University of Twente

Internship report

Automated mesh generation for the rotator section of a radial inflow turbine

Author:
Tjarke van Jindelt

Supervisors:
Dr. Peter Jacobs
Dr. Carlos André de Miranda Ventura

UNIVERSITY OF TWENTE.

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Contents

List of Figures

Chapter 1

Introduction

World wide energy consumption is increasing, which is a challenge that is facing many societies. Paired with ecological threats such as climate change and smog the need for more renewable energy is apparent.
A contribution can be made with geothermal energy. Australia has enough geothermal energy to provide the countries electrical energy needs for 450 years according to an estimate by the Centre for International Economics[1]. Though many technical challenges still have to be solved to be able to extract this energy.
Another provider could be solar energy. Which can be obtained on a large scale in Australia's Deserts, as well as on a smaller scale locally.
Both solar and geothermal energy can be converted to electrical energy using turbines driven with fluids heated by either source of energy. Here the optimal shape depends on different parameters such as type of fluid, temperature of fluid, flow rate, etc. These parameters can change per site and application. Therefore an automated approach to design and optimize turbines is needed.
This internship builds on work done by C.A. de Miranda Ventura on radial inflow turbines for renewable power generation applications [2]. A sketch of a radial inflow turbine is given in figure 1.1.

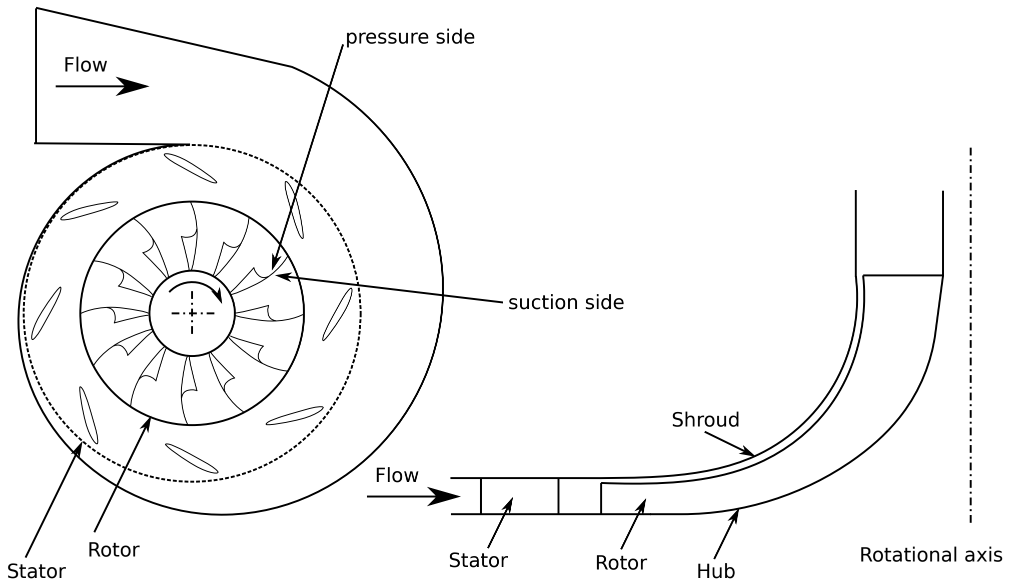

FIGURE 1.1: Sketches of a radial inflow turbine from different perspectives (after [2])

1.1 Objective and scope of the internship

The objective of this internship is to create an automated grid generation tool for one blade section of the rotator of a radial inflow turbine. The input provided are curves describing the blade at various hub to shroud locations.
This can then be used in a larger project to create a automated process of examining, designing and optimizing radial inflow turbines for given flow parameters.

Chapter 2

Background on meshes

2.1 Introduction

In Computational Fluid Dynamics (CFD) the flow is generally simulated with a finite volume approach. Which means the area one is interested in is separated into many small volumes. For each of these smaller volumes the flow quantities are determined through fluxes over their surfaces.
This conglomerate of volumes is called a mesh and the size and shape of its volumes determine whether or not a flow solver is able to accurately compute the flow one is interested in.
While smaller control volumes may be able to resolve even smallest scales of turbulence they also result in a larger amount of volumes per mesh. The amount of volumes determines the time that is needed to compute the flow. This time can mount up to unreasonable durations quite easily when fine meshes for practical applications are considered, especially when just a quick estimate is needed.
In this chapter a short explanation of meshing approaches is given and after that the mesh topology that will be used for the radial inflow turbine is explained.

2.2 Structured and unstructured meshes

There is a distinction between structured and unstructured grids of which examples are shown in figure 2.1.
The unstructured grid offers more flexibility in its creation, as the nodes (corners of a volume element) can be placed more freely. Therefore, unstructured grids have less problems when dealing with complex geometries and usually contain less nodes.

In contrast the clear order of blocks in the structured grid prescribes certain node distributions. While the structure normally increases the number of grid points, it also leaves some opportunities to save memory during calculation. This then allows different solving methods and can cut solving time.
Besides, there are flow features that are resolved better when structured grids are used e.g. boundary layers and vortices.
Eilmer3 is a flow solver that uses structured grids and therefore the grid that is created will be structured, where hexahedral elements are used.

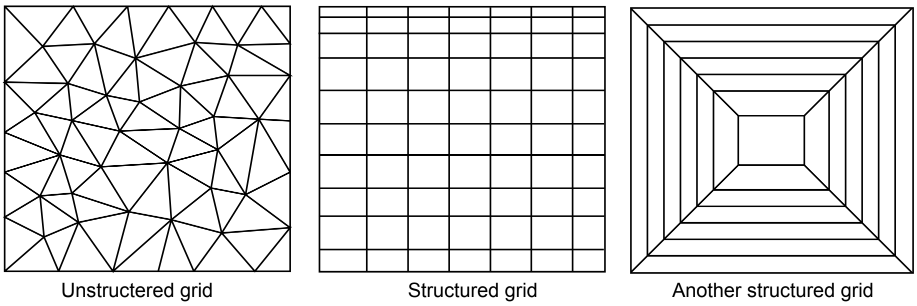

FIGURE 2.1: Examples of unstructured and structured grids

2.2.1 Topologies

There are different ways to structure a grid. The overall order is called the topology and three examples are given in 2.2. Here, the mesh is created around the black object in the center of each grid. The nomenclature of these topologies is straight forward, as they resemble letters of the alphabet.
In a grid highly skewed elements should be avoided. Therefore, round objects can be meshed best with an O-grid type approach, while square corners can be discretised with an H-grid approach.

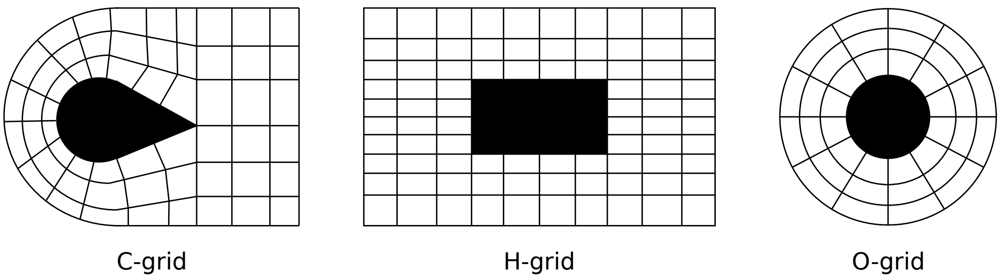

FIGURE 2.2: Example of a C-, a H- and a O-grid topology after [3]

2.2.2 Mapping

As said before, the structure of the mesh allows for different solving methods. This includes mapping from physical to computational coordinates. An example is shown in figure 2.3, where i and j are the computational coordinates.
Ordering mesh elements according to their computational coordinates can be done straight forward. This way the interfaces between elements can be identified more easily and reduce computational effort.
Mapping is used in the next chapter to create lofted surfaces, too.

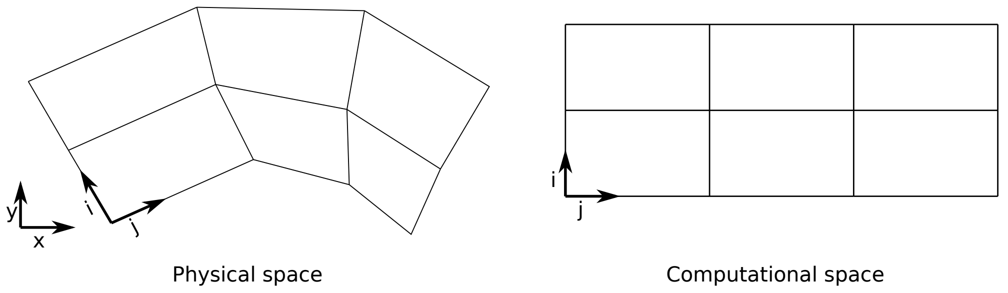

FIGURE 2.3: Example of mapping between the physical and computational space

2.3 Radial inflow turbine mesh topology

Generally it would be possible to mesh the whole turbine. However, for most calculations one can make use of the symmetry and only create a computational domain around one blade, as seen in figure 2.4, which shortens calculation time.
In this case periodic boundaries have to be used on the pressure and suction side, which means the outflow at a certain location on one side equals the inflow at the corresponding location on the other side.
It is chosen to use an O-H grid topology for the rotator section. This can be seen in figure 2.5, where the grid around the turbine is an O-grid and the grid around the O-grid is a H-grid topology. The topology will be extruded in the hub to shroud direction to form a 3-D grid. The blocks depicted here will be divided into smaller volumes, which make up the mesh.

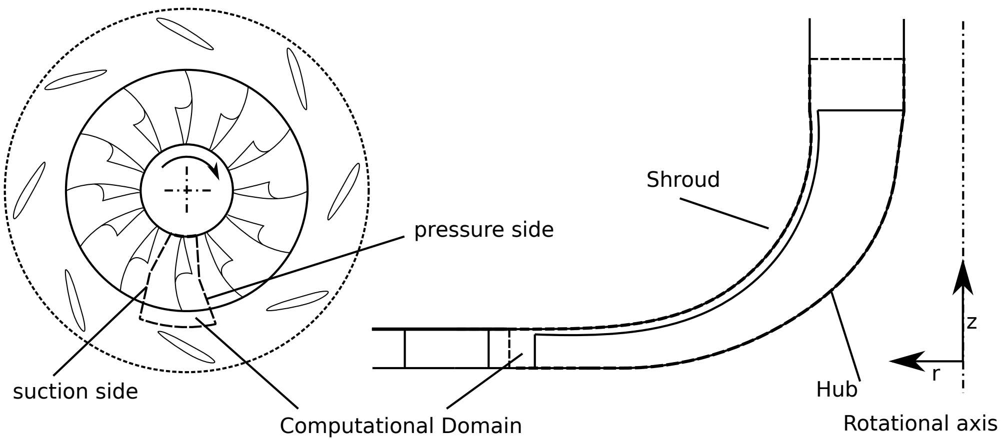

Figure 2.4: Sketch of the computational domain

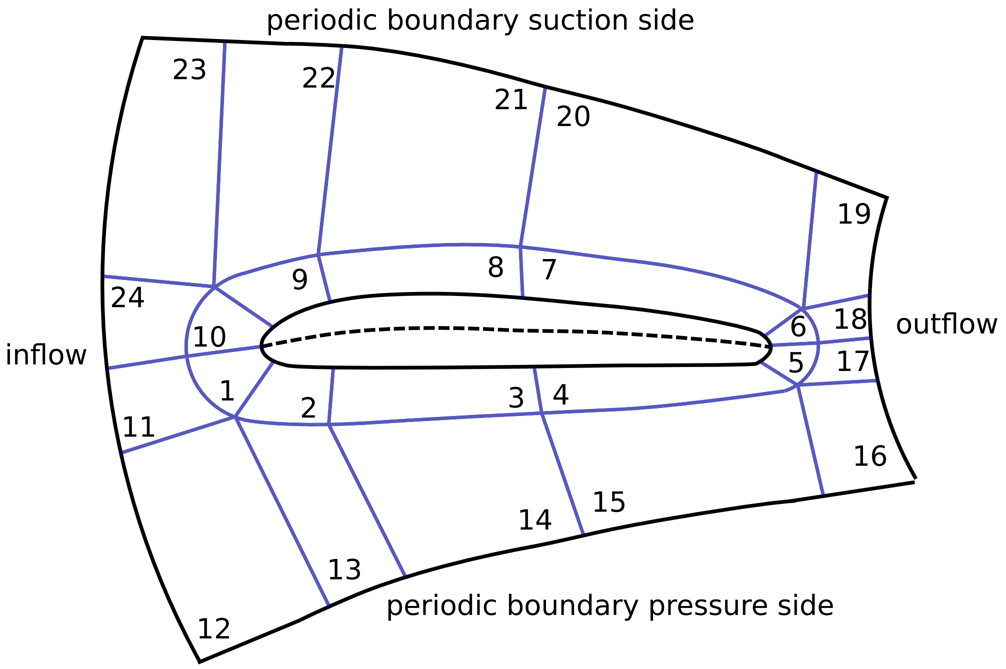

Figure 2.5: Sketch of the rotator topology which is extruded in the hub to shroud direction

Chapter 3

Mesh generation

3.1 Introduction

During the internship a python script was written that, in cooparation with the Eilmer3 code, creates a structured mesh from a given input. In this chapter this process is explained step by step, where a more detailed explanation of the code is given in the appendix A.
The overall process is as follows. At first the input will be used to create the surfaces of the rotor. From that the computational domain is created. Then the computational domain is subdevided into a structure similar to the one seen in figure 2.5.

3.2 Input

The input consists of points along the curves describing the pressure and suction side as well as the camber line of the blade. There are several of these curves at different hub to shroud locations. These points are made into curves with the Eilmer3 spline function, which results for example in figure 3.1 for the camberlines of the blade.

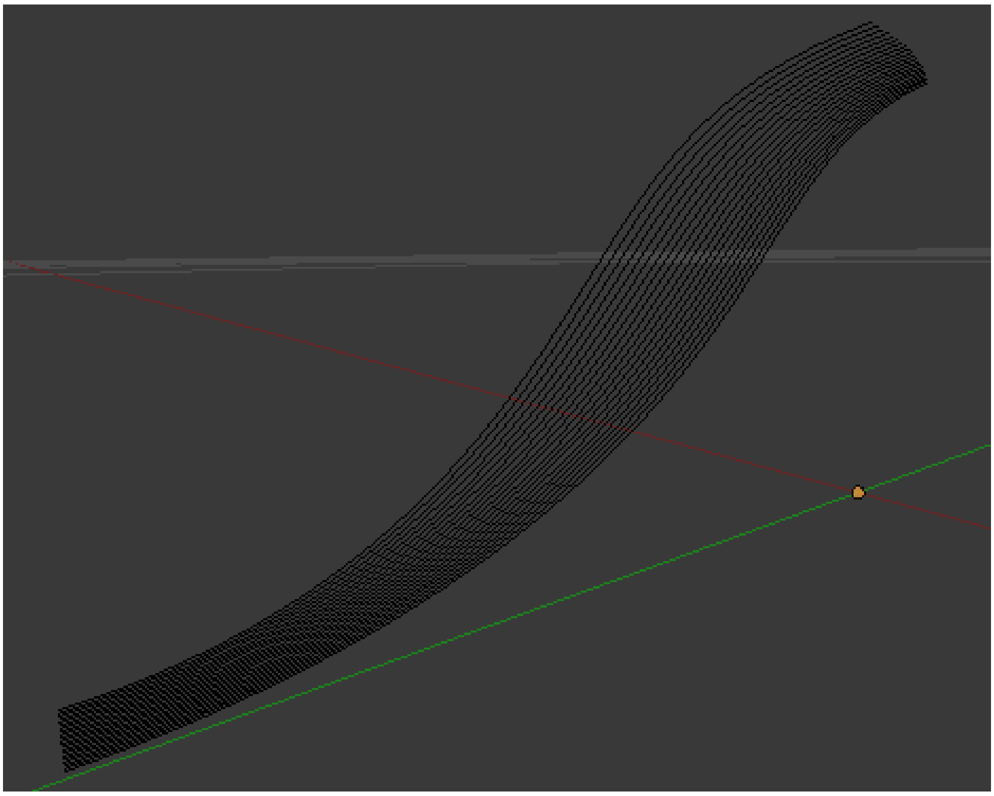

Figure 3.1: Camberline curves created from the input

3.3 Loft

From the curves given in the input a surface has to be created. This is done through lofting and while quite a few options are available to create surfaces with Eilmer3 this is not one of them. The following process describes how lofting was implemented.
Any continuous surface can be described with two coordinates, as can be seen in figure 3.2. The surface has an arbitrary distribution of points in the 3D space.
Define i and j as 0 at the beginning corner and 1 at their respective ends. Now a function $f(i, j)$ with $0 \geq i \geq 1$ and $0 \geq j \geq 1$ can describe the surface completely. This is applied to create a surface from the input curves.
In this case $i = 0$ is the inflow location, $i = 1$ is the outflow location, $j = 0$ is the hub location and $j = 1$ is the shroud location. Several curves in i direction are given in the input at different j positions. To obtain the coordinates of the surface for an arbitrary combination of i and j the following is done. Evaluate all input curves at location $i_{arbitrary}$ this will result in one point per input curve. A new curve in the j direction is then created using all these points. Evaluating this curve at its $j_{arbitrary}$ location gives the coordinates of the surface that one was looking for. The process above can be done for any combination of i and j and therefore describes the complete surface.
This is done for both the pressure side and the suction side of the surface.

