$$S_u = -\rho C_D \alpha |u| u_i \qquad \textbf{Eqn 2.18}$$

The turbulence model used in that studied is *k* – ε model, which has to be modified in order to integrate the turbulence made by the canopy. So source terms (S_K and S_ε) will be added to the transport equations.

$$S_k = \rho C_D \alpha \left(\beta_p |u|^3 - \beta_d k |u|\right) \qquad \textbf{Eqn 2.19}$$

$$S_\varepsilon = \rho C_D \alpha \frac{\varepsilon}{k} \left(C_{\varepsilon 4} \beta_p |u|^3 - C_{\varepsilon 5} \beta_d k |u|\right) \qquad \textbf{Eqn 2.20}$$

1.4.2. Leaf area density (α).

The leaf area density means the surface of tree (leafs, branches or trunk) per cubic metre of forest (m^2/m^3). That parameter can be measured and along the simulations done just C_D usually is assumed.

3. OPENFOAM

3.1. Introduction

The OpenFOAM is a CFD software which is free and open source which was developed at the end of 1980s in London at the Imperial College; it became open source in 2004 with the General Public License GNU. It has to be pointed out that being an open-software means that, it has to be run on an open Operative System (O.S), Linux.

As it is “open” the users can modify, extend, improve or customise the existing applications of OpenFOAM; that characteristic of the software makes it a common CFD tool used along the engineer community. It can solve a wide range of fluid dynamics problems, complex fluid flows combining with turbulence, heat transfer and even chemical reactions. It also is used to analyse solid dynamics and electromagnetics systems.

The software includes pre-processing tools as for example, meshing tools to create your own mesh (e.g. BlockMesh or snappyHexMesh) or for converting a given mesh from another software (e.g. FLUENT) to OpenFOAM mesh file. Post-processing tools, as paraView, also are provided with the software.

There is a big community online which share their customized applications and their skills with new users of OpenFOAM **[14]**.

Solver Capabilities

1. Incompressible flows
2. Multiphase flows
3. Combustion
4. Buoyancy-driven flows
5. Conjugate heat transfer
6. Compressible flows
7. Particle methods (DEM, DSMC, MD)
8. Solid dynamics
9. Electromagnetics

Meshing Tools

1. Mesh generation in OpenFOAM
2. Converting meshes into OpenFOAM format
3. Tools to manipulate meshes

Library Functionality

1. Turbulence models
2. Transport/rheology models
3. Thermo physical models
4. Lagrangian particle tracking
5. Reaction kinetics / chemistry

Post-processing

1. ParaView and VTK post-processing
2. Run-time post-processing
3. Third-party post-processing
4. Core Technology
5. Numerical method
6. Linear system solvers
7. ODE system solvers
8. Parallel computing
9. Dynamic mesh

It can be said that OpenFOAM is just a library which creates executable files (applications) for solving fluid dynamics problems. Two types of applications can be made with OpenFOAM:

- Solvers, which solve the cases.
- Utilities, which are used to transform data, pre-processing and post-processing.

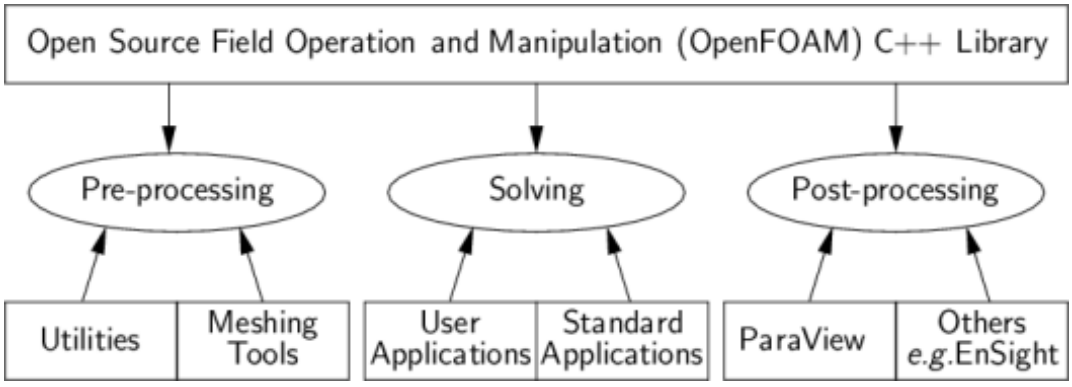


Fig 3.1 Structure of OpenFOAM [14]

The programming language used by OpenFOAM is C++. This language is one of the most common languages that nowadays are being used. It is not necessary to be an expert programmer for modifying the default codes supplied by OpenFOAM, but it is a good idea to be familiarised with it. On the World Wide Web some tutorials, which are prepared for given the acknowledgement needed for customizing an application, can be found **[20]**. Moreover a manual can be downloaded from the OpenFOAM website **[15]** (Programmer´s Guide).

3.2. Applications in OpenFOAM.

3.2.1. A new application in OpenFOAM will need the next file for being compiled.

The main file which is going to be read at first time when the application is executed is the .C file (*newApp.C* in this situation), it contains the class definition.

A class object construction, class member functions and data storage.

Every class and every operation that is performed by the main code has to exist, so a class declaration will be needed by each class, and they usually are in headers files (.H extension). The class names and its functions are declared in those files which are included at the beginning of the .C code file, and the compiler will take them into account by writing the line: #include “class.H”.

In some applications of OpenFOAM there are a file called *createFields.H* whose code creates fields and reads fields input data in order to the header files not be just a class declaration.

Fig 3.2 Application structure [14].

UNIX/Linux systems offers a compiler called Make, however the OpenFOAM provides a more versatile and easier compiler; wmake. In order to OpenFOAM recognize the application code; the whole code has to be compiled. A folder called *Make* will be needed in the class directory and this folder will contain two files: *options* and *files.*

The options file contains the full directory paths and library names where the header files are. The syntax used is:

```
EXE_INC = \
        -I<directoryPath1>\
        -I<directoryPath2>\
        …
        -I<directoryPathN>
EXE_LIBS = \
        -L<libraryPath1>\
        -L<libraryPath2>\
        …
        -L<libraryPathN>\
```

-I<library1>\

-I<library2>\

…

-I<libraryN>

The compiler will need a list of .C source files that have to be compiled. The *files* file contains a full path and name of the compiled executable. In that list, at least the name of the main .C file (*newApp.C*) has to be written down, but other sources that are made for the application and they are not in any class library. The syntax in the *files* file will be:

newApp.C

EXE = $(FOAM_USER_APPBIN)/newApp

The new applications developed by the user has to be stored in the path $FOAM_USER_APPBIN; and the standard applications that are offered by OpenFOAM are stored in $FOAM_APPBIN.

For instance in this thesis, the application" treeFoam" needs others sources files that are not in the class libraries (*kEpsilon.C* and *RASModelhig.C* which are in the *src* user´s folder) but they still have to be compiled.

Once "wmake" has been executed, a dependency list file (.dep extension) is created (*newApp.dep*). If an existing class is wanted to be recompiled after suffering some modifications, the old dependencies have to be removed by typing "wclean" at the terminal.