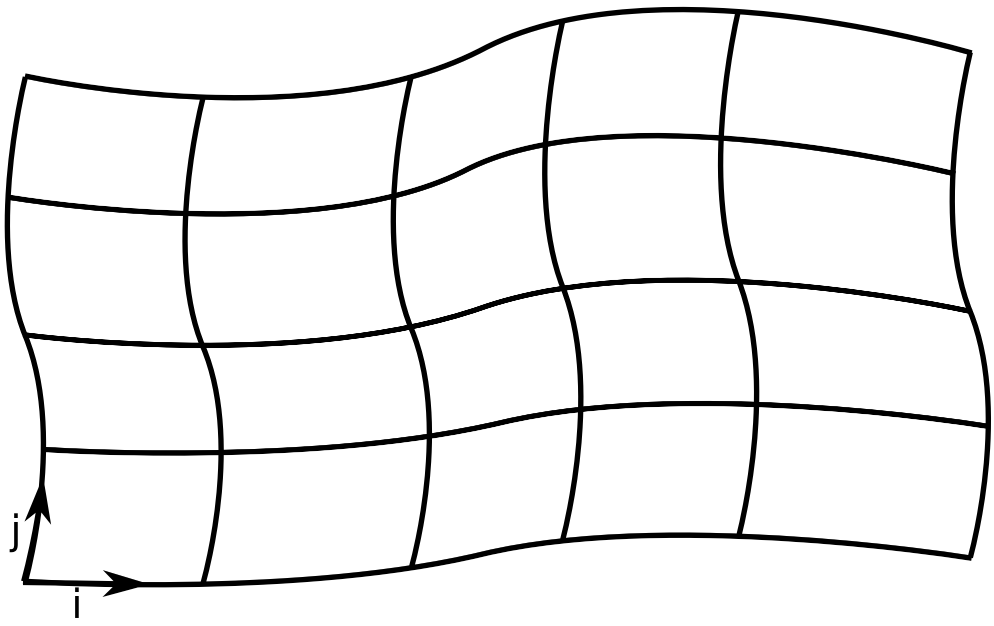

Figure 3.2: Sketch of a surface to explain lofting

3.4 Computational Domain

To create a hexahedral volume in the given environment it is sufficient to define all its six faces. This works in a similar way as described for the surface in the previous section, where a third computational dimension is added. The structure of these volume blocks is explained further in appendix A section A.3.
The whole computational domain is build up out of several blocks as seen in the topology sketch, figure 2.5. In the script the whole computational domain is created first by separately creating six volumes, as seen in figure 3.2. Their surfaces are then later used when blocks are created.
The division into volumes is sketched in figure 3.3 where this sketch has to be extruded in the hub to shroud direction to obtain volumes. It has to be mentioned that the clearance, the space between rotor and shroud is not yet taken into consideration in this code.
The surfaces of the volumes are of main interest here, especially the periodic boundaries.

- The hub and shroud surfaces follow from the geometry and can be implemented straight forward.
- The inflow surface is chosen to be part of a cylinder shell and the outflow surface is part of a disk.
- The surface of the blades the hub and the shroud follow from the input.

- The periodic boundaries are more complex and are explained below

The periodic boundaries cannot simply be surfaces with a constant azimuthal coordinate. That is because of the curvature of the blades, especially in the outflow section, which can be seen for example in figure 3.4. This curvature in combination with an option to use a large number of blades require a different kind of boundary.

The periodic boundaries consist of three parts, the inflow part belonging to volumes V5 and V6, the rotator part belonging to volume V1 and V4 and the outflow part belonging to V2 and V3. The inflow part flat surfaces with constant azimuthal position. The rotator part is a lofted surface created from rotated camberline curves. The outflow part of the periodic boundary is then created in a way that it smoothly connects to the camberline surface of V1 and V4. This is done by matching the slope $\frac{\partial\theta}{\partial z}$ of both surfaces at their connection.

With all surfaces the computational domain is defined as seen in figure 3.5 and the creation of the grid can begin.

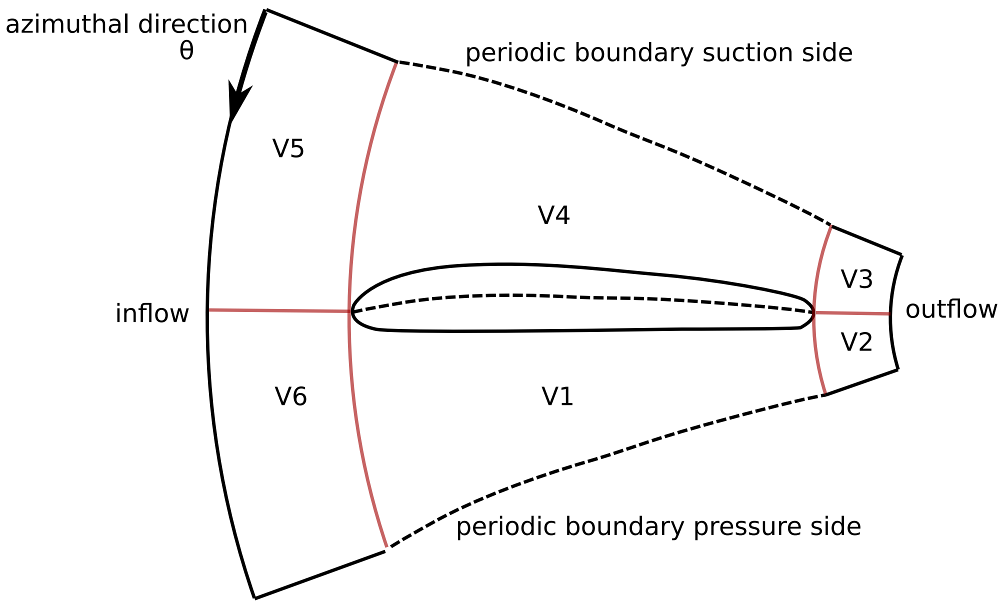

FIGURE 3.3: Sketch of the structure of the computational domain

Figure 3.4: Photo of a rotator section of a radial inflow turbine

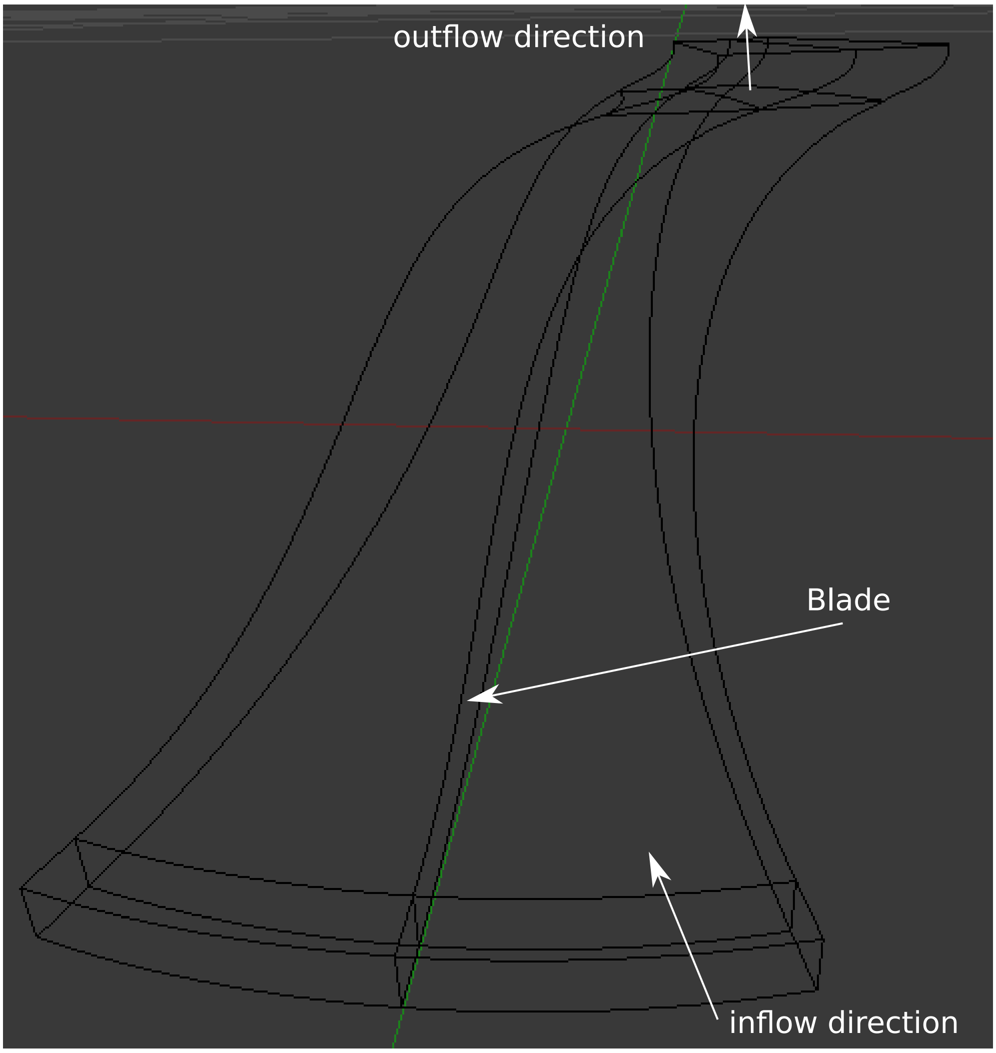

Figure 3.5: Image of the computational domain in 3D

3.5 Creation of the Topology

A sketch of the topology is given in figure 2.5. One has to keep in mind, that this is sketch of a cut through the volume at a hub to shroud location. Therefore the surface is not just 2D as displayed here, it actually follows the hub, inflow and outflow geometry. Thus the following operations seem straight forward but are complicated by the fact that they have to be done in the respective working surface.

The remaining lines (surfaces when extruded) that still need to be created are displayed blue in figure 2.5. First an O-grid line is created which has a constant distance to the blade. Then nodes are distributed over the inflow and outflow lines, over the periodic boundaries and over the O-grid line. These are then connected. This is done in a way that can easily be adapted, so that the number of blocks can be increased and their node position can be changed, see appendix A section A.5.1.

The resulting distribution in 3D can be seen in figure 3.6.

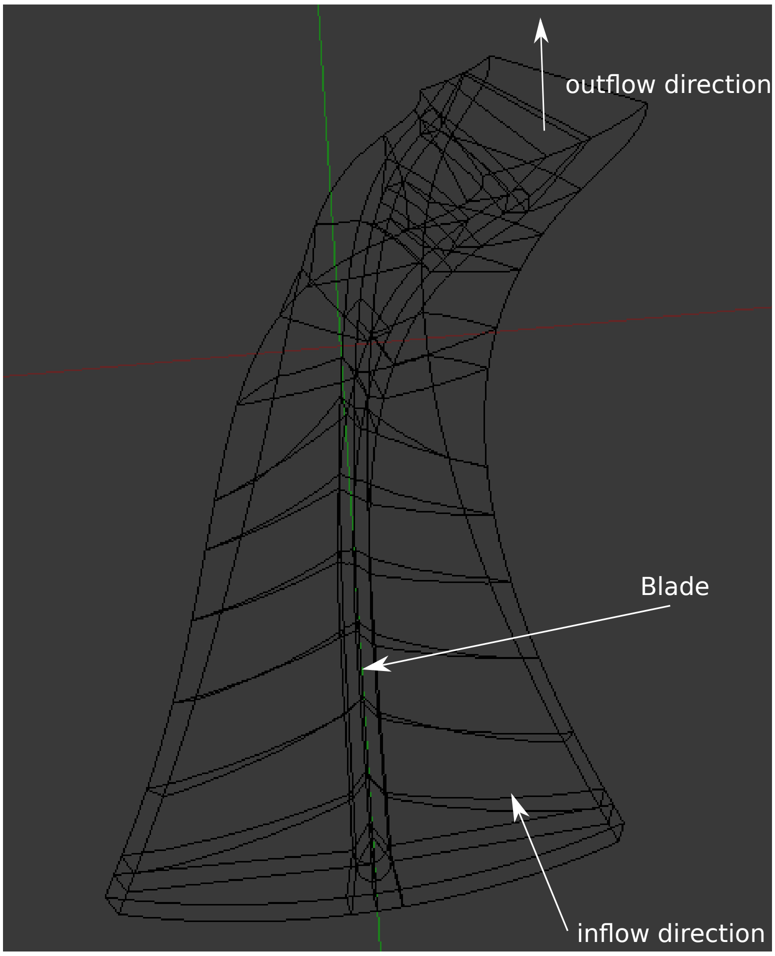

FIGURE 3.6: Image of the topology in 3D

Chapter 4

Summary

A script has been created to automatically create an OH-grid type structured mesh around blades of the rotor of an radial inflow turbine.
A way to loft a surface from a given set of curves has been found, which can easily be implemented into the Eilmer3 code and can be used for for many different applications.
The challenging geometry has been devided into an OH-grid type structure, where blocks are created automatically and the number of blocks and their location can be adjusted by changing very few input parameters.
The distribution of the cells along the boundaries still has to be adjusted to capture the boundary layer for example. Currently the cells are evenly divided along the block boundaries. Once this is achieved an automated process can be run to solve CFD calculations on this grid.
Another feature that might need to be added to the script is the ability to model the clearance in between blade and shroud.
Functions used in the code can be reused to create scripts for automatic mesh generation stators or other geometries.

Appendix A

Code explanations

In this appendix the script is explained in more detail. This includes an explanation of how to work with the script and which lines are responsible for what operation. Overall the script is run with the data preparation tool of Eilmer3. Command:
$\$e3prep.py --job = jobname$

The input that is required consists of ".obj" files containing curves on the pressure side, suction side and the camberline of the blade. These files need to be ordered in hub to shroud direction and they should start at the leading edge and end at the trailing edge. Next to the curves given in the input also the number of blades has to be defined, as well as the inflow and outflow length. This will determine the size of the computational domain.
In general only the file location and number of blade has to be given and the rest is done automatically. For further fine-tuning see the different sections in this appendix.
During the code creation the pressure and suction side did not start at the leading and trailing edge therefore only the camberlines were used. Once this is implemented certain curves in this code have to be changed, most importantly Volume lines (line 158,337), and lines 880 - 882,1101 have match the correct geometries.

A.1 Input interpreter

The location of the input curves is determined in code lines 20 to 34. These have to be changed to be able to run the script. The "loader" function and the "input interpreter" function are used together to create splines from the input.
They make use of the structure of the ".obj" file to extract data points. A spline using the Eilmer3 spline function is then created through every point per curve.

A.2 Loft

The idea behind lofting is explained is section 3.3. This procedure can be implemented into the current Eilmer3 environment using only 20 lines of code, see the "Loft" function in the python script, which is the input to the already existing "PyFunctionSurface" in Eilmer3 to create a lofted surface.
It evaluates all input curves at the desired i location and creates a spline through all these points. This spline will then be evaluated at the desired j location and thus the Cartesian coordinates for any combination of i and j can be obtained.
However, this function assumes that the curves are given in the same i direction and that they follow a strict order in the j direction otherwise the resulting surface does not make sense. Furthermore in case of the radial inflow turbine the coordinates are assumed to be such, that the center axis is in the z direction and goes through the origin of a Cartesian coordinate system.

A.3 Computational domain

The arrangement of volumes is shown in figure 3.3. In Eilmer3 the volumes are arranged as displayed in A.1. The code is generated in a way, that the top face corresponds with the shroud, the bottom face corresponds with the hub, the west surface belongs to the inflow direction and the east surface belongs to the outflow direction.
The volumes are created in the code in lines 155 to 645.

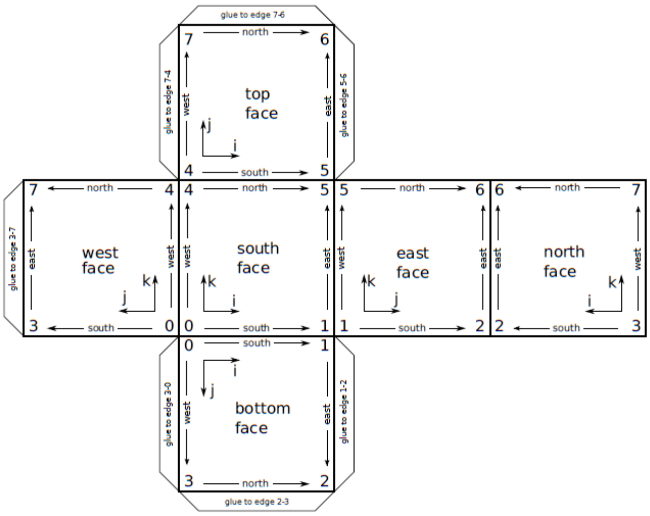

FIGURE A.1: Arrangement of a hexaedral block in Eilmer3 [4]

A.4 Automated block creation

A more detailed sketch of the topology is given in figure A.2. It contains the nomenclature used in the code. The following parts will explain how the remaining surfaces are created.

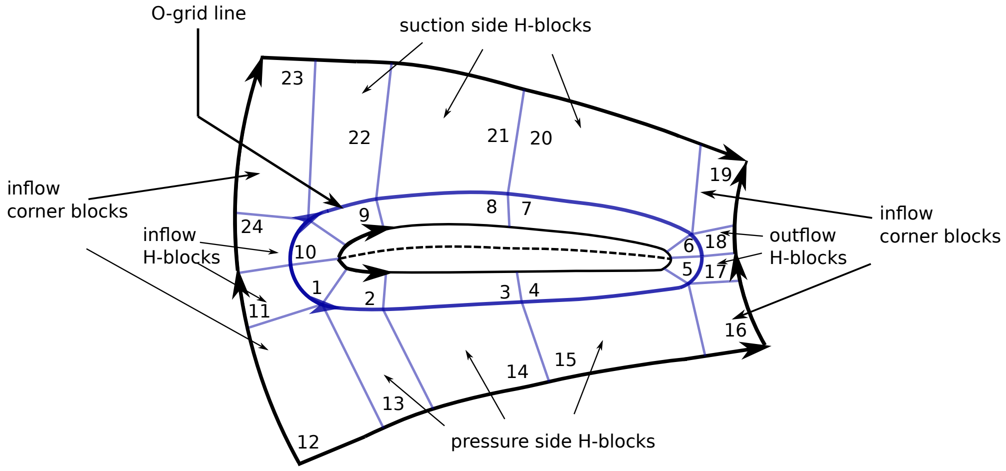

FIGURE A.2: Sketch of the structure of the computational domain

A.4.1 O-grid surface

The part of the code that is responsible for creating the O-grid surfaces is given in lines 655 to 888.

The O-grid line is created in the following way:

At 120 locations along the input curve points are created at a fixed distance to the blade (adjustable through `number_of_points_for_ogl` and `distance_ogl_blade`). It is made sure that these points lie on the working surface.

Through these points a curve is created to obtain a O-grid line. Then all the O-grid lines are lofted to create an O-grid surface.

A.4.2 Connection between nodes

The function `simple_connector` (lines 911 to 1096) creates a curve connecting two nodes. This curve is part of the working surface. The connecting surface is then created using its four surrounding paths (two times path on surface and two times simple conector curves).

A.4.3 Automated block creation functions

With the O-grid surface defined and the ability to connect nodes, automated functions are written to automatically create blocks (lines 1130 to 1945). Here a distinction is made between inflow H-blocks (`create_In_Block`), periodic side H-blocks (`create_H_Block`), outflow H-blocks (`create_Out_Block`), O-grid blocks (`create_O_Block`) and corner blocks, see sketch A.2.

A.5 Grid creation

Using all the functions described above the complete grid can be created starting with the pressure side of the computational domain (lines 2092 to 2303). First the inflow H-blocks are created, followed by the inflow corner block then the periodic H-blocks, afterwards the outflow corner block is created and then the outflow H-blocks are created. Finally the blocks in the O-grid section are created. This process is repeated for the suction side.

The number of blocks created depends on the number of nodes defined along each curve. Thus changing the location and number of the nodes defines the topology of the computational domain. A preliminary selection of node locations has been done for the given geometry and is given in the next section. However, the location and number of nodes might have to be changed for different geometries.

A final adjustment that has to be made by the user is to activate the block creation parts in the code that are currently commented and include cluster functions to distribute the nodes along each block.

A.5.1 Node location

Node locations are chosen along the periodic boundaries, the blade, the O-grid surface, the inflow surface and the outflow surface. Here 0 stands for the beginning of the surface and 1 for its end. The directions are shown in figure A.2.

The following node locations were chosen to produce figure 3.6 they can be found in the code at lines 1962 to 2074. A note of caution: When the number of inlet or outlet H-blocks is changed the corresponding node on the O-grid line has to be changed, see lines 2071 to 2074.

```
Nodes_on_Blade_pressure = [0,0.03,0.1,0.2,0.3,0.4,0.5,0.6,0.65,0.7,0.85,0.9,0.98,1]
Nodes_on_Blade_suction = [0,0.03,0.1,0.2,0.3,0.4,0.5,0.6,0.65,0.75,0.85,0.88,0.98,1]
Nodes_on_o_grid_line_pressure = [0,0.03,0.1,0.2,0.3,0.4,0.5,0.6,0.65,0.7,0.85,0.9,0.98,1]
Nodes_on_o_grid_line_suction = [0,0.03,0.1,0.2,0.3,0.4,0.5,0.6,0.65,0.7,0.82,0.9,0.96,1]
Nodes_on_inlet_pressure = [0.9,1]
Nodes_on_inlet_suction = [0.0,0.1]
```

```
Nodes_on_periodic_pressure = [0.15,0.28,0.36,0.4,0.44,0.475,0.5,0.55,0.57,0.60,0.62,0.65]
Nodes_on_periodic_suction = [0.15,0.28,0.36,0.4,0.44,0.475,0.5,0.55,0.60,0.62,0.7,0.9]
Nodes_on_outlet_pressure = [0.9,1]
Nodes_on_outlet_suction = [0.0,0.4]
```

A.6 Visualisation

During the creation of the script intermediate results had to be visualized. This is done both to check for failures in the code and to determine the location of certain blocks. The preferred way to do this is with the Blender [5] software. First the mesh generation script is used to create text files containing coordinates of points along curves that should be displayed. These text files should have the following structure:

```
x1 y1 z1
x2 y2 z2
...
```

In the code this is generally done by appending curve elements to the list of A. At the end of the code all curves contained in that list are written to separate text files. Then Blender can be started in scripting mode. Use the script below where the file location and number of curves of to be changed. Run the script and it will load and display the desired curves.

```
## Run this script in blender and it will create and display curves
## that come from input files contain only x,y and z coordinates of points (one point per line).
## Just change the file location and number of curves and press run. Then the curves will be displayed.

import bpy
from mathutils import Vector

w = 1                   # weight  just leave this as 1
number_of_curves = 10   # gives the number of curves that are loaded into blender

for n in range (0,number_of_curves):
    cList = None
    cList = []

    for line in open("/filelocation/file%d.txt" %n, "r"):        ## change this to the proper file location
        vals = line.split()
        x = float(vals[0])
        y = float(vals[1])
        z = float(vals[2])
        cList.append(Vector((x,y,z)))


    curvedata = bpy.data.curves.new(name='Curve', type='CURVE')
    curvedata.dimensions = '3D'

    objectdata = bpy.data.objects.new("ObjCurve", curvedata)
    objectdata.location = (0,0,0) #object origin
    bpy.context.scene.objects.link(objectdata)

    polyline = curvedata.splines.new('POLY')
    polyline.points.add(len(cList)-1)
```

```
    for num in range(len(cList)):
        x, y, z = cList[num]
        polyline.points[num].co = (x, y, z, w)
```

Appendix B

Mesh generation code

The whole code to generate the mesh

```
job_title = "Creating the mesh..."
print job_title

# Accept defaults for air giving R=287.1, gamma=1.4
select_gas_model(model='ideal gas', species=['air'])

initialCond = FlowCondition(p=1000.0, u=0.0, T=300.0)
M_inf = 4.0
u_inf = M_inf * initialCond.flow.gas.a
inflowCond  = FlowCondition(p=50.0e3, u=u_inf, T=300.0)


#################### input needed for geometry

number_of_blades = 12           #for now 3 blades not working dont know why :/ everything else is
clearance = 0.02     #not implemented yet
inflow_space = 0.05 #this factor times the camberline length gives the inflow length
outflow_space = 0.05

file_camber_hub = "/file_location/obj_3D_hub.obj"
file_camber_hub_to_shroud = "/file_location/obj_3D_hubShroudIntermediateLine%d.obj"
number_of_intermediate_camber_files = 20
file_camber_shroud = "/file_location/obj_3D_shroud.obj"


file_pressure_side_hub = "/file_location/obj_3D_hub_side2.obj"
file_pressure_side_hub_to_shroud = "/file_location/obj_3D_hubShroudIntermediates_pressureSide%d.obj"
number_of_intermediate_pressure_files = 20
file_pressure_side_shroud = "/file_location/obj_3D_shroud_side2.obj"

file_suction_side_hub = "/file_location/obj_3D_hub_side1.obj"
file_suction_side_hub_to_shroud = "/file_location/obj_3D_hubShroudIntermediates_suctionSide%d.obj"
number_of_intermediate_suction_files = 20
file_suction_side_shroud = "/file_location/obj_3D_shroud_side1.obj"

######################################## File loader
def loader(filename,number_of_files):   ### load .obj files that are ordered by numbers e.g. filename1 filename2 ... for number of files = 0 it just loads the file filename
    verts=[]                            ### input example:  filename =  "obj_3D_hubShroudIntermediateLine%d.obj"
    if number_of_files == 0:
        for line in open(filename, "r"):
            vals = line.split()
            if vals[0] == "v":
                v = map(float, vals[1:4])
```

```
                verts.append(v)
    else:
        for line in open(filename %number_of_files, "r"):
                vals = line.split()
                if vals[0] == "v":
                    v = map(float, vals[1:4])
                    verts.append(v)

    return verts

#################################### Create input curves from given files in the order : Hub, hub to shroud intermediates , Shroud. All are located in input_curves
#################################### The curve starts at the inflow blade, ends at outflow of blade
def input_interpreter(hublocation,hub_to_shroud_location,shroudlocation, number_of_inbetween_files):
    input_curves=[]
    dummy = []

    linecoords = loader(hublocation,0)        ### load all the points from one file, then create a spline through it
    for i in range(0,len(linecoords)):
            dummy.append(Vector(linecoords[i][0],linecoords[i][1],linecoords[i][2]))
    input_curves.append(Spline(dummy))


    for n in range(1,number_of_inbetween_files+1):
        Q=None
        Q=[]
        linecoords=loader(hub_to_shroud_location,n)
        for i in range(0,len(linecoords)):
                Q.append(Vector(linecoords[i][0],linecoords[i][1],linecoords[i][2]))
        input_curves.append(Spline(Q))

    dummy=None
    dummy=[]
    linecoords = loader(shroudlocation,0)
    for i in range(0,len(linecoords)):
            dummy.append(Vector(linecoords[i][0],linecoords[i][1],linecoords[i][2]))
    input_curves.append(Spline(dummy))

    return input_curves


################################### Define Loft surface ... r = 0 inflow r = 1 outflow , s = 0 hub s = 1 shroud
################################### PySurface only allows 2 inputs.. therefore make sure you call input_curves = input_interpreter(hub,hts,shroud,NoF) first!
###################################  example :  input_curves = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud, number_of_intermediate_camber_files)
###################################             desired_lofted_surface = PyFunctionSurface(LoftUserDefined)
################ Volume 1

input_curves = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud, number_of_intermediate_camber_files)

def Loft(input_curves):
    input_curvesss = []
    for i in range (0,len(input_curves)):
        input_curvesss.append(input_curves[i].clone())
    def LoftUserDefined(r,s):

        Q=None
        Q=[]

        r_Curves=input_curvesss

        for n in range(0,len(r_Curves)):
            Q.append(r_Curves[n].eval(r))

        s_Curve=Spline(Q)
        Coords = s_Curve.eval(s)