3.2.2. Equation representation.

One of the main advantages of using OpenFOAM is that the equations are written in the code files in a syntax which looks like the differential equations. For instance in an incompressible laminar flow the momentum equation is:

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \mu \nabla U = -\nabla p \qquad \textbf{Eqn 3.1}$$

This equation is written in C++ code as:

```
Solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::lapalcian(mu, U)
    ==
  - fvc::grad(p)
);
```

In order to develop new applications, some knowledge of the programming language C++ is necessary but knowing the equations models, methods and algorithms for the solution is also necessary and more important for making new solvers.

3.2.3. Standard applications.

STANDARD SOLVERS

OpenFOAM supplies some solvers that can be found in the $FOAM_SOLVERS directory (shortcut by typing app on the terminal). These standard solvers cover several fields as continuum mechanics (incompressible flow), combustion and solid body stress analysis; the name of the solvers usually shows what the application is used for. In the OpenFOAM user´s guide **[14]** a list of the standard solvers is shown. Before start developing a new solver is a good idea to use these standard solvers that belong to the field that the new user wants to work on.

STANDARD UTILITIES

They are placed in the $FOAM_UTILITIES directory (shortcut util on the terminal). One of the most important utilities is the blockMesh one which is used for meshing a domain; but there are several mesh conversion utilities that are rather important because allow the user to realise the mesh with other software and after that convert it to OpenFOAM format, also in the other way round. A list with the standard utilities with a short description of what they do can be found in the OpenFOAM user´s guide **[14].**

STANDARD LIBRARIES

The libraries are not any applications but they are necessary for running the solvers. In the $FOAM_LIB/$WM_OPTIONS directory (shortcut lib) they are placed. The classes and functions are defined in that kind of libraries but the models which are used in the computational continuum mechanics are also described in here. The list of those standard libraries is in the OpenFOAM user´s guide **[14].**

3.3. OpenFOAM cases

Each case in OpenFOAM requires a minimum pack of files in order to run some application; in the figure 3.3 these files are shown. There are several cases which are given by the software, they are in $FOAM_TUTORIALS directory. They are split up by the solver which is used to solve them. When the users want to create a new case, they often choose a given case which resembles the situation they want to study, and then they just modify what is needed.

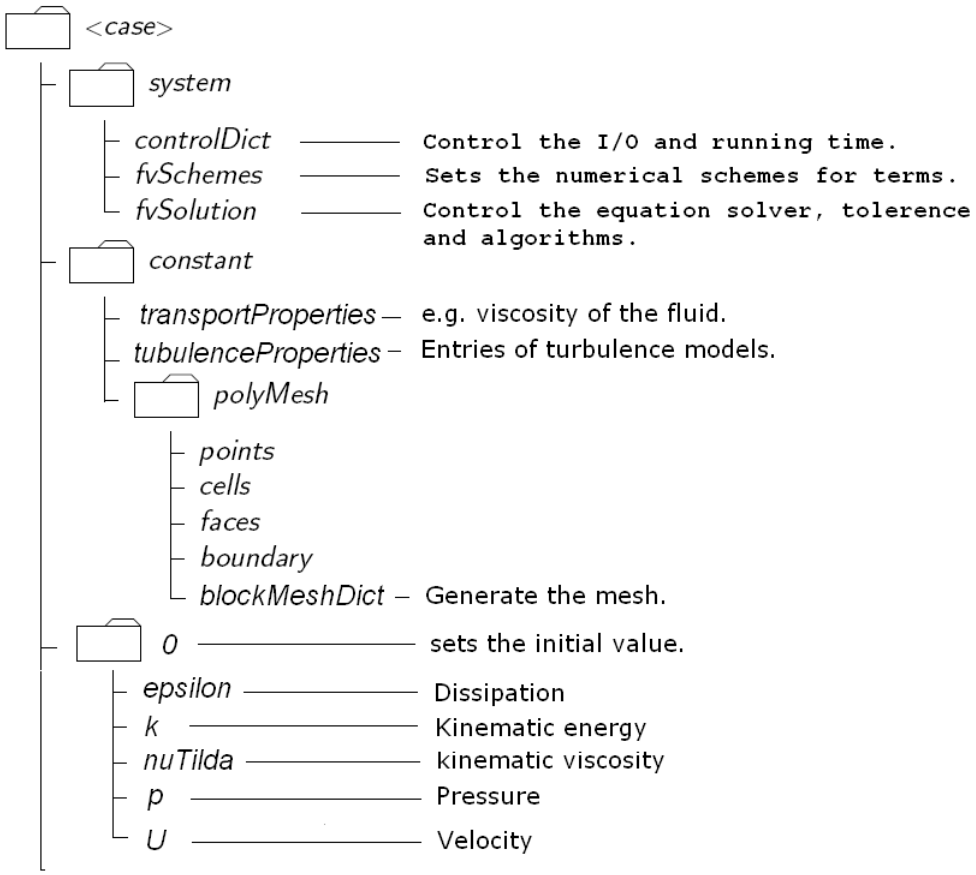


Fig 3.3 Structure of a case in OpenFOAM [14].

3.3.1. System Folder

At least the next three files are in this folder:

controlDict file, run control parameters are defined in this file. Start/end time and time step, also the output data parameters (see APPENDIX IV.1).

fvSchemes file sets the numerical schemes for terms. How the gradient, the divergence, the laplacian, the interpolations of values and so on are solved are defined in that file (see APPENDIX IV.2).

fvSoluton file controls the equation solvers, tolerances and algorithms (see APPENDIX IV.3).

3.3.2. Constant Folder

In that folder the physical properties of the case are specified by two files:

***transportProperties*:** It defines the properties of the fluid which is being used, for instance the viscosity of the fluid (see APPENDIX IV.4).

***turbulenceProperties*:** The turbulence model that will be used in the case is set in this file (see APPENDIX IV.5).

The mesh is also defined in this folder, inside the constant folder there is a subdirectory called ***polyMesh*.** In the latter directory the files which define the mesh can be found:

points: this file contains a list of every point that are needed for define the studied figure. Each point defines its location in 3-D space by a vector of 3 components. Two or more different points cannot be defined in the same position (see APPENDIX IV.6).

cells: this file is used to define the mesh by hand.

faces: Faces are made by a list of points which are called by its label (the point coordinates are not used in here) , and the points are ordered so an edge connects each two neighbouring points. The faces are list and each one has its own label in order to referred, the label represents its position on the list (see APPENDIX IV.7).

Two types of faces can be defined:

- Internal faces.

  Faces which connect two cells and just two.

- Boundary faces.

  These faces belong to one cell because they are situated in the boundary of the domain.

boundary: The boundary is made by a list of patches which ones has associated a boundary condition. A patch is formed by a list of boundary faces which are called by theirs labels; no internal face can be part of a patch (see APPENDIX IV.8).

owner: In that file every face is associated to a cell, it represents the cell which every face belongs to (see APPENDIX IV.9).

neighbour: This file is almost the same as the owner file, however in the *neighbour* file no boundary faces are referred.

blockMeshDict: If the mesh is defined using the "blockMesh" utility this file will set the procedure that the tool makes the mesh.

The mesh can be defined by hand what is a really hard task, using the "blockMesh" utility or the "snappyHexMesh" one provided with the OpenFOAM software or importing it from another meshing tool as "GAMBIT" for instance (see APPENDIX V).