```

```
        return (Coords.x,Coords.y,Coords.z)
    return LoftUserDefined


#################################### rotate the lofted surface counterclockwise around z BE CAREFULL ABOUT input_curves!!

def LoftRotator(input_curves):
    input_curvesss = []
    for i in range (0,len(input_curves)):
        input_curvesss.append(input_curves[i].clone())
    def LoftUserDefined(r,s):

        Q=None
        Q=[]

        r_Curves=input_curvesss

        for n in range(0,len(r_Curves)):
            Q.append(r_Curves[n].eval(r))

        s_Curve=Spline(Q)
        Coords = s_Curve.eval(s)

        return (Coords.x,Coords.y,Coords.z)

    def LoftRotate(r,s):
        from math import cos, sin, pi

        theta = 2 * pi / number_of_blades *1/2 #only rotate half the angle but in both directions of course .... =.=
        cc = cos (theta)
        ss = sin (theta)

        Coords = LoftUserDefined(r,s)
        x = Coords[0]
        y = Coords[1]
        z = Coords[2]
        x_r = x*cc-y*ss
        y_r=x*ss+y*cc

        return (x_r,y_r,z)

    return LoftRotate


####################################
#################################### First Volume according to my definition of top bot etc.
print("Volume1################################")

#north = PyFunctionSurface(LoftUserDefined)
north = PyFunctionSurface(Loft(input_curves))
#south = PyFunctionSurface(LoftRotate)
south = PyFunctionSurface(LoftRotator(input_curves))

p2 = north.eval(1.0, 0.0)
p3 = north.eval(0.0, 0.0)
p6 = north.eval(1.0, 1.0)
p7 = north.eval(0.0, 1.0)

p0 = south.eval(0.0, 0.0)
p1 = south.eval(1.0, 0.0)
p4 = south.eval(0.0, 1.0)
p5 = south.eval(1.0, 1.0)

# print "p0=", p0, "p1=", p1



print("top---------------------------------------")
c56 = Arc(p5, p6,Vector(0,0,p5.z))
c47 = Arc(p4, p7,Vector(0,0,p4.z))
c76 = PathOnSurface(north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
c45 = PathOnSurface(south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
top = CoonsPatch(c45, c76, c47, c56, "top")
```

```


print("bottom_______________________________________")
c03 = Arc(p0, p3,Vector(0,0,p0.z))
c12 = Arc(p1, p2,Vector(0,0,p1.z))
c01 = PathOnSurface(south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
c32 = PathOnSurface(north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
bottom = CoonsPatch(c01, c32, c03, c12, "bottom")


print("west_______________________________________")
c37 = PathOnSurface(north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
c04 = PathOnSurface(south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
west = CoonsPatch(c03, c47, c04, c37,"west")


print("east_______________________________________")
c26 = PathOnSurface(north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
c15 = PathOnSurface(south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
east = CoonsPatch(c12, c56, c15, c26,"east")


pvolume = ParametricVolume(north, east, south, west, top, bottom, "Pressure_side_volume")


#blk1 = Block3D(label="first-block", nni=20, nnj=20, nnk=20,
#               parametric_volume=pvolume,
#               fill_condition=initialCond)
#blk1.set_BC("WEST", "SUP_IN", inflow_condition=inflowCond)
#blk1.set_BC("SOUTH", "SUP_OUT")
#blk1.set_BC("TOP", "SUP_OUT")
#blk1.set_BC("BOTTOM", "SUP_OUT")


################################### Volume number 6 according to my intermediate nomenclature
    , inlfow pressure block
################################### Extrude for inflow  INPUT_CURVES!!!! WARNING 2 solutions
    for x and y cause of sqrt ... :/
print("Volume6################################")

def extrude(point,extrude_length):
    from math import sqrt

    x = point.x
    y = point.y
    z = point.z

    r2 = sqrt(x**2+y**2)+extrude_length
    a = x / y

    y = sqrt(r2**2/(1+a**2))
    x = y*a

    return Vector(x,y,z)

v6p2 = p3
v6p3 = extrude(p3,inflow_space*input_curves[0].length())
v6p6 = p7
v6p7 = Vector(v6p3.x,v6p3.y,v6p6.z)

v6p0 = extrude(p0,inflow_space*input_curves[0].length())
v6p1 = p0
v6p5 = p4
v6p4 = Vector(v6p0.x,v6p0.y,v6p5.z)

#print "v6p4=", v6p4, "v6p5=", v6p5, "v6p6=", v6p6, "v6p7=", v6p7
#print "v6p0=", v6p0, "v6p1=", v6p1, "v6p2=", v6p2, "v6p3=", v6p3


print("v6top_______________________________________")
v6c56 = c47
v6c45 = Line(v6p4,v6p5)
v6c76 = Line(v6p7,v6p6)
v6c47 = Arc(v6p4,v6p7,Vector(0,0,v6p4.z))
v6top = CoonsPatch(v6c45, v6c76, v6c47, v6c56, "v6top")
```

```
#


print(" v6bottom_______________________________________")
v6c03 = Arc(v6p0, v6p3,Vector(0,0,v6p0.z))
v6c12 = c03
v6c01 = Line(v6p0,v6p1)
v6c32 = Line(v6p3,v6p2)
v6bottom = CoonsPatch(v6c01, v6c32, v6c03, v6c12, "v6bottom")
#
#
print(" v6west_______________________________________")
v6c37 = Line(v6p3,v6p7)
v6c04 = Line(v6p0,v6p4)
v6west = CoonsPatch(v6c03, v6c47, v6c04, v6c37,"v6west")
#
#
print(" v6east_______________________________________")
v6c26 = c37
v6c15 = c04
#v6east = CoonsPatch(v6c12, v6c56, v6c15, v6c26," v6east")
v6east = west

print(" v6north_______________________________________")
v6north = CoonsPatch(v6c32, v6c76, v6c37, v6c26, "v6north")

print(" v6south_______________________________________")
v6south = CoonsPatch(v6c01, v6c45, v6c04, v6c15, "v6south")




v6pvolume = ParametricVolume(v6north, v6east, v6south, v6west, v6top, v6bottom, "
     Inflow_Pressure_side_block")

#blk6 = Block3D(label="block6", nni=20, nnj=20, nnk=20,
#               parametric_volume=v6pvolume,
#               fill_condition=initialCond)


##################################### Block number 4 according to my intermediate nomenclature ,
      suction side block
##################################### INPUT_CURVES!!!!
print("Volume4###################################")
input_curves = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud,
     number_of_intermediate_camber_files)

def LoftRotatorClockwise(input_curves):
    input_curvesss = []
    for i in range (0,len(input_curves)):
        input_curvesss.append(input_curves[i].clone())
    def LoftUserDefined(r,s):

        Q=None
        Q=[]

        r_Curves=input_curvesss

        for n in range(0,len(r_Curves)):
            Q.append(r_Curves[n].eval(r))

        s_Curve=Spline(Q)
        Coords = s_Curve.eval(s)

        return (Coords.x,Coords.y,Coords.z)

    def LoftRotateClockwise(r,s):
        from math import cos, sin, pi

        theta = -2 * pi / number_of_blades *1/2
        cc = cos (theta)
        ss = sin (theta)

        Coords = LoftUserDefined(r,s)
        x = Coords[0]
```

```
        y = Coords[1]
        z = Coords[2]
        x_r = x*cc-y*ss
        y_r=x*ss+y*cc

        return (x_r,y_r,z)

    return LoftRotateClockwise

v4north = PyFunctionSurface(LoftRotatorClockwise(input_curves))
v4south = PyFunctionSurface(Loft(input_curves))

#v4north = PyFunctionSurface(LoftRotateClockwise)
#v4south = PyFunctionSurface(LoftUserDefined)

v4p2 = v4north.eval(1.0, 0.0)
v4p3 = v4north.eval(0.0, 0.0)
v4p6 = v4north.eval(1.0, 1.0)
v4p7 = v4north.eval(0.0, 1.0)

v4p0 = v4south.eval(0.0, 0.0)
v4p1 = v4south.eval(1.0, 0.0)
v4p4 = v4south.eval(0.0, 1.0)
v4p5 = v4south.eval(1.0, 1.0)

# print "p0=", p0, "p1=", p1



print("v4top_______________________________________")
v4c56 = Arc(v4p5, v4p6,Vector(0,0,v4p5.z))
v4c47 = Arc(v4p4, v4p7,Vector(0,0,v4p4.z))
v4c76 = PathOnSurface(v4north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
v4c45 = PathOnSurface(v4south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
v4top = CoonsPatch(v4c45, v4c76, v4c47, v4c56, "v4top")


print("v4bottom_______________________________________")
v4c03 = Arc(v4p0, v4p3,Vector(0,0,v4p0.z))
v4c12 = Arc(v4p1, v4p2,Vector(0,0,v4p1.z))
v4c01 = PathOnSurface(v4south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
v4c32 = PathOnSurface(v4north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
v4bottom = CoonsPatch(v4c01, v4c32, v4c03, v4c12, "v4bottom")


print("v4west_______________________________________")
v4c37 = PathOnSurface(v4north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
v4c04 = PathOnSurface(v4south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
v4west = CoonsPatch(v4c03, v4c47, v4c04, v4c37,"v4west")


print("v4east_______________________________________")
v4c26 = PathOnSurface(v4north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
v4c15 = PathOnSurface(v4south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
v4east = CoonsPatch(v4c12, v4c56, v4c15, v4c26,"east")


v4pvolume = ParametricVolume(v4north, v4east, v4south, v4west, v4top, v4bottom, "
    Pressure_side_block")


#blk4 = Block3D(label="fourth-block", nni=20, nnj=20, nnk=20,
#               parametric_volume=v4pvolume,
#               fill_condition=initialCond)


###################################Volume 5
print("Volume5#################################")
v5p2 = v4p3
v5p6 = v4p7
v5p3 = extrude(v5p2,v6c32.length())
v5p7 = Vector(v5p3.x,v5p3.y,v5p6.z)

v5p0 = v6p3
v5p1 = v6p2
```

```
v5p4 = v6p7
v5p5 = v6p6




print("v5top_______________________________________")
v5c56 = v4c47
v5c45 = v6c76
v5c76 = Line(v5p7,v5p6)
v5c47 = Arc(v5p4,v5p7,Vector(0,0,v5p4.z))
v5top = CoonsPatch(v5c45, v5c76, v5c47, v5c56, "v5top")
#


print("v5bottom_______________________________________")
v5c03 = Arc(v5p0, v5p3,Vector(0,0,v5p0.z))
v5c12 = v4c03
v5c01 = v6c32
v5c32 = Line(v5p3,v5p2)
v5bottom = CoonsPatch(v5c01, v5c32, v5c03, v5c12, "v5bottom")
#
#
print("v5west_______________________________________")
v5c37 = Line(v5p3,v5p7)
v5c04 = v6c37
v5west = CoonsPatch(v5c03, v5c47, v5c04, v5c37,"v6west")
#
#
print("v5east_______________________________________")
v5c26 = v4c37
v5c15 = v4c04
#v5east = CoonsPatch(v5c12, v5c56, v5c15, v5c26,"v5east")
v5east = v4west

print("v5north_______________________________________")
v5north = CoonsPatch(v5c32, v5c76, v5c37, v5c26, "v5north")

print("v5south_______________________________________")
#v5south = CoonsPatch(v5c01, v5c45, v5c04, v5c15, "v5south")
v5south = v6north

v5pvolume = ParametricVolume(v5north, v5east, v5south, v5west, v5top, v5bottom, "
    suction_side_inlet_block")
#blk5 = Block3D(label="fourth-block", nni=20, nnj=20, nnk=20,
#              parametric_volume=v5pvolume,
#              fill_condition=initialCond)


#print "v5c32", v5c32.eval(0), v5c32.eval(1), "v5c76", v5c76.eval(0), v5c76.eval(1), "v5c37",
    v5c37.eval(0), v5c37.eval(1),"v5c26", v5c26.eval(0), v5c26.eval(1)
#print "pvolume", pvolume.eval(0,0,0), "pvolume",pvolume.eval(1,0,1)
#print "c04", c04.eval(0), c04.eval(1)

####################################Volume 2 is the outflow pressure side block INPUT CURVES!
print("Volume2################################")

##########To create the periodic boundary without edges, the camberline/surface is evaluated
    at its end (outflow)
##########the curve will be mirrored/when the angles are concerned but the radius remains
    constant with increasing z
### Refinement options:
z_resolution = 0.01
points_in_spline = 8

def get_curved_outflow_points(surface,a,b):
    from numpy import sqrt, arctan2
    from math import cos, sin
    ## Find the length in z-direction
    z_length = outflow_space*input_curves[0].length()
    x_start = surface.eval(a,b).x
    y_start = surface.eval(a,b).y
    z_start = surface.eval(a,b).z
    r_start = sqrt(x_start**2+y_start**2)
    theta_start = arctan2(y_start,x_start)
```

```

    ## Find the coresonding meridional percentage
    length_along_curve = 1
    while length_along_curve > 0:
        if surface.eval(length_along_curve,b).z < z_start-z_length:
            while length_along_curve < 1:
                if surface.eval(length_along_curve,b).z > z_start-z_length:
                    break
                length_along_curve += z_resolution
            break
        length_along_curve -= 0.1

    points_for_spline=[]
    points_for_spline.append(Vector(x_start,y_start,z_start))

    for i in range(1,points_in_spline+1):
        step = (1 - length_along_curve)/points_in_spline
        x = surface.eval(a-i*step,b).x
        y = surface.eval(a-i*step,b).y
        z = surface.eval(a-i*step,b).z

        theta= arctan2(y,x)
        theta_before = arctan2(points_for_spline[-1].y,points_for_spline[-1].x)
        theta_new = theta_before + (theta_start - theta)*(1-i/float(points_in_spline))**1.5

        x_new = cos(theta_new) * r_start
        y_new = sin(theta_new) * r_start
        z_new = z_start + (z_start-z)

        points_for_spline.append(Vector(x_new,y_new,z_new))
    return points_for_spline

points_for_spline_v2c45 = get_curved_outflow_points(top,1,0)
v2c45 = Spline(points_for_spline_v2c45)

points_for_spline_v2c01 = get_curved_outflow_points(bottom,1,0)
v2c01 = Spline(points_for_spline_v2c01)

points_for_spline_v2c76 = get_curved_outflow_points(top,1,1)
v2c76 = Spline(points_for_spline_v2c76)

points_for_spline_v2c32 = get_curved_outflow_points(bottom,1,1)
v2c32 = Spline(points_for_spline_v2c32)


v2p0 = p1
v2p3 = p2
v2p4 = p5
v2p7 = p6

v2p1 = v2c01.eval(1)
v2p2 = v2c32.eval(1)
v2p5 = v2c45.eval(1)
v2p6 = v2c76.eval(1)

#print "p0=", v2p0, "p1=", v2p1, "p2=", v2p2, "p3=", v2p3, "p4=", v2p4, "p5=", v2p5, "p6=", v2p6, "p7=", v2p7

print("v2top________________________________________")
v2c56 = Arc(v2p5, v2p6,Vector(0,0,v2p5.z))
#v2c45 = Line(v2p4,v2p5)
#v2c76 = Line(v2p7,v2p6)
v2c47 = c56
v2top = CoonsPatch(v2c45, v2c76, v2c47, v2c56, "v2top")
#


print("v2bottom_____________________________________")
v2c03 = c12
v2c12 = Arc(v2p1, v2p2,Vector(0,0,v2p1.z))
#v2c32 = Line(v2p3,v2p2)
v2bottom = CoonsPatch(v2c01, v2c32, v2c03, v2c12, "v2bottom")
#
#
print("v2west_______________________________________")
```

```
v2c37 = c26
v2c04 = c15
v2west = east
#
#
print("v2east----------------------------------------")
v2c26 = Line(v2p2,v2p6)
v2c15 = Line(v2p1,v2p5)
v2east = CoonsPatch(v2c12, v2c56, v2c15, v2c26,"v2east")


print("v2north----------------------------------------")
v2north = CoonsPatch(v2c32, v2c76, v2c37, v2c26, "v2north")

print("v2south----------------------------------------")
v2south = CoonsPatch(v2c01, v2c45, v2c04, v2c15, "v2south")

v2pvolume = ParametricVolume(v2north, v2east, v2south, v2west, v2top, v2bottom, "pressure_side_outlet_block")
#
#print "south", v2south.eval(1,0), "east", v2east.eval(0,0), " bottom", v2bottom.eval(1,0)
#
#print "volume p 100", v2pvolume.eval(1)

#blk2 = Block3D(label="fourth-block", nni=20, nnj=20, nnk=20,
#               parametric_volume=v2pvolume,
#               fill_condition=initialCond)

###################################Volume 3 is the outflow suction side block
print("Volume3#################################")

points_for_spline_v3c76 = get_curved_outflow_points(v4top,1,1)
v3c76 = Spline(points_for_spline_v3c76)

points_for_spline_v3c32 = get_curved_outflow_points(v4bottom,1,1)
v3c32 = Spline(points_for_spline_v3c32)

points_for_spline_v3c45 = get_curved_outflow_points(v4top,1,0)
v3c45 = Spline(points_for_spline_v3c45)

points_for_spline_v3c01 = get_curved_outflow_points(v4bottom,1,0)
v3c01 = Spline(points_for_spline_v3c01)

v3p0 = v4p1
v3p3 = v4p2
v3p4 = v4p5
v3p7 = v4p6


v3p2 = v3c32.eval(1)
v3p1 = v3c01.eval(1)
v3p6 = v3c76.eval(1)
v3p5 = v3c45.eval(1)

#from math import sqrt
#print sqrt(v3c32.eval(1).x**2+v3c32.eval(1).y**2), sqrt(v3p1.x**2+v3p1.y**2)


print("v3top----------------------------------------")
v3c56 = Arc(v3p5, v3p6,Vector(0,0,v3p5.z))
#v3c45 = Line(v3p4,v3p5)
v3c47 = v4c56
v3top = CoonsPatch(v3c45, v3c76, v3c47, v3c56, "v3top")
#


print("v3bottom----------------------------------------")
v3c03 = v4c12
v3c12 = Arc(v3p1, v3p2,Vector(0,0,v3p1.z))
#v3c01 = Line(v3p0,v3p1)
v3bottom = CoonsPatch(v3c01, v3c32, v3c03, v3c12, "v3bottom")
#
#
print("v3west----------------------------------------")
v3c37 = v4c26
```

```
v3c04 = v4c15
#v3west = CoonsPatch(v3c03, v3c47, v3c04, v3c37,"v3west")
v3west = v4east
#
#
print("v3east_______________________________________")
v3c26 = Line(v3p2,v3p6)
v3c15 = Line(v3p1,v3p5)
v3east = CoonsPatch(v3c12, v3c56, v3c15, v3c26,"v3east")


print("v3north_______________________________________")
v3north = CoonsPatch(v3c32, v3c76, v3c37, v3c26, "v3north")

print("v3south_______________________________________")
#v3south = CoonsPatch(v3c01, v3c45, v3c04, v3c15, "v3south")
v3south = v2north

v3pvolume = ParametricVolume(v3north, v3east, v3south, v3west, v3top, v3bottom, "pressure_side_outlet_block")


#blk3 = Block3D(label="fourth-block", nni=20, nnj=20, nnk=20,
#               parametric_volume=v3pvolume,
#               fill_condition=initialCond)



##########--------------------------------#################--------------------#######################------------#################
# ##########################################################################################################################
# ##########################################################################################################################
# ##########################################################################################################################
######## Now that the control volume is created, the blocks will be made. We cannot use every surface of volume 1 to 6 since an OH grid mesh will be
######## made but having a control volume by itself might come in handy sometimes. (with wich I am trying to say it was not useless to create all the 6 volumes)
######## The control volume will be sliced up in surfaces in hub to shroud direction and the 3d block will be an extrusion of the 2d patches created on that surface.
######## These surfaces is split by an O-gridline and after that into surfaces. Maybe I should add a figure in the folder to make it easier to understand.

######  bent / revolve ... slice, z-r plane
from numpy import sqrt, arctan2, pi, arccos
from math import cos, sin
number_of_points_for_ogl = 120
distance_ogl_blade = float(0.001)
arc_resolution =10**-3     #this should not be at 10^-2 thats not good enough!

number_of_points_on_connection = 10

def get_radius(point):      #The distance between the z-axis and a point has to be determined a couple of times, which is normally the radius (at least one exception)
    from math import sqrt
    radius = sqrt(point.x**2+point.y**2)
    return radius


input_curves = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud,number_of_intermediate_camber_files)


def ogl_creator(input_curves,camber_curves,pressure_or_suction):            #pressure_or_suction = 1 pressure side,             pressure_or_suction = -1 suction side

    oglines = []

    def reposition_into_plane(point_to_be_repositioned,p1,i): #bent and rotate about p1      #i is the position along the blade
```

```
        u_ogl_point=point_to_be_repositioned
        theta = arctan2( p1.y, p1.x )
        r = get_radius(p1)
        z = p1.z
        u_theta = arctan2(u_ogl_point.y,u_ogl_point.x)
        u_r = get_radius(u_ogl_point)
        ur_projectedinto_z_r_plane = cos(u_theta-theta) * u_r

        bent_arc_length = sqrt( (ur_projectedinto_z_r_plane -r)**2 + (u_ogl_point.z-z)**2  )
        #print "bent_arc_length",bent_arc_length
        t_arc_length = 0
        l=0
        tp=[Vector(0,0,0),Vector(0,0,0)]
        if abs(u_ogl_point.z-z) == 0 :  #either the point has to be in the inlet area or, it will also have the same radius (distance to z) in its final form
                x1 = r*cos(theta)+ (bent_arc_length) * cos (theta)  #in case of same r , bent arc length is zero
                y1 = r*sin(theta)+ (bent_arc_length) * sin (theta)
                z1 = z
                tp[1] = Vector( x1,y1,z1 )          #tp[1]? cause of the programing style atm
                tp[0]=tp[1].clone()
        else:
            d = (u_ogl_point.z-z) / abs(u_ogl_point.z-z)    # this will be +1 or -1 and thus determines in which direction to go on the z-r curve
            tp = [p1 , Vector(0,0,0)]
            while t_arc_length < bent_arc_length:
                if (i * 1/float(number_of_points_for_ogl)+d*l* arc_resolution ) >= 0:       # determines wether we are in inlet outlet or in betweeen
                    if (i * 1/float(number_of_points_for_ogl)+d*l* arc_resolution ) <= 1:   # we are in between inlet and outlet
                        tp[1] = input_curves[n].eval( i * 1/float(number_of_points_for_ogl)+d*l* arc_resolution )
                        t_r = get_radius(tp[1])
                        t_z = tp[1].z
                        t_dr = t_r-get_radius(tp[0])
                        t_dz = t_z - tp[0].z
                        t_arc_length = t_arc_length + sqrt (t_dr**2+t_dz**2)
                        #tp[0]=tp[1].clone()
                        tp[0]=Vector(cos(theta)*t_r,sin(theta)*t_r,t_z).clone()
                        l+=1

                    elif (i * 1/float(number_of_points_for_ogl)+d*l* arc_resolution) > 1: # outlet
                        tp[1] = Vector(camber_curves[n].eval(1).x,camber_curves[n].eval(1).y, tp[0].z + bent_arc_length-t_arc_length)
                        t_r = get_radius(tp[1])
                        t_z = tp[1].z
                        t_arc_length = bent_arc_length
                        #tp[0]=tp[1].clone()
                        tp[0]=Vector(cos(theta)*t_r,sin(theta)*t_r,t_z).clone()
                        #print "larger than one detected"
                elif (i * 1/float(number_of_points_for_ogl)+d*l* arc_resolution ) < 0:  #inlet
                    tp_theta = arctan2(tp[0].y,tp[0].x)
                    r_max = get_radius(camber_curves[n].eval(0))
                    x1 = r_max*cos(tp_theta)+ (bent_arc_length-t_arc_length) * cos (tp_theta)
                    y1 = r_max*sin(tp_theta)+ (bent_arc_length-t_arc_length) * sin (tp_theta)
                    z1 = camber_curves[n].eval(0).z
                    tp[1] = Vector( x1,y1,z1 )
                    t_arc_length = bent_arc_length
                    tp[0]=tp[1].clone()
                    #print "detected, point =%s, loop iterations=%s" %(i,l), "tp_r=", get_radius(tp[0])-r_max
                    #print "bent_arc_length=",bent_arc_length
        b_ogl_point = tp[0]         #now that we got the bent point, we have to revolve it around the z-axis
        tp_theta = arctan2(tp[0].y,tp[0].x)
        distance_u_ogl_point_to_zr_plane = sin(u_theta-theta) * u_r
        revolving_angle = distance_u_ogl_point_to_zr_plane / r    # d / (2Pi*r) * 2Pi = angle of an arc of length d with radius r
        ogl_point = b_ogl_point.rotate_about_zaxis(revolving_angle).clone()

        return ogl_point


    for n in range (0,len(input_curves)):
```

```
        r_max = get_radius(input_curves[n].eval(0))
        z_max = input_curves[n].eval(1).z
        ogl_points= None
        ogl_points = []
        for i in range (0,number_of_points_for_ogl):                                   #first get the 0 g-line curve and afterwards we connect the curves for pressure and suction
            p1 = input_curves[n].eval( i * 1/float(number_of_points_for_ogl) )    #determine 2 points on the blade to get a tangent vector to the blade p2-p1 = tangent vector
            p2 = input_curves[n].eval( (i+1) * 1/float(number_of_points_for_ogl) )

            theta = arctan2( p1.y, p1.x )                                               #determine the azimuthal angle (cylindrical coordinate system)
            r = get_radius (p1)                                                         #determine the radius (cylindrical coordinate system)
            z = p1.z                                                                    #determine the radius (cylindrical coordinate system)

            dr = get_radius (p2)- r                                                     #dr and dz together give a vector tangential to the slice at point p1
            dz = p2.z - z

            normal_to_slice = cross( Vector(-sin(theta),cos(theta),0), Vector(cos(theta)*dr,sin(theta)*dr,dz) )     #normal to slice pointing towards hub at point p1

            if pressure_or_suction > 0 :        #pressure side
                ogl_point_direction = cross (p2-p1,normal_to_slice)      #this vector is orthogonal to both the normal to the slice as well as the tangential to the blade
            elif pressure_or_suction < 0:       #suction side
                ogl_point_direction = cross (normal_to_slice,p2-p1)

            u_ogl_point = p1 + ogl_point_direction.norm()*distance_ogl_blade            # u_ogl_point = unbent and "un" rotated its in the plane which is tangential to the slice in p1
                                                                                        # to put the point into the slice surface we have to bent and rotate our tangential plane
            ## a litlle intermezzo to explain: u_ogl_point is now on a plane which is tangential to the slice. But this means it is not on the surface we want it to be.
            ## Thinking in extremes might help, so imagine the distance between the blade and O-gridline is made very large for example a 100 times the distance between two blades.
            ## The point wouldn't even be close to the turbine any more. In the following we force the point to be on the slice surface by bending and rotating the tangential plane.
            ## This is done in the following way: 3 coordinates define our slice: r,theta and z. All of these can be gathered from the input curves.
            ## all throughout the file we will refer to rotating as the rotation around the z-axis. The other deformation will be refered to as bending
            ## (imagine the plane with constant z (inflow plane) being bent into a plane with constant r (outflow plane), the corner would be the palce where the compressor is)
            ## The bending of the plane will give the point the correct r and z coordinates to be in the slice. The rotation afterwards will put it at the correct angle.
            ## Both "motions" are done by taking the appropriate distance between p1 and the u_ogl_point and moving along the surface: sqrt(dr^2 + dz^2)  gives the distance for the bending motion


            #ogl_points.append(ogl_point)
            ogl_points.append(reposition_into_plane(u_ogl_point,p1,i))

            #print "line =", n,   "point=", i
        #now create the connections with the camber line:
        #we start with the inlet and we are workingwith the first point of the camber line:

        wp = camber_curves[n].eval(0)   #working point
        wp_r = get_radius(wp)
        wp_theta = arctan2(wp.y,wp.x)
        wp_z=wp.z


        # a lot of work to put the first ogl point into the right plane again : (which is the work we done before backwards and for a different plane)
        foglp = ogl_points[0]
        foglp_r = get_radius(foglp)
        foglp_theta = arctan2(foglp.y,foglp.x)
        foglp_z= foglp.z

```

```
        rotation_arcl = (foglp_theta-wp_theta)*foglp_r        #full circle theta = 2 pi and so the arc would be 2*Pi*r
        #print "rotation arcl = ", rotation_arcl ," dist =", vabs(wp-foglp)

        dp=wp-camber_curves[n].eval(1/float(number_of_points_for_ogl))

        ogl_starting_point = wp+Vector(dp.x,dp.y,0).norm()*distance_ogl_blade

        if foglp_r>=wp_r:        #if this is the case the points are already in the same plane.
            Arc_connection = Arc(foglp,ogl_starting_point,wp)
            for ii in range (0,11):
                ogl_points.insert(0,Arc_connection.eval(ii*0.1))
            #print"____________!_!_!_!_!_!_!_!_!_!_!_", n
        else:
            a = 0
            b=0
            walker = [wp,Vector(0,0,0)]
            dw = 0
            while a >= 0:
                walker[1] = camber_curves[n].eval(b*arc_resolution)
                dw += vabs(walker[0] - walker[1])
                a=get_radius(walker[1])-foglp_r
                b +=1
                walker[0] = walker[1].clone()

            foglp_in_the_right_plane = wp-Vector(dp.x,dp.y,0).norm()*dw+Vector(-dp.y,dp.x,0).norm()*rotation_arcl
            #print"dw=",dw, "rotation_arcl", rotation_arcl ,"sqrt=" ,sqrt(dw**2+rotation_arcl**2)
            #print "distance new point-wp=",vabs(foglp_in_the_right_plane-wp)," old distance =", vabs(wp-foglp)
            x_vector = ogl_starting_point-wp
            scewed_y_vec = foglp_in_the_right_plane-wp


            for ii in range (0,11):
                point_on_connecting_arc = sin(pi/2/10*ii)*x_vector+cos(pi/2/10*ii)*scewed_y_vec+wp
                repositioned = reposition_into_plane(point_on_connecting_arc,wp,0)
                ogl_points.insert(0,repositioned)
                #print "r=", get_radius(repositioned) , "                    r=", get_radius(foglp)

        #now we have to connect them in the outflow

        wp2 = camber_curves[n].eval(1)    #working point at the end
        wp2_r = get_radius(wp2)
        wp2_theta = arctan2(wp2.y,wp2.x)
        wp2_z=wp2.z


        # a lot of work to put the first ogl point into the right plane again : (which is the same thing we did before backwards and for a different plane)
        loglp = ogl_points[-1]
        loglp_r = get_radius(loglp)
        loglp_theta = arctan2(loglp.y,loglp.x)
        loglp_z= loglp.z

                #full circle theta = 2 pi and so the arc would be 2*Pi*r
        #print "rotation arcl = ", rotation_arcl ," dist =", vabs(wp-foglp)
        dp2=wp2-camber_curves[n].eval(1-1/float(number_of_points_for_ogl))


        arc_arm1 = dp2.norm()*distance_ogl_blade
        arc_arm2 = cross(dp2,Vector(cos(wp2_theta),sin(wp2_theta),0)*pressure_or_suction).norm()*distance_ogl_blade
        #arc_arm2 = (loglp_in_the_working_plane-wp2).norm()*distance_ogl_blade

        starting_angle = arccos(dot(ogl_points[-1]-wp2,dp2)/vabs(ogl_points[-1]-wp2)/vabs(dp2))
        #print "starting_angle", starting_angle/pi*180 , "first_angle" , (pi/2-starting_angle)*180/pi
        l = 10
        for iii in range (4,l+1):
            #point_on_connecting_arc2 = sin(pi/2/l*iii)*arc_arm1+cos(pi/2/l*iii)*arc_arm2+wp2
```

```
            point_on_connecting_arc2 = sin(pi/2-starting_angle+starting_angle/l*iii)*arc_arm1+cos(pi/2-starting_angle+starting_angle/l*iii)*arc_arm2+wp2
            repositioned2 = reposition_into_plane(point_on_connecting_arc2,wp2,float(number_of_points_for_ogl))          #lekker handig
            ogl_points.append(repositioned2)
            #ogl_points.append(point_on_connecting_arc2)




        #make a spline of all those points:
        oglines.append(Spline(ogl_points))
        print "%g out of %g" %(n+1,len(input_curves))


    return oglines

###
##print "-------------------------------------------------------------------------------"

#
input_curves_pressure = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud,number_of_intermediate_camber_files)
input_curves_suction = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud,number_of_intermediate_camber_files)
input_curves_camber = input_interpreter(file_camber_hub,file_camber_hub_to_shroud,file_camber_shroud,number_of_intermediate_camber_files)


O_grid_line_surface_pressure = PyFunctionSurface(Loft(ogl_creator(input_curves_pressure,input_curves_camber,1)))
O_grid_line_surface_suction = PyFunctionSurface(Loft(ogl_creator(input_curves_suction,input_curves_camber,-1)))
#
### all the main surfaces are there, now we need to connect them in the right places to create blocks
### first functions are created to make the different kind of blocks.. blocks in the o grid part, blocks in the H - block part
### and blocks in the inflow and outflow part... all automated depending on the number of nodes
#
#########################################################################################################
#########################################################################################################

#
#
def acurate_length(curve):
    ### A more acurate way to obtain length
    L = 0.0
    n = 100
    dt = 1.0 / n
    p0 = curve.eval(0.0)
    p1 = Vector(0,0,0)
    for i in range (1,n+1):
        p1 = curve.eval(dt * i)
        L += vabs(p1 - p0)
        p0 = p1;
    return L


def simple_connector(p1,p2,locationp1,curvep1,camber_curves,top_or_bottom):  #top_or_bottom 0 = bot  or -1 = top
    ### This function is used to connect nodes

    r1 = get_radius(p1)
    z1 = p1.z
    th1 = arctan2(p1.y,p1.x)
    r2 = get_radius(p2)
    z2 = p2.z
    th2 = arctan2(p2.y,p2.x)