3.3.3. 0 Folder

The variables that are going to be used in the case has to be defined in a file inside the *0* folder, the velocity, pressure, kinematic energy or the new ones that are going to be needed in order to include the canopy effect as the drag coefficient (C_D) and the leaf density (α). In those files the initial value for the variables are settled (an example is shown in the APPENDIX III.6). The velocity profile at the initial time in the simulation will be define in the *U* file so will do the pressure in the *p* file (see APPENDIX VI.1 and VI.2); in this situation the C_D and α values will be written down in the C_D and *alphaL* files after the pre-processing tool "leafIndexToFoam" was run (see APPENDIX VI.3).

4. Process and Results

4.1. Test Case The Bolund hill.

The Bolund case is real situation that has been studied several times by experimental processes and CFD programs. It is used for validating flow models in complex terrain. Each day there are more wind farms that have to be installed over complex terrain, where the turbulence and shear effects are really important. For this reason the CFD tools are becoming more relevant when the optimum positions for settled wind turbines has to be determined. The models that the CFD programs use usually are just test in wind tunnels and over simple terrain.

Fig 4.1 Bolund case landscape.

Fig 4.2 Bolund hill.

The Bolund mesh will be used for testing the applications made. Bolund is a 130 m long, 12 m high and 75 m wide hill situated at the coast, at the north of Risø DTU (Denmark).

As that hill has a sharp change in surface roughness from water to grass and it is well-exposed vertical escarpment is a good 3-D challenge for any flow solver. However, as it can be seen in the pictures (Fig 4.1 and Fig 4.2) there are no trees on the hill, so the canopy implementation will be tested through the Bolund hill by simulating a rectangular bunch of trees over the hill. These trees will 10 or 15 meters high **[8]** (although there are some tree species as the Norway or European Spruce which is between 30 to 50 meters high).

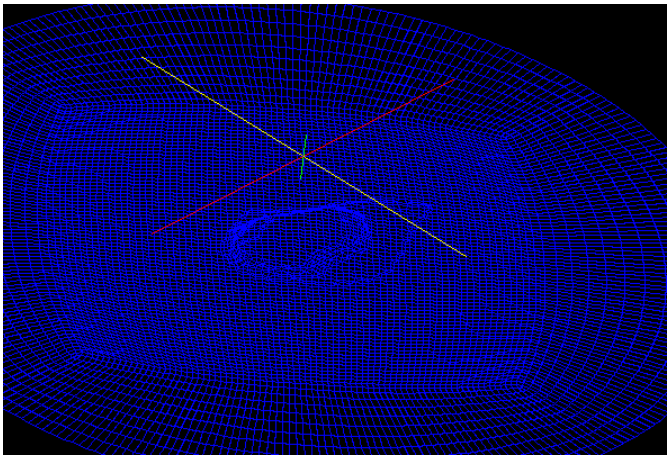


Fig 4.3 Bolund hill ´s mesh.

4.2. Canopy implementation.

As it was said before, the effect of the canopy over the airflow will be included through the drag coefficient (C_D) and the leaf area density (α). Once these parameters are known for a determined canopy, both of them modify the Navier-Stokes equations that are used for solving the airflow distribution. The momentum equation will be added a source term (S_u).

The turbulence model k-epsilon also will be changed by these parameters, source terms S_k and S_ε (Eqn 2.19 and 2.20) will appear into the transport equations, both of them, the kinetic energy equation and the dissipation kinetic energy equation (Eqn 2.15 and 2.16 respectively).

Moreover, a pre-processing tool will be developed in order to convert the given canopy data (map file) into OpenFOAM files (Cd and alphaL files in the "0" folder).

4.2.1. Adding Canopy as a variable in OpenFOAM

In order to the solver takes into account the C_D and α variables, first of all some libraries which it will need must be modified. The standard libraries are situated in the "src" folder inside the OpenFOAM directory. So as to avoid mess up the default files of OpenFOAM, a new "src" folder will be created in ($WM_PROJECT_USER_DIR) the user directory, and there the new libraries will be defined, they have to be compiled as well but instead of type "wmake", the libraries are compiled using "wmake libso" **[14]**.

The files *kEpsilon.C* and *kEpsilon.H* which are needed to solve the k-ε model in OpenFOAM will be modified in order to add S_k and S_ε. C_D and α will be defined in those files, also every constant which the source terms need. The whole new code can be seen in the enclosed DVD, but it has to be pointed out that the main changes are in the kinetic energy equation and in the dissipation equation:

```
// Dissipation equation
tmp<fvScalarMatrix> epsEqn
(
  fvm::ddt(epsilon_)
+ fvm::div(phi_, epsilon_)
- fvm::Sp(fvc::div(phi_), epsilon_)
- fvm::laplacian(DepsilonEff(), epsilon_)
==
C1_*G*epsilon_/k_
- fvm::Sp(C2_*epsilon_/k_, epsilon_)
+          fvm::Sp(Cd_*alphaL_/k_*(CEps4_*betap_*pow(mag(U_),3)-
CEps5_*betad_*k_*mag(U_)), epsilon_); //Source Term)
```

```
// Turbulent kinetic energy equation
tmp<fvScalarMatrix> kEqn
(
  fvm::ddt(k_)
+ fvm::div(phi_, k_)
- fvm::Sp(fvc::div(phi_), k_)
- fvm::laplacian(DkEff(), k_)
==
G
- fvm::Sp(epsilon_/k_, k_)
+                  fvm::Sp(Cd_*alphaL_/k_*(betap_*pow(mag(U_),3)-
betad_*k_*mag(U_)), k_); //Source term)
```

The new solver called “treeFoam” will call the new libraries, and it will be created in the user directory ($FOAM_USER_APPBIN). The solver will include the new source term in the momentum equation, which is defined in the *UEqn.H* file.

```
// Momentum equation
tmp<fvVectorMatrix> UEqn
(
fvm::div(phi, U)
+ turbulence->divDevReff(U)
+ 2*(Omega ^ U)  //coriolis force 2*(omega x U)
+ gravity        //gravity force

+ fvm::Sp(Cdalpha*mag(U)*U,U) //canopy drag force
==
sources(U)

);
```

The problem of create a new library for the new application is that these libraries are linked with a huge amount of others files and libraries, so a really good skill programmer will be needed. Finally after trying hard to create new libraries with no success, it was chosen to unlock the OpenFOAM files and modified directly the *kEpsilon.C* file and *kEpsilon.H* in order to add the source terms and its constants.

4.3. Pre-processing application

4.3.1. Map files

The canopy data will be provided through a ".map" extension file where the drag coefficient (C_D), the leaf area density (α), and the height of the canopy (z_o and Δz) are defined. As the canopy is set for some areas of the map, each area has to be defined as well. The projected area of the canopy over the X-Y plane will be set in the file as a contour line or a polygon.

The first four lines of the map file are skipped by the code because just the name of the file, the dimension for C_D and α, are in those lines, so there are not relevant for convert the map file into OpenFOAM file. After these lines the definition of the canopy starts. They are defined in pools and each pool is formed by (see APPENDIX III.5):

- The C_D, α, Δz, z_o and the number of vertex of the polygon which form the area of the canopy on the X-Y plane, those are written in the first line of each polygon.
- After the header of the pool, the 2-D coordinates of every vertex is declared below, they are ordered clock wisely.

With the 2-D coordinates of the polygons and the values of z_o and Δz, a volume which contains C_D and α values, is define.

In this situation simple rectangular prism shape bunch of trees were defined, which projection is just a rectangle.