```

```
LE = camber_curves[top_or_bottom].eval(0)
TE = camber_curves[top_or_bottom].eval(1)
r_max = get_radius(LE)
z_max = TE.z

def walking_along_the_curve(p1,p2,s):    #S = WALKING DIRECTION
    z1 = p1.z
    z2 = p2.z
    r1 = get_radius(p1)
    r2 = get_radius(p2)
    th1 = arctan2(p1.y,p1.x)
    th2 = arctan2(p2.y,p2.x)
    dw = 0
    dzw = 0
    walker = [p1,Vector(0,0,0)]
    points_on_connection = []
    n = number_of_points_on_connection + 1
    a = s
    distance = sqrt((r2-r1)**2+(z2-z1)**2)
    for i in range(1,n):
        while dw < distance/n*i:
        #while abs(dzw) < abs((z2-z1)/n*i):
            walker[1] = curvep1.eval(locationp1+a*arc_resolution)
            #dzw += (walker[1].z-walker[0].z)**2 + (get_radius(walker[1])-get_radius(walker[0]))**2
            dzw = walker[1].z-walker[0].z
            drw = get_radius(walker[1])-get_radius(walker[0])
            dw += sqrt(dzw**2 + drw**2)
            a += s
            #print "locationp1+a*arc_resolution", locationp1+a*arc_resolution ,"dzleft" , abs(distance)-abs(dw) , "zwalker" , walker[1].z
            #print "z2" , z2
            if locationp1+a*arc_resolution < 0:
                #print "entered this part when i is" , i , "and a" , a ,
                dzw = (z2-z1)*s
                dw = distance
                walker[1] = curvep1.eval(0)
            elif locationp1+a*arc_resolution > 1:
                #print "entered this other part when i is" , i , "and a" , a ,
                dw= distance
                dzw = (z2-z1)*s
                walker[1] = curvep1.eval(1)
            rr = get_radius(walker[1])
            zz = walker[1].z
            walker[0] = walker[1].clone()
        thth= th1+(th2-th1)/n*i
        points_on_connection.append(Vector(rr*cos(thth),rr*sin(thth),zz))
    return points_on_connection

def walking_along_the_camber(p1,p2,s,start,top_or_bottom):     #start 0 or 1, top_or_bottom 0 = bot  or -1 = top
    z1 = p1.z
    z2 = p2.z
    r1 = get_radius(p1)
    r2 = get_radius(p2)
    th1 = arctan2(p1.y,p1.x)
    th2 = arctan2(p2.y,p2.x)
    dw = 0
    dzw = 0
    walker = [p1,Vector(0,0,0)]
    points_on_connection = []
    n = number_of_points_on_connection + 1
    a = s
    distance = sqrt((r2-r1)**2+(z2-z1)**2)
    for i in range(1,n):
        #print "dw", dw , "distance/n*i", distance/n*i, "distance", distance, "n", n ,"i", i
        while dw < distance/n*i:
        #while abs(dzw) < abs((z2-z1)/n*i):
            walker[1] = camber_curves[top_or_bottom].eval(start+a*arc_resolution)
            #dzw += (walker[1].z-walker[0].z)**2 + (get_radius(walker[1])-get_radius(walker[0]))**2
            dzw = walker[1].z-walker[0].z
            drw = get_radius(walker[1])-get_radius(walker[0])
            dw += sqrt(dzw**2 + drw**2)
```

```
                a += s
                #print "locationp1+a*arc_resolution", locationp1+a*arc_resolution ,"dzleft" , abs(distance)-abs(dw) , "zwalker" , walker[1].z
                #print "z2" , z2
                if start+a*arc_resolution < 0:
                    #print "entered this part when i is" , i , "and a" , a ,
                    dzw = (z2-z1)*s
                    dw = distance
                    walker[1] = curvep1.eval(0)
                elif start+a*arc_resolution > 1:
                    #print "entered this other part when i is" , i , "and a" , a ,
                    dw= distance
                    dzw = (z2-z1)*s
                    walker[1] = curvep1.eval(1)
                rr = get_radius(walker[1])
                zz = walker[1].z
                walker[0] = walker[1].clone()
            thth= th1+(th2-th1)/n*i
            points_on_connection.append(Vector(rr*cos(thth),rr*sin(thth),zz))
        return points_on_connection

    #print "p1", p1 , "p2" ,p2
    #print "r1",r1 ,"r2", r2, "rmax", r_max, "z_max" , z_max ,"z1" ,z1 , "z2",z2
    if z1 <= z_max:
        if z2 <= z_max:
            if r1 <= r_max:
                if r2 <= r_max: #both are in bent area
                    #print" we made it here" , "r2" ,r2 , "r_max" , r_max
                    if r1 == r2:
                        connection = Arc(p1,p2,Vector(0,0,p1.z))
                    else:
                        if r1>r2 :
                            s=1
                        else :
                            s = -1
                        points = walking_along_the_curve(p1,p2,s)
                        points.insert(0,p1)
                        points.append(p2)
                        connection = Spline(points)
                else:#now p2 is in inlet area p1 is not therefore r1<r2
                    s=-1
                    LE_angle= arctan2(LE.y,LE.x)
                    point_at_correct_angle = LE.clone().rotate_about_zaxis((th2-th1)*(r_max-r1)/(r2-r1)+(th1-LE_angle))
                    #point_at_correct_angle = LE.clone().rotate_about_zaxis((th2-LE_angle))
                    point_in_between = Vector(point_at_correct_angle.x,point_at_correct_angle.y,z2)
                    #point_in_between = rotate_point_around_z_axis(LE,(th2-LE_angle)*6)
                    points = walking_along_the_curve(p1,point_in_between,s)
                    #points = [point_in_between]
                    points.insert(0,p1)
                    points.append(point_in_between)
                    #print "factor ", (r_max-r1)/(r2-r1) , "th2" , th2*180/pi, th1*180/pi , LE_angle*180/pi
                    #print "point in betwe", arctan2(point_in_between.y,point_in_between.x)*180/pi
                    points.append(p2)
                    connection = Spline(points)
            elif r2 < r_max:
                s = 1
                LE_angle= arctan2(LE.y,LE.x)
                point_at_correct_angle = LE.clone().rotate_about_zaxis((th2-th1)*(r_max-r1)/(r2-r1)+(th1-LE_angle))
                point_in_between = Vector(point_at_correct_angle.x,point_at_correct_angle.y,z1)
                points = walking_along_the_camber(point_in_between,p2,s,0,top_or_bottom)
                points.insert(0,point_in_between)
                points.insert(0,p1*1/3+point_in_between*2/3)     #SPLINE BEHAVES WEIRD WITHOUT THIS AT p1curve = PathOnSurface( O_grid_line_surface_pressure , LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))  p1 =p1curve.eval(0.01)  p2 = c45.eval(0.01)
                points.insert(0,p1)
                points.append(p2)
                connection = Spline(points)
            else:
                connection = Spline([p1,p2])
```

```
        else:
            s=1
            TE_angle= arctan2(TE.y,TE.x)
            point_at_correct_angle = TE.clone().rotate_about_zaxis((th2-th1)*(z_max-z1)/(z2-z1)+(th1-TE_angle))
            point_in_between = Vector(point_at_correct_angle.x,point_at_correct_angle.y,TE.z)
            points = walking_along_the_curve(p1,point_in_between,s)
            #points = [point_in_between]
            points.insert(0,p1)
            points.append(point_in_between)
            #print "factor "
            #print "point in betwe"
            points.append(p2)
            connection = Spline(points)
            #print "p1 in between, p2 in outflow"
    elif z2 < z_max:
        s = -1
        TE_angle= arctan2(TE.y,TE.x)
        point_at_correct_angle = TE.clone().rotate_about_zaxis((th2-th1)*(z_max-z1)/(z2-z1)+(th1-TE_angle))
        point_in_between = Vector(point_at_correct_angle.x,point_at_correct_angle.y,TE.z)
        points = walking_along_the_camber(point_in_between,p2,s,1,top_or_bottom)
        points.insert(0,point_in_between)
        points.insert(0,p1)
        points.append(p2)
        connection = Spline(points)
        #print "p1 in outflow, p2 in bewteen"
    else:
        points = []
        n = number_of_points_on_connection +1
        for i in range(1,n):
            th_in_between = th1*(1-i/float(n))+th2*i/n
            r_in_between = r1*(1-i/float(n))+r2*i/n
            x = cos(th_in_between) * r_in_between #r1 should be equal to r2 but its only close .. so this will but them closer together
            y = sin(th_in_between) * r_in_between
            z = z1*(1-i/float(n))+z2*i/n
            points.append(Vector(x,y,z).clone())
        points.insert(0,p1)
        points.append(p2)
        connection = Spline(points)
        #print "p1 in outflow, p2 in outflow"
    return connection



camber_curves = input_curves # I SHOULD PUT THIS IN FRONT OF EVERYTHING
blade_pressure_curves = input_curves
### Here the three parts of the surface on a periodic boundary are joined together.

def surface_connector_perodic_pressure(Surf1,Surf2,Surf3):
    def true_surface(r,s):
        p1 = PathOnSurface(Surf1,LinearFunction(1.0,0.0), LinearFunction(0.0,float(s)))
        p2 = PathOnSurface(Surf1,LinearFunction(1.0,0.0), LinearFunction(0.0,float(s)))
        p3 = PathOnSurface(Surf1,LinearFunction(1.0,0.0), LinearFunction(0.0,float(s)))
        length_p1 = acurate_length(p1)
        length_p2 = acurate_length(p2)
        length_p3 = acurate_length(p3)
        total_p = length_p1+length_p2+length_p3
        if r <= length_p1/total_p:
            p = Surf1.eval(r*total_p/length_p1,s)
        elif r < (length_p1+length_p2)/total_p:
            r_coordinate = (r-length_p1/total_p)*total_p/length_p2
            p = Surf2.eval(r_coordinate,s)
        else:
            r_coordinate = (r-(length_p1+length_p2)/total_p)*total_p/length_p3
            p = Surf3.eval(r_coordinate,s)
        x = p.x
        y = p.y
        z = p.z
        return (x,y,z)
    return true_surface

pressure_periodic = PyFunctionSurface(surface_connector_perodic_pressure(v6south,south,v2south))
```

```
suction_periodic = PyFunctionSurface(surface_connector_perodic_pressure(v5north,v4north,v3north))

### Function to create blocks in the H-Block part

def create_H_Block(nodes_on_periodic_bondary,nodes_on_O_gridline,pressure_or_suction):  # example : create_H_Block([0.5,0.75],[0.4,0.66],1)
    if pressure_or_suction > 0 :
        def H_south_python(nodes):
            def H_s(r,s):
                r_coordinate = (nodes[1]-nodes[0])*r + nodes[0]
                #print "r_coordinate",r_coordinate , "nodes[0]", nodes[0] , "nodes[1]", nodes[1]
                point = pressure_periodic.eval(r_coordinate,s)
                x = point.x
                y = point.y
                z = point.z
                return (x,y,z)
            return H_s
        H_south = PyFunctionSurface(H_south_python(nodes_on_periodic_bondary))
        H_c01 = PathOnSurface (H_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        H_c15 = PathOnSurface (H_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        H_c04 = PathOnSurface (H_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        H_c45 = PathOnSurface (H_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_south +=================================================+"

        def H_north_python(r,s):
            r_coordinate = (nodes_on_O_gridline[1]-nodes_on_O_gridline[0])*r + nodes_on_O_gridline[0]
            point = O_grid_line_surface_pressure.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        H_north = PyFunctionSurface(H_north_python)
        H_c32 = PathOnSurface (H_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        H_c26 = PathOnSurface (H_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        H_c37 = PathOnSurface (H_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        H_c76 = PathOnSurface (H_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_north +=================================================+"

        H_p4 = H_c45.eval(0)
        H_p5 = H_c45.eval(1)
        H_p7 = H_c76.eval(0)
        H_p6 = H_c76.eval(1)

        H_p0 = H_c01.eval(0)
        H_p1 = H_c01.eval(1)
        H_p3 = H_c32.eval(0)
        H_p2 = H_c32.eval(1)

        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        H_c47 = simple_connector(H_p7,H_p4,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        H_c56 = simple_connector(H_p6,H_p5,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        H_c03 = simple_connector(H_p3,H_p0,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        H_c12 = simple_connector(H_p2,H_p1,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)

        H_c47.reverse()
        H_c56.reverse()
        H_c03.reverse()
        H_c12.reverse()

        p0 = H_c03.eval(0)
        p1 = H_c12.eval(0)
        p2 = H_c12.eval(1)
```

```
        p3 = H_c03.eval(1)
        p4 = H_c47.eval(0)
        p5 = H_c56.eval(0)
        p6 = H_c56.eval(1)
        p7 = H_c47.eval(1)

        #p0 = H_c01.eval(0)
        #p1 = H_c01.eval(1)
        #p2 = H_c32.eval(1)
        #p3 = H_c32.eval(0)
        #p4 = H_c45.eval(0)
        #p5 = H_c45.eval(1)
        #p6 = H_c76.eval(1)
        #p7 = H_c76.eval(0)

        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        H_top = CoonsPatch(H_c45,H_c76,H_c47,H_c56)
        #print "H_top+===========================================+"
        #
        H_bottom = CoonsPatch(H_c01,H_c32,H_c03,H_c12)
        #print "H_bottom+===========================================+"

        H_east = CoonsPatch(H_c12,H_c56,H_c15,H_c26)
        #print "H_east+===========================================+"

        H_west = CoonsPatch(H_c03,H_c47,H_c04,H_c37)
        #print "H_west +===========================================+"

        H_volume = ParametricVolume(H_north,H_east,H_south,H_west,H_top,H_bottom)

        #A=[]
        A.append(H_c01)
        A.append(H_c32)
        #A.append(H_c12)
        A.append(H_c03)
        A.append(H_c45)
        #A.append(H_c56)
        A.append(H_c76)
        A.append(H_c47)
        #A.append(H_c26)
        #A.append(H_c15)
        A.append(H_c04)
        A.append(H_c37)


    elif pressure_or_suction < 0:
        def H_south_python(r,s):
            r_coordinate = (nodes_on_O_gridline[1]-nodes_on_O_gridline[0])*r + nodes_on_O_gridline[0]
            point = O_grid_line_surface_suction.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)
        H_south = PyFunctionSurface(H_south_python)

        H_c01 = PathOnSurface (H_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        H_c15 = PathOnSurface (H_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        H_c04 = PathOnSurface (H_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        H_c45 = PathOnSurface (H_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_south +===========================================+"

        def H_north_python(nodes):
            def H_n(r,s):
                r_coordinate = (nodes[1]-nodes[0])*r + nodes[0]
                #print "r_coordinate",r_coordinate , "nodes[0]", nodes[0] , "nodes[1]", nodes[1]
                point = suction_periodic.eval(r_coordinate,s)
                x = point.x
                y = point.y
                z = point.z
                return (x,y,z)
            return H_n
        H_north = PyFunctionSurface(H_north_python(nodes_on_periodic_bondary))
```

```

        H_c32 = PathOnSurface (H_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        H_c26 = PathOnSurface (H_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        H_c37 = PathOnSurface (H_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        H_c76 = PathOnSurface (H_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_north +==================================================+"

        H_p4 = H_c45.eval(0)
        H_p5 = H_c45.eval(1)
        H_p7 = H_c76.eval(0)
        H_p6 = H_c76.eval(1)

        H_p0 = H_c01.eval(0)
        H_p1 = H_c01.eval(1)
        H_p3 = H_c32.eval(0)
        H_p2 = H_c32.eval(1)

        Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        H_c47 = simple_connector(H_p4,H_p7,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        H_c56 = simple_connector(H_p5,H_p6,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        H_c03 = simple_connector(H_p0,H_p3,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        H_c12 = simple_connector(H_p1,H_p2,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)


        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        H_top = CoonsPatch(H_c45,H_c76,H_c47,H_c56)
        #print "H_top+==================================================+"
        #
        H_bottom = CoonsPatch(H_c01,H_c32,H_c03,H_c12)
        #print "H_bottom+==================================================+"

        H_east = CoonsPatch(H_c12,H_c56,H_c15,H_c26)
        #print "H_east+==================================================+"

        H_west = CoonsPatch(H_c03,H_c47,H_c04,H_c37)
        #print "H_west +==================================================+"

        H_volume = ParametricVolume(H_north,H_east,H_south,H_west,H_top,H_bottom)

        #A=[]
        A.append(H_c01)
        A.append(H_c32)
        A.append(H_c12)
        #A.append(H_c03)
        A.append(H_c45)
        A.append(H_c56)
        A.append(H_c76)
        #A.append(H_c47)
        A.append(H_c26)
        A.append(H_c15)
        #A.append(H_c04)
        #A.append(H_c37)
        #for n in range(0,len(A)):
        #     string = "curve_file%g.txt" %n
        #     print string , n , " of " , len(A)
        #     with open(string,"w") as thisfile:
        #         thisfile.write("")
        #     with open(string,"a") as thisfile:
        #         for i in range (0,101):
        #             B = A[n].eval(i*0.01)
        #             C = "%g %g %g \n" %(B.x,B.y,B.z)
        #             #print C
        #             thisfile.write(C)
    #H_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
    #       parametric_volume=H_volume,
    #       fill_condition=initialCond))
```

```
    return "hello"
    #return H_block
#print create_H_Block([0.3,0.4],[0.3,0.4],-1)

#first_H_block_ever = create_H_Block([0.3,0.4],[0.3,0.4],1)


#### Here a function is created to create blocks in the O-grid part

def create_O_Block(nodes_on_O_gridline,nodes_on_blade,pressure_or_suction):  #example : create_O_Block([0.5,0.75],[0.4,0.66],1)
    if pressure_or_suction > 0 :
        def O_south_python(nodes):
            def O_s(r,s):
                r_coordinate = (nodes[1]-nodes[0])*r + nodes[0]
                point = O_grid_line_surface_pressure.eval(r_coordinate,s)
                x = point.x
                y = point.y
                z = point.z
                return (x,y,z)
            return O_s
        O_south = PyFunctionSurface(O_south_python(nodes_on_O_gridline))
        O_c01 = PathOnSurface (O_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        O_c15 = PathOnSurface (O_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        O_c04 = PathOnSurface (O_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        O_c45 = PathOnSurface (O_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_south +==============================================+"

        def O_north_python(r,s):
            r_coordinate = (nodes_on_blade[1]-nodes_on_blade[0])*r + nodes_on_blade[0]
            point = north.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        O_north = PyFunctionSurface(O_north_python)
        O_c32 = PathOnSurface (O_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        O_c26 = PathOnSurface (O_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        O_c37 = PathOnSurface (O_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        O_c76 = PathOnSurface (O_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_north +==============================================+"

        O_p4 = O_c45.eval(0)
        O_p5 = O_c45.eval(1)
        O_p7 = O_c76.eval(0)
        O_p6 = O_c76.eval(1)

        O_p0 = O_c01.eval(0)
        O_p1 = O_c01.eval(1)
        O_p3 = O_c32.eval(0)
        O_p2 = O_c32.eval(1)
        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        #
        O_c47 = simple_connector(O_p4,O_p7,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        O_c56 = simple_connector(O_p5,O_p6,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        O_c03 = simple_connector(O_p0,O_p3,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        O_c12 = simple_connector(O_p1,O_p2,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)
        #
        #
        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        O_top = CoonsPatch(O_c45,O_c76,O_c47,O_c56)
        #print "O_top+==============================================+"
        #
```

```
        O_bottom = CoonsPatch(O_c01,O_c32,O_c03,O_c12)
        #print "O_bottom+==================================================+"

        O_east = CoonsPatch(O_c12,O_c56,O_c15,O_c26)
        #print "O_east+====================================================+"

        O_west = CoonsPatch(O_c03,O_c47,O_c04,O_c37)
        #print "O_west +===================================================+"

        O_volume = ParametricVolume(O_north,O_east,O_south,O_west,O_top,O_bottom)

        #A=[]
        A.append(O_c01)
        A.append(O_c32)
        A.append(O_c12)
        A.append(O_c03)
        A.append(O_c45)
        A.append(O_c56)
        A.append(O_c76)
        A.append(O_c47)
        A.append(O_c26)
        A.append(O_c15)
        A.append(O_c04)
        A.append(O_c37)
        ##for n in range(0,len(A)):
        ##     string = "curve_file%g.txt" %n
        ##     print string , n
        ##     with open(string,"w") as thisfile:
        ##         thisfile.write("")
        ##     with open(string,"a") as thisfile:
        ##         for i in range (0,101):
        ##             B = A[n].eval(i*0.01)
        ##             C = "%g %g %g \n" %(B.x,B.y,B.z)
        ##             #print C
        ##             thisfile.write(C)

    elif pressure_or_suction < 0:
        def O_north_python(nodes):
            def O_n(r,s):
                r_coordinate = (nodes[1]-nodes[0])*r + nodes[0]
                point = O_grid_line_surface_suction.eval(r_coordinate,s)
                x = point.x
                y = point.y
                z = point.z
                return (x,y,z)
            return O_n
        O_north = PyFunctionSurface(O_north_python(nodes_on_O_gridline))
        O_c32 = PathOnSurface (O_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        O_c26 = PathOnSurface (O_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        O_c37 = PathOnSurface (O_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        O_c76 = PathOnSurface (O_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_north +==================================================+"

        def O_south_python(r,s):
            r_coordinate = (nodes_on_blade[1]-nodes_on_blade[0])*r + nodes_on_blade[0]
            point = v4south.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        O_south = PyFunctionSurface(O_south_python)
        O_c01 = PathOnSurface (O_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        O_c15 = PathOnSurface (O_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        O_c04 = PathOnSurface (O_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        O_c45 = PathOnSurface (O_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        #print "H_south +==================================================+"

        O_p4 = O_c45.eval(0)
        O_p5 = O_c45.eval(1)
        O_p7 = O_c76.eval(0)
        O_p6 = O_c76.eval(1)

        O_p0 = O_c01.eval(0)
        O_p1 = O_c01.eval(1)
```

```
        O_p3 = O_c32.eval(0)
        O_p2 = O_c32.eval(1)
        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction , LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction , LinearFunction(1.0,0.0) , LinearFunction(0.0,0.0))
        #
        O_c47 = simple_connector(O_p7,O_p4,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        O_c56 = simple_connector(O_p6,O_p5,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        O_c03 = simple_connector(O_p3,O_p0,nodes_on_O_gridline[0],Ogridcurve_bottom, camber_curves,0)
        O_c12 = simple_connector(O_p2,O_p1,nodes_on_O_gridline[1],Ogridcurve_bottom, camber_curves,0)
        O_c47.reverse()
        O_c56.reverse()
        O_c03.reverse()
        O_c12.reverse()
        #
        #
        #print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        O_top = CoonsPatch(O_c45,O_c76,O_c47,O_c56)
        #print "O_top+==============================================================+"
        #
        O_bottom = CoonsPatch(O_c01,O_c32,O_c03,O_c12)
        #print "O_bottom+==============================================================+"

        O_east = CoonsPatch(O_c12,O_c56,O_c15,O_c26)
        #print "O_east+==============================================================+"

        O_west = CoonsPatch(O_c03,O_c47,O_c04,O_c37)
        #print "O_west +==============================================================+"

        O_volume = ParametricVolume(O_north,O_east,O_south,O_west,O_top,O_bottom)

        #A=[]
        A.append(O_c01)
        A.append(O_c32)
        A.append(O_c12)
        A.append(O_c03)
        A.append(O_c45)
        A.append(O_c56)
        A.append(O_c76)
        A.append(O_c47)
        A.append(O_c26)
        A.append(O_c15)
        A.append(O_c04)
        A.append(O_c37)
        #for n in range(0,len(A)):
        #    string = "curve_file%g.txt" %n
        #    print string , n
        #    with open(string,"w") as thisfile:
        #        thisfile.write("")
        #    with open(string,"a") as thisfile:
        #        for i in range (0,101):
        #            B = A[n].eval(i*0.01)
        #            C = "%g %g %g \n" %(B.x,B.y,B.z)
        #            #print C
        #            thisfile.write(C)


    #O_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
        #parametric_volume=O_volume,
        #fill_condition=initialCond))
    return "hello"
    #return O_block

#print create_O_Block([0.3,0.4],[0.3,0.4],-1)

#### Here a function is created to create blocks in the inlet region of the H-grid part
```

```

def create_In_Block(nodes_on_inflow_bondary,nodes_on_O_gridline,pressure_or_suction):  # example : create_In_Block([0.5,0.75],[0.4,0.66],1)
    if pressure_or_suction > 0 :
        In_p7 = v6c47.eval(nodes_on_inflow_bondary[1])
        In_p4 = v6c47.eval(nodes_on_inflow_bondary[0])
        In_p0 = v6c03.eval(nodes_on_inflow_bondary[0])
        In_p3 = v6c03.eval(nodes_on_inflow_bondary[1])

        In_c47 = Arc(In_p4,In_p7,Vector(0,0,p7.z))
        In_c03 = Arc(In_p0,In_p3,Vector(0,0,p0.z))
        In_c04 = Line(In_p0,In_p4)
        In_c37 = Line(In_p3,In_p7)

        In_west = CoonsPatch(In_c03,In_c47,In_c04,In_c37)
        #print "In_west +==============================================================+"

        def In_east_python(r,s):
            r_coordinate = (nodes_on_O_gridline[0]-nodes_on_O_gridline[1])*r + nodes_on_O_gridline[1]
            point = O_grid_line_surface_pressure.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        In_east = PyFunctionSurface(In_east_python)
        In_c12 = PathOnSurface (In_east, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        In_c56 = PathOnSurface (In_east, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        In_c15 = PathOnSurface (In_east, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        In_c26 = PathOnSurface (In_east, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        #print "In_east +==============================================================+"

        In_p1 = In_c12.eval(0)
        In_p2 = In_c12.eval(1)
        In_p5 = In_c56.eval(0)
        In_p6 = In_c56.eval(1)

        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        In_c45 = simple_connector(In_p5,In_p4,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        In_c76 = simple_connector(In_p6,In_p7,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        In_c01 = simple_connector(In_p1,In_p0,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        In_c32 = simple_connector(In_p2,In_p3,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)
        In_c45.reverse()
        In_c76.reverse()
        In_c01.reverse()
        In_c32.reverse()

        ##p0 = In_c03.eval(0)
        ##p1 = In_c12.eval(0)
        ##p2 = In_c12.eval(1)
        ##p3 = In_c03.eval(1)
        ##p4 = In_c47.eval(0)
        ##p5 = In_c56.eval(0)
        ##p6 = In_c56.eval(1)
        ##p7 = In_c47.eval(1)
        ##
        ##p0 = In_c01.eval(0)
        ##p1 = In_c01.eval(1)
        ##p2 = In_c32.eval(1)
        ##p3 = In_c32.eval(0)
        ##p4 = In_c45.eval(0)
        ##p5 = In_c45.eval(1)
        ##p6 = In_c76.eval(1)
        ##p7 = In_c76.eval(0)