4.3.2. Canopy Application.

This application will convert the map file into OpenFOAM files (*Cd.H* and *alphaL.H*). In order to achieve that goal, the implemented code will check every cell that makes the mesh, and if the centre point of the cell is inside any volume, the values of C_D and α will be copied as internal fields of this cell. When the whole mesh is checked, Cd and α values will be written on the *Cd.H* and *alphaL.H* files which are inside the "0" folder (see Fig 3.3).

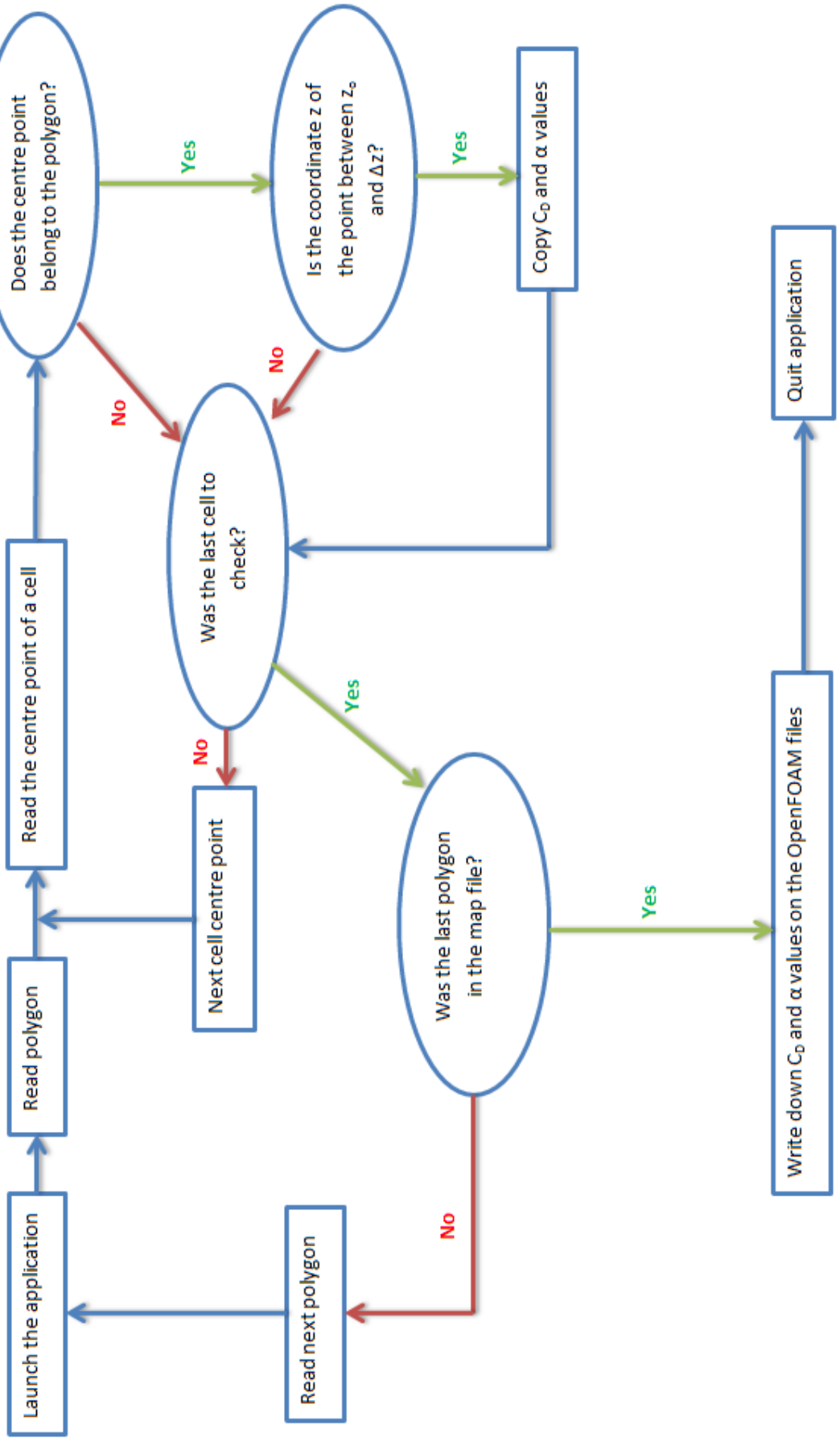


Fig 4.4 Process followed for reading a map file.

In order to carry out the program shown at the figure above (Fig 4.4) the next C++ code has to be developed:

- **leafIndexToFoam :** This file is the executable file which calls the other files needed for doing the process.
- **createCdAlphaL.H :** In order to OpenFOAM knows that C_D and α variables are being changed, these variables has to be created in the "0" folder; so that they can be read later.
- **Check_inside_cell.H :** This piece of code will check if each centre point of the cells belongs to the volume defined by the polygon and the height of the canopy.
- **Readmapfile_CdAlphaL.H :** It opens the map file, it reads the polygon and the parameters (C_D,α,z_o,Δz), and summon the Check_inside.H code.

The full code is shown in the APPENDIX III.

There are several ways that can be used for checking if a point is inside a polygon. Some of them can be seen at the reference [19], on the internet. However two ways will be explain just below:

1. For the first method the angle between the lines which link two consecutive point of the polygon with the point, has to be computed (θ_i). Then if the sum of all angles is equal to 360º means that the point is inside the polygon ($\sum\theta_i = 360º$).

   In order to get the angle, the cross product will be used:

$$v_i x v_{i+1} = |v_i| \cdot |v_{i+1}| \cdot sen\,(\theta_i).\vec{n} \qquad \textbf{Eqn 4.1}$$

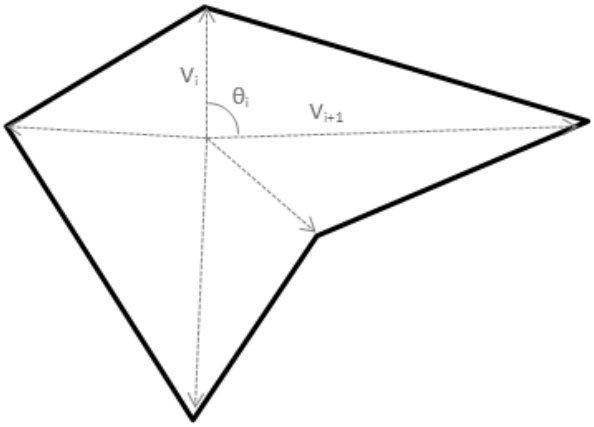


Fig 4.5 Point in polygon method 1

This method can be used even in 3-D systems, so it is a good reason for choosing it whereas the point has to be checked in a volume. The problem of the first method is that sometimes can lead us to an error if the point is placed in concave zones, because the sum of the angles could be 360º but the point will be outside as it can be seen at the figure Fig 4.6 .

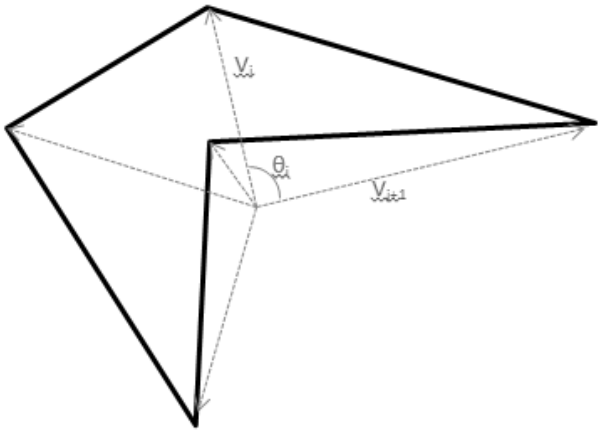


Fig 4.6 Point in polygon method 1 error

2. The second method consists on compare the X-Y coordinates of every couple of consecutive vertexes which form the polygon, with the centre point X-Y coordinates.

There are two conditions; the point has to satisfy either of them for being considered inside, otherwise will be outside.

Conditions A:

$$(point_x < vertex1_x)\ AND\ (point_x > vertex2_x)$$

$$(point_y < vertex1_y)\ AND\ (point_y > vertex2_y)$$

Conditions B:

$$(point_x > vertex1_x)\ AND\ (point_x < vertex2_x)$$

$$(point_y > vertex1_y)\ AND\ (point_y < vertex2_y)$$

This method it can be used just for checking if the point is inside the polygon in 2-D, so at first sight is useless for searching point in a volume. However as the map file supplies the height of the canopy, once the point belongs to the polygon in the X-Y plane, the Z component will be check if it is between z_o and (z_o + Δz); If it belongs, it will mean the point is in the volume.

Finally the second option will be chosen and the checking in the Z axis will be added. This code can be seen in the APPENDIX III.4 (*Check_inside_cell.H*).

4.3.3. Simulation of canopies system.

A ".map" extension file will be developed in order to simulate if the application leafIndexToFoam works properly. In the map file the bunch of trees are defined which data are shown in the table below (Table 4.1).

	Black Spruce	**Jack Pine**	**Aspen forests**
Δz(m)	10	15	10
LAI	9,19	2,22	3,57
C_D	0,15	0,45	0,4
α	0,919	0,148	0,357

Table 4.1 Characteristics of the black spruce, jack pine and aspen forest. [8]

$$LAI = \int_0^h \alpha \, . \, dz \qquad \textbf{Eqn 4. 2}$$

Where LAI means Leaf Area Index and α is the leaf area density (m^2/m^3).

The height at which the canopy starts (z_o) will be assumed 3 m for every kind of tree. As the Bolund hill is 12 m high, z_o = 15 m. For doing the simulation, two canopies systems will be defined; the first one will be 1600 m^2 of black spruce trees and the second one will be a bunch of Jack pines which will cover 400 m^2 (See APPENDIX III. Map file).

The files *Cd.H* and *alphaL.H* which are in the "0" folder will be changed by the application. These files have to be reinitialized for re-executing the program (See APPENDIX III.6)

At the following pictures it can be observed α value that the application leafIndexToFoam assigned to each cell of the mesh. The cells that are in red (α = 0,919) represents black spruce species and the cells in green (α = 0,148) belongs to Jack pine trees.

As the code checks just the centre point of the cell, if it is inside the whole cell is given the α value but if not the cell α values remains null, the conversion can miss some cells because it does not reach the cell centre an also some cells can be declared as canopy, when just half of the cell belong to the canopy volume. The magnitude of this error will be lower as the mesh is finer and as the canopy system studied is bigger.

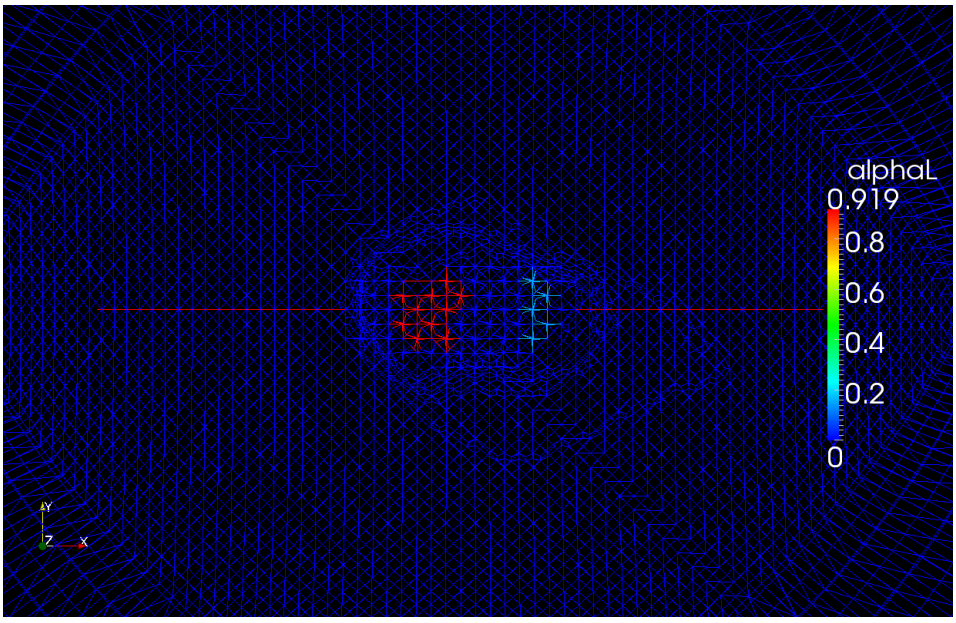


Fig 4.7 leafIndexToFoam sight normal to Z axis.

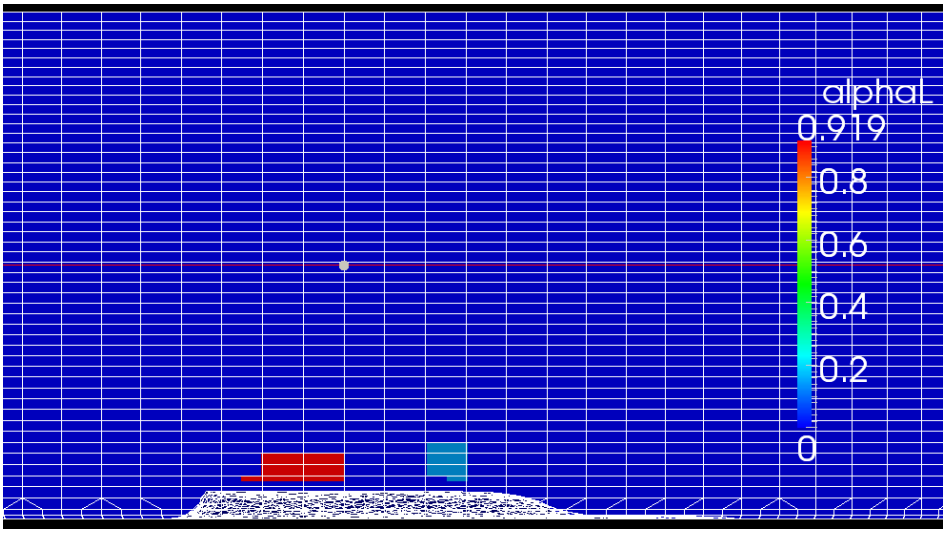


Fig 4.8 leafIndexToFoam sight normal to Y axis.