```

```
        ##print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" ,
    p5 ,"\n p6",p6 ,"\n p7" ,p7
        In_top = CoonsPatch(In_c45,In_c76,In_c47,In_c56)
        #print "In_top+==================================================+"
        #
        In_bottom = CoonsPatch(In_c01,In_c32,In_c03,In_c12)
        #print "In_bottom+==================================================+"

        In_north = CoonsPatch(In_c32,In_c76,In_c37,In_c26)
        #print "In_north+==================================================+"

        In_south = CoonsPatch(In_c01,In_c45,In_c04,In_c15)
        #print "In_south+==================================================+"

        In_volume = ParametricVolume(In_north,In_east,In_south,In_west,In_top,In_bottom)

        #A=[]
        #A.append(In_c01)
        A.append(In_c32)
        A.append(In_c12)
        A.append(In_c03)
        #A.append(In_c45)
        A.append(In_c56)
        A.append(In_c76)
        A.append(In_c47)
        A.append(In_c26)
        #A.append(In_c15)
        #A.append(In_c04)
        A.append(In_c37)
        #for n in range(0,len(A)):
        #    string = "curve_file%g.txt" %n
        #    print string , n
        #    with open(string,"w") as thisfile:
        #        thisfile.write("")
        #    with open(string,"a") as thisfile:
        #        for i in range (0,101):
        #            B = A[n].eval(i*0.01)
        #            C = "%g %g %g \n" %(B.x,B.y,B.z)
        #            #print C
        #            thisfile.write(C)

    elif pressure_or_suction < 0:
        In_p7 = v5c47.eval(nodes_on_inflow_bondary[1])
        In_p4 = v5c47.eval(nodes_on_inflow_bondary[0])
        In_p0 = v5c03.eval(nodes_on_inflow_bondary[0])
        In_p3 = v5c03.eval(nodes_on_inflow_bondary[1])

        In_c47 = Arc(In_p4,In_p7,Vector(0,0,p7.z))
        In_c03 = Arc(In_p0,In_p3,Vector(0,0,p0.z))
        In_c04 = Line(In_p0,In_p4)
        In_c37 = Line(In_p3,In_p7)

        In_west = CoonsPatch(In_c03,In_c47,In_c04,In_c37)
        #print "In_west +==================================================+"

        def In_east_python(r,s):
            r_coordinate = (nodes_on_O_gridline[1]-nodes_on_O_gridline[0])*r +
    nodes_on_O_gridline[0]
            point = O_grid_line_surface_suction.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        In_east = PyFunctionSurface(In_east_python)
        In_c12 = PathOnSurface (In_east, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        In_c56 = PathOnSurface (In_east, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        In_c15 = PathOnSurface (In_east, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        In_c26 = PathOnSurface (In_east, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        #print "In_east +==================================================+"

        In_p1 = In_c12.eval(0)
        In_p2 = In_c12.eval(1)
        In_p5 = In_c56.eval(0)
        In_p6 = In_c56.eval(1)
```

```

        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        In_c45 = simple_connector(In_p5,In_p4,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        In_c76 = simple_connector(In_p6,In_p7,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        In_c01 = simple_connector(In_p1,In_p0,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        In_c32 = simple_connector(In_p2,In_p3,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)
        In_c45.reverse()
        In_c76.reverse()
        In_c01.reverse()
        In_c32.reverse()

        ##p0 = In_c03.eval(0)
        ##p1 = In_c12.eval(0)
        ##p2 = In_c12.eval(1)
        ##p3 = In_c03.eval(1)
        ##p4 = In_c47.eval(0)
        ##p5 = In_c56.eval(0)
        ##p6 = In_c56.eval(1)
        ##p7 = In_c47.eval(1)
        ##
        ##p0 = In_c01.eval(0)
        ##p1 = In_c01.eval(1)
        ##p2 = In_c32.eval(1)
        ##p3 = In_c32.eval(0)
        ##p4 = In_c45.eval(0)
        ##p5 = In_c45.eval(1)
        ##p6 = In_c76.eval(1)
        ##p7 = In_c76.eval(0)

        ##print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        In_top = CoonsPatch(In_c45,In_c76,In_c47,In_c56)
        #print "In_top+==================================================+"
        #
        In_bottom = CoonsPatch(In_c01,In_c32,In_c03,In_c12)
        #print "In_bottom+==================================================+"

        In_north = CoonsPatch(In_c32,In_c76,In_c37,In_c26)
        #print "In_north+==================================================+"

        In_south = CoonsPatch(In_c01,In_c45,In_c04,In_c15)
        #print "In_south+==================================================+"

        In_volume = ParametricVolume(In_north,In_east,In_south,In_west,In_top,In_bottom)

        #A=[]
        A.append(In_c01)
        #A.append(In_c32)
        A.append(In_c12)
        A.append(In_c03)
        A.append(In_c45)
        A.append(In_c56)
        #A.append(In_c76)
        A.append(In_c47)
        #A.append(In_c26)
        A.append(In_c15)
        A.append(In_c04)
        #A.append(In_c37)
        #for n in range(0,len(A)):
        #    string = "curve_file%g.txt" %n
        #    print string , n
        #    with open(string,"w") as thisfile:
        #        thisfile.write("")
        #    with open(string,"a") as thisfile:
        #        for i in range (0,101):
        #            B = A[n].eval(i*0.01)
```

```
        #               C = "%g %g %g \n" %(B.x,B.y,B.z)
        #               #print C
        #               thisfile.write(C)

    #In_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
    #       parametric_volume=In_volume,
    #       fill_condition=initialCond))
    return "hello"
    #return In_block

#print create_In_Block([0.3,0.4],[0.02,0.025],-1)

#### Here a function is created to create blocks in the outlet region of the H-grid part

def create_Out_Block(nodes_on_outflow_bondary,nodes_on_O_gridline,pressure_or_suction):  # example : create_H_Block([0.5,0.75],[0.4,0.66],1)
    if pressure_or_suction > 0 :
        Out_p6 = v2c56.eval(nodes_on_outflow_bondary[1])
        Out_p5 = v2c56.eval(nodes_on_outflow_bondary[0])
        Out_p1 = v2c12.eval(nodes_on_outflow_bondary[0])
        Out_p2 = v2c12.eval(nodes_on_outflow_bondary[1])

        Out_c56 = Arc(Out_p5,Out_p6,Vector(0,0,p5.z))
        Out_c12 = Arc(Out_p1,Out_p2,Vector(0,0,p1.z))
        Out_c26 = Line(Out_p2,Out_p6)
        Out_c15 = Line(Out_p1,Out_p5)

        Out_east = CoonsPatch(Out_c12,Out_c56,Out_c15,Out_c26)
        #print "Out_east +==================================================+"

        def Out_west_python(r,s):
            r_coordinate = (nodes_on_O_gridline[1]-nodes_on_O_gridline[0])*r + nodes_on_O_gridline[0]
            point = O_grid_line_surface_pressure.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        Out_west = PyFunctionSurface(Out_west_python)
        Out_c47 = PathOnSurface (Out_west, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Out_c37 = PathOnSurface (Out_west, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        Out_c04 = PathOnSurface (Out_west, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        Out_c03 = PathOnSurface (Out_west, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        #print "Out_west +==================================================+"

        Out_p4 = Out_c47.eval(0)
        Out_p7 = Out_c47.eval(1)
        Out_p0 = Out_c03.eval(0)
        Out_p3 = Out_c03.eval(1)

        #if pressure_or_suction > 0:
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        Out_c45 = simple_connector(Out_p4,Out_p5,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        Out_c76 = simple_connector(Out_p7,Out_p6,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        Out_c01 = simple_connector(Out_p0,Out_p1,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        Out_c32 = simple_connector(Out_p3,Out_p2,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)
        #Out_c45.reverse()
        #Out_c76.reverse()
        #Out_c01.reverse()
        #Out_c32.reverse()


        ##print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        Out_top = CoonsPatch(Out_c45,Out_c76,Out_c47,Out_c56)
        #print "Out_top+==================================================+"
```

```
        #
        Out_bottom = CoonsPatch(Out_c01,Out_c32,Out_c03,Out_c12)
        #print "Out_bottom+=================================================+"

        Out_north = CoonsPatch(Out_c32,Out_c76,Out_c37,Out_c26)
        #print "Out_north+==================================================+"

        Out_south = CoonsPatch(Out_c01,Out_c45,Out_c04,Out_c15)
        #print "Out_south+==================================================+"

        Out_volume = ParametricVolume(Out_north,Out_east,Out_south,Out_west,Out_top,Out_bottom)

        #A=[]
        #A.append(Out_c01)
        A.append(Out_c32)
        A.append(Out_c12)
        A.append(Out_c03)
        #A.append(Out_c45)
        A.append(Out_c56)
        A.append(Out_c76)
        A.append(Out_c47)
        A.append(Out_c26)
        #A.append(Out_c15)
        #A.append(Out_c04)
        A.append(Out_c37)
        #for n in range(0,len(A)):
        #    string = "curve_file%g.txt" %n
        #    print string , n
        #    with open(string,"w") as thisfile:
        #        thisfile.write("")
        #    with open(string,"a") as thisfile:
        #        for i in range (0,101):
        #            B = A[n].eval(i*0.01)
        #            C = "%g %g %g \n" %(B.x,B.y,B.z)
        #            #print C
        #            thisfile.write(C)
        #
    elif pressure_or_suction < 0 :
        Out_p6 = v3c56.eval(nodes_on_outflow_bondary[1])
        Out_p5 = v3c56.eval(nodes_on_outflow_bondary[0])
        Out_p1 = v3c12.eval(nodes_on_outflow_bondary[0])
        Out_p2 = v3c12.eval(nodes_on_outflow_bondary[1])

        Out_c56 = Arc(Out_p5,Out_p6,Vector(0,0,p5.z))
        Out_c12 = Arc(Out_p1,Out_p2,Vector(0,0,p1.z))
        Out_c26 = Line(Out_p2,Out_p6)
        Out_c15 = Line(Out_p1,Out_p5)

        Out_east = CoonsPatch(Out_c12,Out_c56,Out_c15,Out_c26)
        #print "Out_east +==================================================+"

        def Out_west_python(r,s):
            r_coordinate = (nodes_on_O_gridline[0]-nodes_on_O_gridline[1])*r + nodes_on_O_gridline[1]
            point = O_grid_line_surface_suction.eval(r_coordinate,s)
            x = point.x
            y = point.y
            z = point.z
            return (x,y,z)

        Out_west = PyFunctionSurface(Out_west_python)
        Out_c47 = PathOnSurface (Out_west, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Out_c37 = PathOnSurface (Out_west, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
        Out_c04 = PathOnSurface (Out_west, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
        Out_c03 = PathOnSurface (Out_west, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
        #print "Out_west +==================================================+"

        Out_p4 = Out_c47.eval(0)
        Out_p7 = Out_c47.eval(1)
        Out_p0 = Out_c03.eval(0)
        Out_p3 = Out_c03.eval(1)

        #if pressure_or_suction > 0:
```

```
        Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
        Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

        Out_c45 = simple_connector(Out_p4,Out_p5,nodes_on_O_gridline[0],Ogridcurve_top,camber_curves,-1)
        Out_c76 = simple_connector(Out_p7,Out_p6,nodes_on_O_gridline[1],Ogridcurve_top,camber_curves,-1)
        Out_c01 = simple_connector(Out_p0,Out_p1,nodes_on_O_gridline[0],Ogridcurve_bottom,camber_curves,0)
        Out_c32 = simple_connector(Out_p3,Out_p2,nodes_on_O_gridline[1],Ogridcurve_bottom,camber_curves,0)
        #Out_c45.reverse()
        #Out_c76.reverse()
        #Out_c01.reverse()
        #Out_c32.reverse()


        ##print "p0", p0 , "\n p1" , p1 , "\n p2" ,p2 , "\n p3" ,p3 ,"\n p4" ,p4 , "\n p5" , p5 ,"\n p6",p6 ,"\n p7" ,p7
        Out_top = CoonsPatch(Out_c45,Out_c76,Out_c47,Out_c56)
        #print "Out_top+==========================================================+"
        #
        Out_bottom = CoonsPatch(Out_c01,Out_c32,Out_c03,Out_c12)
        #print "Out_bottom+==========================================================+"

        Out_north = CoonsPatch(Out_c32,Out_c76,Out_c37,Out_c26)
        #print "Out_north+==========================================================+"

        Out_south = CoonsPatch(Out_c01,Out_c45,Out_c04,Out_c15)
        #print "Out_south+==========================================================+"

        Out_volume = ParametricVolume(Out_north,Out_east,Out_south,Out_west,Out_top,Out_bottom)

        #A=[]
        A.append(Out_c01)
        #A.append(Out_c32)
        A.append(Out_c12)
        A.append(Out_c03)
        A.append(Out_c45)
        A.append(Out_c56)
        #A.append(Out_c76)
        A.append(Out_c47)
        #A.append(Out_c26)
        A.append(Out_c15)
        A.append(Out_c04)
        #A.append(Out_c37)
        #for n in range(0,len(A)):
        #    string = "curve_file%g.txt" %n
        #    print string , n
        #    with open(string,"w") as thisfile:
        #        thisfile.write("")
        #    with open(string,"a") as thisfile:
        #        for i in range (0,101):
        #            B = A[n].eval(i*0.01)
        #            C = "%g %g %g \n" %(B.x,B.y,B.z)
        #            #print C
        #            thisfile.write(C)

    #Out_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
    #       parametric_volume=Out_volume,
    #       fill_condition=initialCond))
    return "hello"
    #return Out_block

#print create_Out_Block([0.3,0.4],[0.95,1],-1)

# ################################################################################################################
# ################################################################################################################
```

```
# ####################################################################################################
# ####################################################################################################
# ####################################################################################################
##Let us build the Topography ##########################################################################
# ####################################################################################################
# ####################################################################################################
# ####################################################################################################
# ####################################################################################################
# ####################################################################################################
#print "here we go"

##Nodes_on_Blade_pressure = [0.3,0.4,0.5,0.6,0.7]
##Nodes_on_Blade_pressure = [0.3,0.4,0.5,0.6,0.7]
Nodes_on_Blade_pressure = []
Nodes_on_Blade_pressure.append(0)        #node 0 Bp
Nodes_on_Blade_pressure.append(0.03)     #node 1 Bp
Nodes_on_Blade_pressure.append(0.1)      #node 2 Bp
Nodes_on_Blade_pressure.append(0.2)      #node 3 Bp
Nodes_on_Blade_pressure.append(0.3)      #node 4 Bp
Nodes_on_Blade_pressure.append(0.4)      #node 5 Bp
Nodes_on_Blade_pressure.append(0.5)      #node 6 Bp
Nodes_on_Blade_pressure.append(0.6)      #node 7 Bp
Nodes_on_Blade_pressure.append(0.65)      #node 8 Bp
Nodes_on_Blade_pressure.append(0.72)      #node 9 Bp
Nodes_on_Blade_pressure.append(0.85)     #node 10 Bp
Nodes_on_Blade_pressure.append(0.9)        #node 11 Bp
Nodes_on_Blade_pressure.append(0.98)     #node 12 Bp
Nodes_on_Blade_pressure.append(1)        #node 13 Bp

#Nodes_on_Blade_suction = [0,0.05,0.1,0.2,0.4,0.45,0.5,0.55,0.6,0.7,0.9,0.95]
Nodes_on_Blade_suction = []
Nodes_on_Blade_suction.append(0)        #node 0 Bs
Nodes_on_Blade_suction.append(0.03)     #node 1 Bs
Nodes_on_Blade_suction.append(0.1)      #node 2 Bs
Nodes_on_Blade_suction.append(0.2)      #node 3 Bs
Nodes_on_Blade_suction.append(0.3)      #node 4 Bs
Nodes_on_Blade_suction.append(0.4)      #node 5 Bs
Nodes_on_Blade_suction.append(0.5)      #node 6 Bs
Nodes_on_Blade_suction.append(0.6)      #node 7 Bs
Nodes_on_Blade_suction.append(0.65)      #node 8 Bs
Nodes_on_Blade_suction.append(0.75)      #node 9 Bs