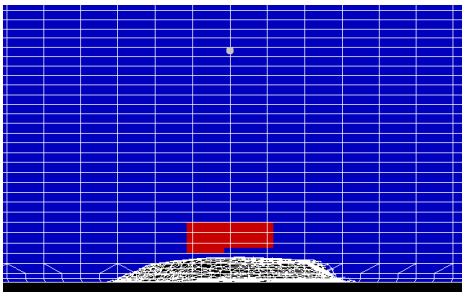


Fig 4.9 leafIndexToFoam sight normal to X axis at -30 m (Black Spruce).

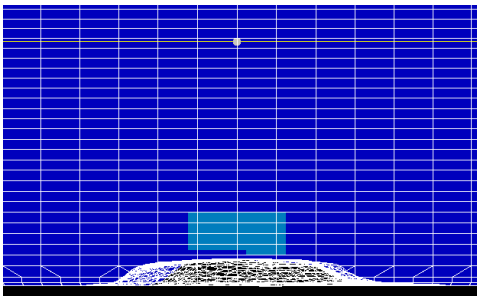


Fig 4.10 leafIndexToFoam sight normal to X axis at 45 m (Jack Pine).

4.4. TreeFoam solver test.

Once the libraries are compiled after adding the effect of the canopies systems in the airflow, and the pre-processing tool can get properly the canopy from the map files, the last step is check if the new solver “treeFoam” runs in the expected way.

With new libraries made the solver could not reach the solution because no convergence was got after 1000 runs. In order to detect if the problem was in the solver or in the libraries, a new solver was coded but this one uses the standard libraries of OpenFOAM. This new solver was

checked with a null canopy system against the “simpleFoam” solver, as the result was the same, it was notice that the problem of the “treeFoam” solver was in the turbulence libraries made. As it was said before, finally the standard turbulence libraries will be used but the *kEpsilon.C* and the *kEpsilon.H* files will be improved for adding the source terms due to the canopy.

In order to carry out this task, the Bolund case will be studied four times:

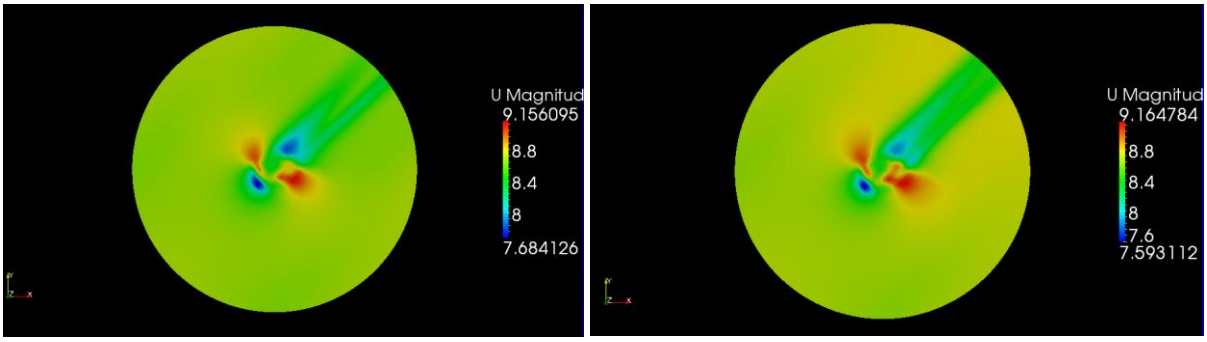
- **First simulation:** Simulate the Bolund case, using “simpleFoam”.
- **Second simulation:** Run the “treeFoam” solver with no canopy over the hill.
- **Third simulation:** The canopy will be added on the hill for this simulation. The α value will be 0,919 (m^2/m^3).
- **Fourth simulation:** Run “treeFoam” after loading the canopy system on the hill. The α value will be 1,838 (m^2/m^3).

The pictures shown on the report will be three of them looking from upside, at three different distant (20, 10 and 40 meters) which show the velocity profiles; and then another picture will show the wind speed along the wind direction (1, 1, 0) by making a slice of the domain.

4.4.1. No canopy system define in the domain.

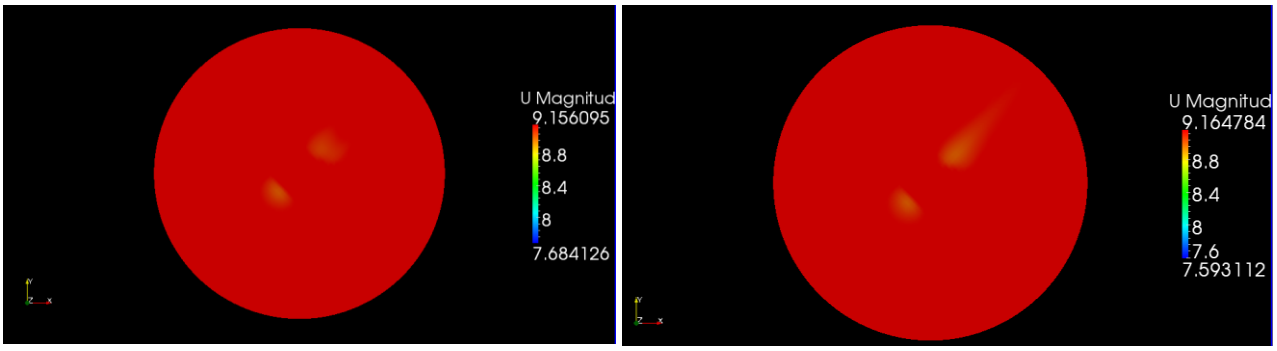
As the “treeFoam” solver is developed from the “simpleFoam” default solver, the reference case with no canopy systems will be made with that solver. The simulations will be run along 1000 times and the programme will write the data (velocity, pressure, drag coefficient and leaf area density) every 50 runs but when it converges the simulation can be stopped because it is no necessary to go on running it. After fifty runs the convergence of the solution reached is good enough.

The solution given by the new solver with no canopy has to be the same as the “simpleFoam” one. So the next pictures compare the velocity profile of the two solvers.



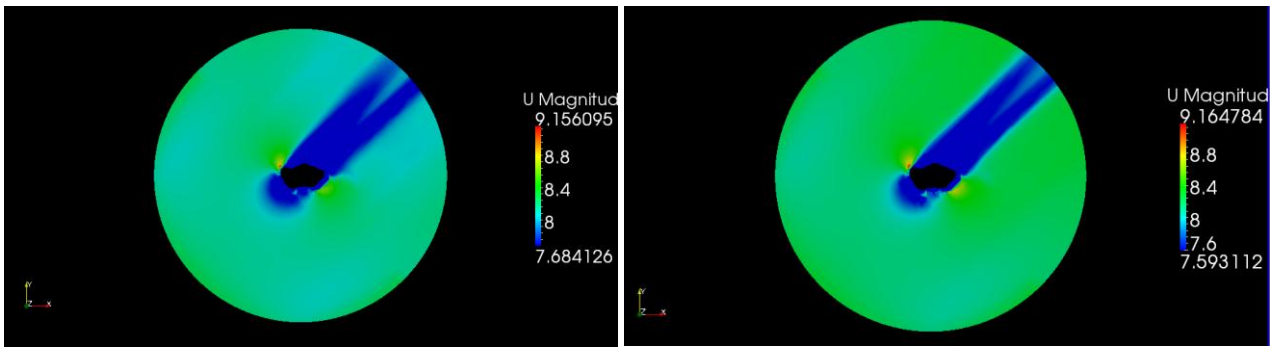
"simpleFoam" solution "treeFoam" solution

Fig 4.11 Velocity (m/s) profile at 20 m.



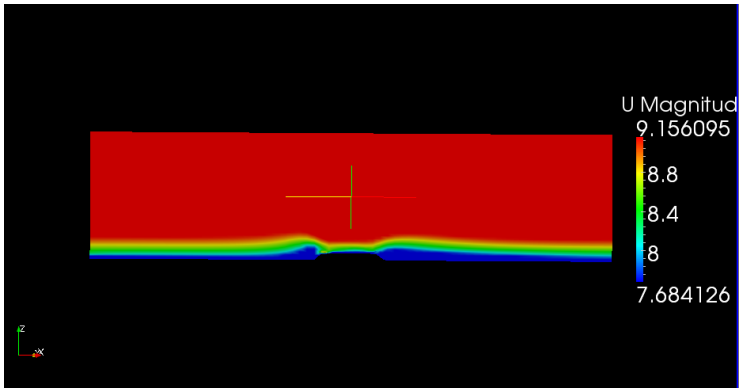
"simpleFoam" solution "treeFoam" solution

Fig 4.12 Velocity (m/s) profile at 40 m.

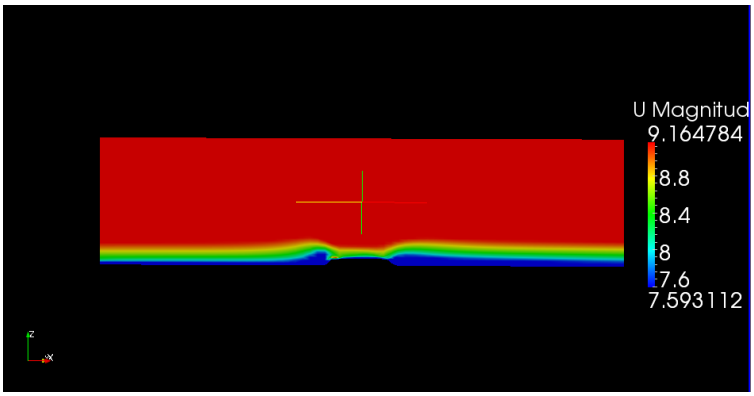


"simpleFoam" solution "treeFoam" solution

Fig 4.13 Velocity (m/s) profile at 10 m.



"simpleFoam" solution



"treeFoam" solution

Fig 4.14 Velocity (m/s) profile along the wind direction.

As it can be seen in the pictures (Fig 4.11, 4.12, 4.13 y4.14), both solvers give almost the same velocity profile, the small difference is due to the difference in the convergence error reached by the two solvers. After that the new solvers is known to works properly, the next step will be to check if the canopy is taken into account by the "treeFoam" solver.

4.4.2. TreeFoam simulation adding canopy systems.

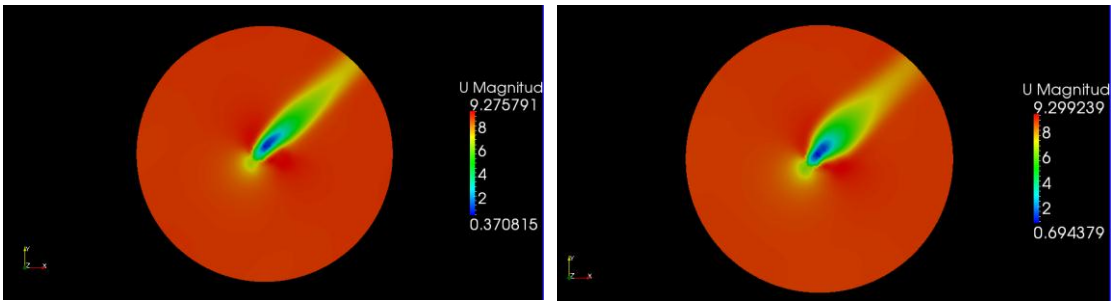
In this situation the new application "treeFoam" will be run, as just the effect of the canopy is going to be checked, the other source terms included in the momentum equation as the gravity force and the corilois force will be kick out in order to appreciate just what we are interested in. The map file used in this simulation is *tree.map* inside the "bolund225degCdAlphaL" folder where the (See APPENDIX III. Map file).

Canopy system	
Δz(m)	50
z_o(m)	15
C_D	0,15
α(m^2/m^3)	0,919 – 1,838
Area(m^2)	1600

Table 4.2 Characteristics of canopy used. [8]

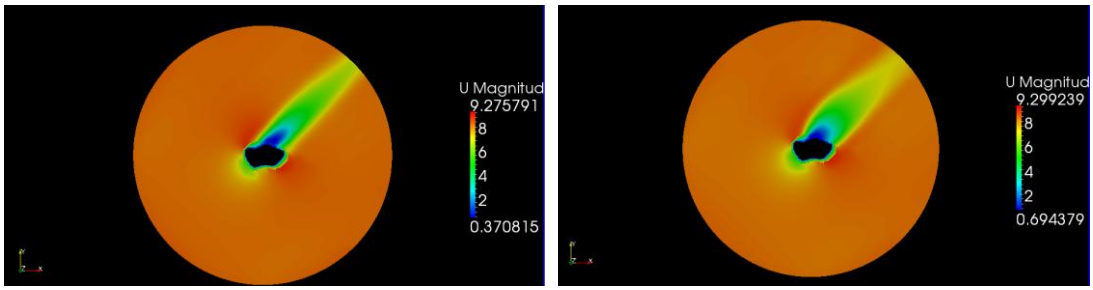
Then "treeFoam" will simulate the air flow over the Bolund case after running the "leafIndexToFoam" application.

In this situation the canopy system will be a bunch of black spruce but in order to appreciate easily the effect of them, they will be 50 m tall. The second one the α value will be doubled and both solutions will be compared.



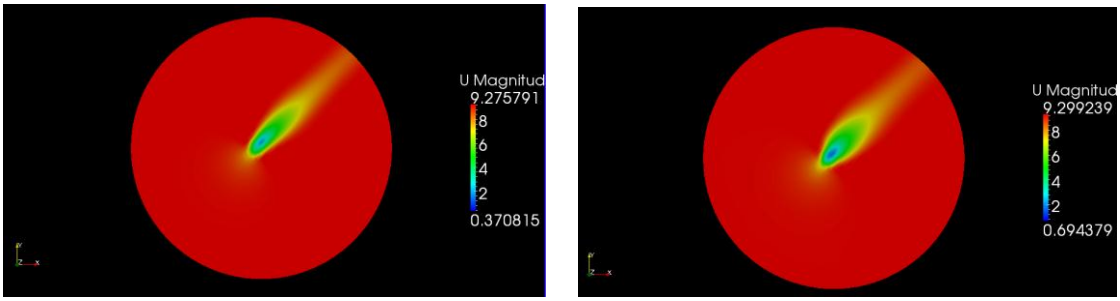
α =0,919 α =1,838

Fig 4.15 Velocity (m/s) profile at 20 m with canopy system.



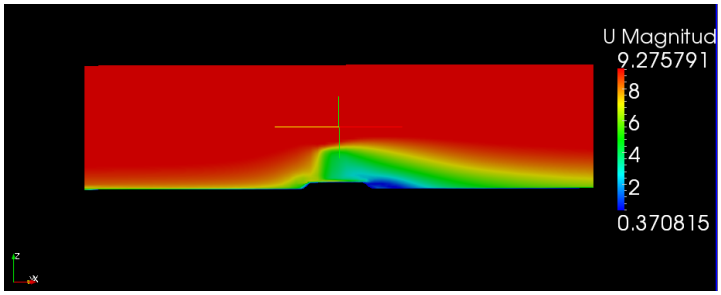
α =0,919 α =1,838

Fig 4.16 Velocity (m/s) profile at 10 m with canopy system.

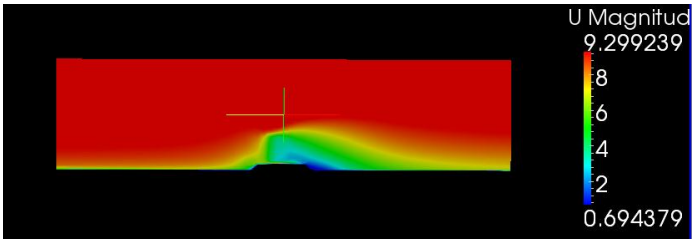


α =0,919 α =1,838

Fig 4.17 Velocity (m/s) profile at 40 m with canopy system.



α =0,919



α =1,838

Fig 4.18 Velocity (m/s) profile along the wind direction with canopy system.

First of all, it can be appreciated a huge difference between the “simpleFoam” solution or the “treeFoam” with no canopy one, against any solution given by “treeFoam” when there is a canopy system on the hill. So the source terms that the canopy system produces works in the “treeFoam” solver.

Thanks to the last two simulations, it can be concluded that the higher α value disturbs the airflow harder near to the canopy system by comparing the latter pictures (Fig 4.15, 4.16, 4.17,

4.18). So the “treeFoam” solver is checked and works properly. However, in order to confirm that a CFD solver runs 100 per cent correctly, the computer simulations must be compared with experimental data.

5. CONCLUSION

To start with, the main problem of that work is that some knowledge was needed in order to carry it out, more than expected at the first moment. It is necessary to have basic handling with C++ programme language, but is more important to know something about how OpenFOAM works because even with the tutorials that can be found on the website, it is a hard task.

However, as OpenFOAM is free software which everybody can use, it is perfect CFD software for developing new solvers. The code use by the software is open source code, so the users can change it in order to add some improvements or adapt for different necessities.

Two programmes have been developed along that thesis. The first one “leafIndexToFoam” could be tested easily and as it was seen in the pictures (Fig 4.7, 4.8, 4.9 and 4.10) the application runs properly. The α and C_D values are written successfully on the *alphaL.H* and *Cd.H* files, and then they can be observed with the viewer tool. Also the viewer shows that the canopy systems follows the map files defined previously, in a proper way.

The second application, “treeFoam” solver, was more difficult to develop because, although the results obtained looks really good because it can be seen how the airflow is affected by the canopy systems, an experimental studied should be realised in order to check if the airflow behaves properly or it has to be checked with another CFD software; it is necessary to check if the behaviour of the airflow through the canopy systems is realistic. However the solver carries his goal out, which is to add the effect of the canopy and show how the velocity profile of wind is affected. The pictures (Fig 4.15, 4.16, 4.17 and 4.18) show that effect, the wind speed falls down harder if the α value is increased. The solver was run with really high values of α but the solver could not get a convergence solution, so in order to run the solver the values of the canopy system have to be realistic ones, which is a really difficult task with the drag coefficient because is not clearly define for the different species of trees so it usually is supposed, that the reason why in this work just the α value was varies.

6. FUTURE WORK

The main task that should be developed is to automate the process for running a simulation. This improvement will allow the user to run several simulations with different airflow profiles, changing the wind speed and the direction. The python library which can be easily install in OpenFOAM allows to control the OpenFOAM runs and change the OpenFOAM data, so it will be a good start to know how this works and how it can be modified in order to carry out a new application that automate the simulation process.

It is suggested also to add the effect caused by other wind mill, because in the wind farm usually several wind mill are settled, and each one disturb the airflow the it goes through them.

Finally it should be taken into account to change from the steady-state to the transient model.

7. REFERENCES.

[1] International Energy Agency (IEA)

[2] H.K. Versteeg and W. Malalasekera. *An introduction to Computational Fluid Dynamics. The Finite Volume Method.* Longman Group Ltd, 1995.

[3] Lars Davidson. *An introduction to turbulence Models Göteborg*, Sweden: Chalmers University of Technology, 2011.

[4] D. Poggi, A. Porporato, C. Ridolfi, J.D. Albertson and G.G. Katul. *The effect of vegetation density on canopy sub-layer turbulence.* Kluwer Academic Publishers, 2004.

[5] B. Marcolla, A.Pitacco and A.Cescatti. *Canopy architecture and turbulence structure in a coniferous forest.* Kluwer Academic Publishers, 2003.

[6] A.Cescatti and B.Marcolla. "Drag coefficient and turbulence intensity in conifer canopies." *Agricultural and Forest Meteorology*, vol. 121, pp. 197-206, 2004.

[7] B.Dalpé and C.Masson. "Numerical simulation of wind flow near a forest edge." *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 97, pp. 228-241, 2009.

[8] B.Dalpé and C.Masson. "Numerical Study of Fully Developed Turbulent Flow Within and Above a Dense Forest." *Wind Energy,* vol.11, pp. 503-515, 2008.

[9] Xabier Pedruelo. *Modeling of Wind Flow over Complex Terrain Using OpenFOAM.* Vattenfal Research and Development AB, 2009.

[10] Eric Furbo. *Evaluation of RANS turbulence models for flow problems with significant impact of boundary layers.* Uppsala University, 2010.

[11] Benjamin Martinez. Wind resource in complex terrain with OpenFOAM. Risø DTU, National Laboratory for Sustainable Energy, 2011.

[12] Håkan Nilsson. *Basic of C++ and object orientation in OpenFOAM.* Chalmers University.

[13] Jan Skansholm *C++ From the Beguinning*. Addison Wesley Longman, 1977.

[14] *OpenFOAM User´s Guide*. OpenFOAM Foundation, 2011.

[15] *OpenFOAM Programmer´s Guide*. OpenFOAM Foundation, 2011

[16] Wikipedia. “Nuclear power in Sweden”. Internet: http://en.wikipedia.org/wiki/Nuclear_power_in_Sweden

[17] Godfrey Boyle. *Renewable Energy: Power for a Sustainable Future.*Oxford.

[18] Wikipedia. “Wind rose”. Internet: http://en.wikipedia.org/wiki/Wind_rose

[19] Paul Bourke. “Determining if a point lies on the interior of a polygon”. Internet: http://local.wasp.uwa.edu.au/~pbourke/geometry/insidepoly/

[20] *OpenFOAM programming tutorial.* Politecnico de Milano, Chalmers.

[21] Wind Energy THE FACTS. “The Annual Variability of Wind Speed”. Internet: http://www.wind-energy-the-facts.org/en/part-i-technology/chapter-2-wind-resource-estimation/local-wind-resource-assessment-and-energy-analysis/the-annual-variability-of-wind-speed.html

[22] WIND POWER PROGRAM. “Wind Turbine power output variation with steady wind speed.” Internet: http://www.wind-power-program.com/turbine_characteristics.htm