Nodes_on_Blade_suction.append(0.85)     #node 10 Bs
Nodes_on_Blade_suction.append(0.88)        #node 11 Bs
Nodes_on_Blade_suction.append(0.98)     #node 12 Bs
Nodes_on_Blade_suction.append(1)        #node 13 Bs

##Nodes_on_o_grid_line_pressure = [0,0.03,0.1,0.2,0.3,0.4,0.5,0.6,0.8,0.9,0.98,1]
Nodes_on_o_grid_line_pressure = []
Nodes_on_o_grid_line_pressure.append(0)        #node 0 Op
Nodes_on_o_grid_line_pressure.append(0.03)     #node 1 Op
Nodes_on_o_grid_line_pressure.append(0.1)      #node 2 Op
Nodes_on_o_grid_line_pressure.append(0.2)      #node 3 Op
Nodes_on_o_grid_line_pressure.append(0.3)      #node 4 Op
Nodes_on_o_grid_line_pressure.append(0.4)      #node 5 Op
Nodes_on_o_grid_line_pressure.append(0.5)      #node 6 Op
Nodes_on_o_grid_line_pressure.append(0.6)      #node 7 Op
Nodes_on_o_grid_line_pressure.append(0.65)      #node 8 Op
Nodes_on_o_grid_line_pressure.append(0.7)      #node 9 Op
```

```
Nodes_on_o_grid_line_pressure.append(0.85)      #node 10 Op
Nodes_on_o_grid_line_pressure.append(0.9)         #node 11 Op
Nodes_on_o_grid_line_pressure.append(0.98)      #node 12 Op
Nodes_on_o_grid_line_pressure.append(1)         #node 13 Op

#Nodes_on_o_grid_line_suction = [0,0.05,0.1,0.2,0.4,0.45,0.5,0.55,0.6,0.7,0.9,0.95]
Nodes_on_o_grid_line_suction = []
Nodes_on_o_grid_line_suction.append(0)         #node 0 Os
Nodes_on_o_grid_line_suction.append(0.03)      #node 1 Os
Nodes_on_o_grid_line_suction.append(0.1)       #node 2 Os
Nodes_on_o_grid_line_suction.append(0.2)       #node 3 Os
Nodes_on_o_grid_line_suction.append(0.3)       #node 4 Os
Nodes_on_o_grid_line_suction.append(0.4)       #node 5 Os
Nodes_on_o_grid_line_suction.append(0.5)       #node 6 Os
Nodes_on_o_grid_line_suction.append(0.6)       #node 7 Os
Nodes_on_o_grid_line_suction.append(0.65)       #node 8 Os
Nodes_on_o_grid_line_suction.append(0.7)       #node 9 Os
Nodes_on_o_grid_line_suction.append(0.82)      #node 10 Os
Nodes_on_o_grid_line_suction.append(0.9)         #node 11 Os
Nodes_on_o_grid_line_suction.append(0.96)      #node 12 Os
Nodes_on_o_grid_line_suction.append(1)         #node 13 Os

Nodes_on_inlet_pressure = [0.9,1]

Nodes_on_inlet_suction = [0.0,0.1]

#Nodes_on_periodic_pressure = [0.15,0.28,0.36,0.4,0.44,0.475,0.65,0.7,0.8,0.95]
Nodes_on_periodic_pressure = []
Nodes_on_periodic_pressure.append(0.15)          #node 0 Pp
Nodes_on_periodic_pressure.append(0.28)          #node 1 Pp
Nodes_on_periodic_pressure.append(0.36)          #node 2 Pp
Nodes_on_periodic_pressure.append(0.4)           #node 3 Pp
Nodes_on_periodic_pressure.append(0.44)          #node 4 Pp
Nodes_on_periodic_pressure.append(0.475)         #node 5 Pp
Nodes_on_periodic_pressure.append(0.5)          #node 6 Pp
Nodes_on_periodic_pressure.append(0.55)           #node 7 Pp
Nodes_on_periodic_pressure.append(0.57)           #node 8 Pp
Nodes_on_periodic_pressure.append(0.60)          #node 9 Pp
Nodes_on_periodic_pressure.append(0.62)          #node 10 Pp
Nodes_on_periodic_pressure.append(0.65)             #node 11 Pp

#Nodes_on_periodic_suction = [0,0.05,0.1,0.2,0.4,0.45,0.5,0.55,0.6,0.7,0.9,0.95]
Nodes_on_periodic_suction = []
Nodes_on_periodic_suction.append(0.15)          #node 0 Ps
Nodes_on_periodic_suction.append(0.28)          #node 1 Ps
Nodes_on_periodic_suction.append(0.36)          #node 2 Ps
Nodes_on_periodic_suction.append(0.4)           #node 3 Ps
Nodes_on_periodic_suction.append(0.44)          #node 4 Ps
Nodes_on_periodic_suction.append(0.475)         #node 5 Ps
Nodes_on_periodic_suction.append(0.5)          #node 6 Ps
Nodes_on_periodic_suction.append(0.55)           #node 7 Ps
Nodes_on_periodic_suction.append(0.60)           #node 8 Ps
Nodes_on_periodic_suction.append(0.62)          #node 9 Ps
Nodes_on_periodic_suction.append(0.7)          #node 10 Ps
Nodes_on_periodic_suction.append(0.9)             #node 11 Ps


Nodes_on_outlet_pressure = [0.9,1]

Nodes_on_outlet_suction = [0.0,0.4]

#Nodes_on_Outlet_surface = [0,0.05,0.1,0.2,0.4,0.45,0.5,0.55,0.6,0.7,0.9,0.95]
#Nodes_on_periodic_suction = [0,0.05,0.1,0.2,0.4,0.45,0.5,0.55,0.6,0.7,0.9,0.95]
Inflow_pressure_corner_node = 1 ##the number here is the position/index of the inflow corner node in the list of O_gridlines
Inflow_suction_corner_node = 1 ##the number here is the position/index of the inflow corner node in the list of O_gridlines
Outflow_pressure_corner_node = -2 ##the number here is the position/index of the outflow corner node in the list of O_gridlines
Outflow_suction_corner_node = -2 ##the number here is the position/index of the outflow corner node in the list of O_gridlines
#
#Testlist = [ 1, 2, 3, 4, 5]
#
#print "Testlist" , Testlist[Outflow_pressure_corner_node+0:Outflow_pressure_corner_node+0+2]
```

```
#print "Testlist" , Testlist[Outflow_pressure_corner_node+1:Outflow_pressure_corner_node+1+2]
#print "Testlist" , Testlist[Outflow_pressure_corner_node+2:Outflow_pressure_corner_node+2+2]
#print "Testlist" , [Testlist[Outflow_pressure_corner_node],Testlist[Outflow_pressure_corner_node+1]]

#Nodes_on_o_grid_line_pressure = [0.3,0.4,0.5]
#Nodes_on_periodic_pressure = [0.32,0.42,0.52]
A=[]
#Blocks = []
################################################################################################
################################################################################################
################################################################################################
################################################################################################
################################################################################################
### Pressure side: inflow -> corner -> periodic -> corner -> outflow -> O-grid

####inflow
for i in range (0,len(Nodes_on_inlet_pressure)-1):
    print create_In_Block(Nodes_on_inlet_pressure,Nodes_on_o_grid_line_pressure[i:i+2],1), "this is IN block number", i
    ###Blocks.append(create_In_Block(Nodes_on_inlet_pressure,Nodes_on_o_grid_line_pressure[i:i+2],1))

####inflow corner
#
I_cor_p7 = v6c47.eval(Nodes_on_inlet_pressure[0])
I_cor_p3 = v6c03.eval(Nodes_on_inlet_pressure[0])
I_cor_p4 = v6p4
I_cor_p0 = v6p0

I_cor_c47 = Arc(I_cor_p4, I_cor_p7, Vector(0,0,I_cor_p4.z))
I_cor_c03 = Arc(I_cor_p0, I_cor_p3, Vector(0,0,I_cor_p0.z))
I_cor_c37 = Line(I_cor_p3, I_cor_p7)
I_cor_c04 = Line(I_cor_p0, I_cor_p4)

I_cor_west = CoonsPatch(I_cor_c03, I_cor_c47, I_cor_c04, I_cor_c37)


def I_cor_south_py(r,s):
    r_coordinate = Nodes_on_periodic_pressure[0]*r
    point = pressure_periodic.eval(r_coordinate,s)
    x = point.x
    y = point.y
    z = point.z
    return (x,y,z)

I_cor_south = PyFunctionSurface(I_cor_south_py)
I_cor_c01 = PathOnSurface (I_cor_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
I_cor_c15 = PathOnSurface (I_cor_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
I_cor_c04 = PathOnSurface (I_cor_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
I_cor_c45 = PathOnSurface (I_cor_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))

I_cor_p1 = pressure_periodic.eval(Nodes_on_periodic_pressure[0],0)
I_cor_p5 = pressure_periodic.eval(Nodes_on_periodic_pressure[0],1)

Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))

I_cor_p6 = Ogridcurve_top.eval(Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node])
I_cor_p2 = Ogridcurve_bottom.eval(Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node])

I_cor_c56= simple_connector(I_cor_p6,I_cor_p5,Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node],Ogridcurve_top,camber_curves,-1)
```

```
I_cor_c12 = simple_connector(I_cor_p2,I_cor_p1,Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node],Ogridcurve_bottom,camber_curves,0)
I_cor_c76 = simple_connector(I_cor_p6,I_cor_p7,Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node],Ogridcurve_top,camber_curves,-1)
I_cor_c32 = simple_connector(I_cor_p2,I_cor_p3,Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node],Ogridcurve_bottom,camber_curves,0)
I_cor_c56.reverse()
I_cor_c12.reverse()
I_cor_c76.reverse()
I_cor_c32.reverse()

I_cor_c26 = PathOnSurface(O_grid_line_surface_pressure,LinearFunction(0.0,Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node]), LinearFunction(1.0,0.0))

I_cor_top = CoonsPatch(I_cor_c45,I_cor_c76,I_cor_c47,I_cor_c56)
print "I_cor_top+==================================================+"
#
I_cor_bottom = CoonsPatch(I_cor_c01,I_cor_c32,I_cor_c03,I_cor_c12)
print "I_cor_bottom+==================================================+"

I_cor_north = CoonsPatch(I_cor_c32,I_cor_c76,I_cor_c37,I_cor_c26)
print "I_cor_north+==================================================+"

I_cor_east = CoonsPatch(I_cor_c12,I_cor_c56,I_cor_c15,I_cor_c26)
print "I_cor_east+==================================================+"

I_cor_volume = ParametricVolume(I_cor_north,I_cor_east,I_cor_south,I_cor_west,I_cor_top,I_cor_bottom)

#A=[]
A.append(I_cor_c01)
A.append(I_cor_c32)
A.append(I_cor_c12)
A.append(I_cor_c03)
A.append(I_cor_c45)
A.append(I_cor_c56)
A.append(I_cor_c76)
A.append(I_cor_c47)
A.append(I_cor_c26)
A.append(I_cor_c15)
A.append(I_cor_c04)
A.append(I_cor_c37)
#for n in range(0,len(A)):
#    string = "curve_file%g.txt" %n
#    print string , n , "of", len(A)
#    with open(string,"w") as thisfile:
#        thisfile.write("")
#    with open(string,"a") as thisfile:
#        for i in range (0,101):
#            B = A[n].eval(i*0.01)
#            C = "%g %g %g \n" %(B.x,B.y,B.z)
#            #print C
#            thisfile.write(C)
##
#
##I_cor_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
##       parametric_volume=I_cor_volume,
##       fill_condition=initialCond))
#
#
#################################################################################################################
### Pressure side: -> periodic
for i in range (0,len(Nodes_on_periodic_pressure)-1):
    print create_H_Block(Nodes_on_periodic_pressure[i:i+2],Nodes_on_o_grid_line_pressure[Inflow_pressure_corner_node+i:i+2+Inflow_pressure_corner_node],1), "this is H block number", i
    #Blocks.append(create_H_Block(Nodes_on_periodic_pressure[i:i+2],Nodes_on_o_grid_line_pressure[i:i+2],1))
#
#
#################################################################################################################
### Pressure side: -> outflow corner :
```

```
Out_cor_p2 = v2c12.eval(Nodes_on_outlet_pressure[0])
Out_cor_p6 = v2c56.eval(Nodes_on_outlet_pressure[0])
Out_cor_p5 = v2p5
Out_cor_p1 = v2p1

Out_cor_c56 = Arc(Out_cor_p5, Out_cor_p6, Vector(0,0,Out_cor_p5.z))
Out_cor_c12 = Arc(Out_cor_p1, Out_cor_p2, Vector(0,0,Out_cor_p1.z))
Out_cor_c15 = Line(Out_cor_p1, Out_cor_p5)
Out_cor_c26 = Line(Out_cor_p2, Out_cor_p6)

Out_cor_east = CoonsPatch(Out_cor_c12, Out_cor_c56, Out_cor_c15, Out_cor_c26)


def Out_cor_south_py(r,s):
    r_coordinate = (1-Nodes_on_periodic_pressure[-1])*r+Nodes_on_periodic_pressure[-1]
    point = pressure_periodic.eval(r_coordinate,s)
    x = point.x
    y = point.y
    z = point.z
    return (x,y,z)

Out_cor_south = PyFunctionSurface(Out_cor_south_py)
Out_cor_c01 = PathOnSurface (Out_cor_south, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
Out_cor_c15 = PathOnSurface (Out_cor_south, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
Out_cor_c04 = PathOnSurface (Out_cor_south, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
Out_cor_c45 = PathOnSurface (Out_cor_south, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))

Out_cor_p0 = pressure_periodic.eval(Nodes_on_periodic_pressure[-1],0)
Out_cor_p4 = pressure_periodic.eval(Nodes_on_periodic_pressure[-1],1)

Ogridcurve_top = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0),
    LinearFunction(0.0,1.0))
Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_pressure, LinearFunction(1.0,0.0),
    LinearFunction(0.0,0.0))

Out_cor_p7 = Ogridcurve_top.eval(Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node])
Out_cor_p3 = Ogridcurve_bottom.eval(Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node
    ])

Out_cor_c76 = simple_connector(Out_cor_p7,Out_cor_p6,Nodes_on_o_grid_line_pressure[
    Outflow_pressure_corner_node],Ogridcurve_top,camber_curves,-1)
Out_cor_c47 = simple_connector(Out_cor_p7,Out_cor_p4,Nodes_on_o_grid_line_pressure[
    Outflow_pressure_corner_node],Ogridcurve_top,camber_curves,-1)
Out_cor_c03 = simple_connector(Out_cor_p3,Out_cor_p0,Nodes_on_o_grid_line_pressure[
    Outflow_pressure_corner_node],Ogridcurve_bottom,camber_curves,0)
Out_cor_c32 = simple_connector(Out_cor_p3,Out_cor_p2,Nodes_on_o_grid_line_pressure[
    Outflow_pressure_corner_node],Ogridcurve_bottom,camber_curves,0)
Out_cor_c47.reverse()
Out_cor_c03.reverse()

Out_cor_c37 = PathOnSurface(O_grid_line_surface_pressure,LinearFunction(0.0,
    Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node]), LinearFunction(1.0,0.0))

Out_cor_top = CoonsPatch(Out_cor_c45,Out_cor_c76,Out_cor_c47,Out_cor_c56)
print "Out_cor_top+==========================================================+"
#
Out_cor_bottom = CoonsPatch(Out_cor_c01,Out_cor_c32,Out_cor_c03,Out_cor_c12)
print "Out_cor_bottom+==========================================================+"

Out_cor_north = CoonsPatch(Out_cor_c32,Out_cor_c76,Out_cor_c37,Out_cor_c26)
print "Out_cor_north+==========================================================+"

Out_cor_west = CoonsPatch(Out_cor_c03,Out_cor_c47,Out_cor_c04,Out_cor_c37)
print "Out_cor_east+==========================================================+"

Out_cor_volume = ParametricVolume(Out_cor_north,Out_cor_east,Out_cor_south,Out_cor_west,
    Out_cor_top,Out_cor_bottom)

#A=[]
A.append(Out_cor_c01)
A.append(Out_cor_c32)
A.append(Out_cor_c12)
A.append(Out_cor_c03)
A.append(Out_cor_c45)
```

```
A.append(Out_cor_c56)
A.append(Out_cor_c76)
A.append(Out_cor_c47)
A.append(Out_cor_c26)
A.append(Out_cor_c15)
A.append(Out_cor_c04)
A.append(Out_cor_c37)
##for n in range(0,len(A)):
##     string = "curve_file%g.txt" %n
##     print string , n , "of", len(A)
##     with open(string,"w") as thisfile:
##         thisfile.write("")
##     with open(string,"a") as thisfile:
##         for i in range (0,101):
##             B = A[n].eval(i*0.01)
##             C = "%g %g %g \n" %(B.x,B.y,B.z)
##             #print C
##             thisfile.write(C)


##Out_cor_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
##       parametric_volume=Out_cor_volume,
##       fill_condition=initialCond))

# ##########################################################################################################
##### Pressure side:  -> outflow blocks :

for i in range (0,len(Nodes_on_outlet_pressure)-1):
    print create_Out_Block(Nodes_on_outlet_pressure[i:i+2],[Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node+i],Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node+i+1]],1), "this is Out block number", i
    #Blocks.append(create_Out_Block(Nodes_on_outlet_pressure[i:i+2],[Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node+i],Nodes_on_o_grid_line_pressure[Outflow_pressure_corner_node+i+1]],1))
#
# ##########################################################################################################
#### Pressure side O-grid blocks
##for i in range (len(Nodes_on_o_grid_line_pressure)-2,len(Nodes_on_o_grid_line_pressure)-1):
for i in range (0,len(Nodes_on_o_grid_line_pressure)-1):
    print create_O_Block(Nodes_on_o_grid_line_pressure[i:i+2],Nodes_on_Blade_pressure[i:i+2],1), "this is O block number", i
#    Blocks.append(create_O_Block(Nodes_on_o_grid_line_pressure[i:i+2],Nodes_on_Blade_pressure[i:i+2],1))
# ########################################################################################################
# ########################################################################################################
# ########################################################################################################
# ########################################################################################################
# ########################################################################################################
## Suction side:  inflow ->  corner  -> periodic -> corner -> outflow

###inflow
for i in range (0,len(Nodes_on_inlet_suction)-1):
    print create_In_Block(Nodes_on_inlet_suction,Nodes_on_o_grid_line_suction[i:i+2],-1), "this is In block number", i
    #Blocks.append(create_In_Block(Nodes_on_inlet_suction,Nodes_on_o_grid_line_suction[i:i+2],-1))

###inflow corner suction

I_cor_p7 = v5p7
I_cor_p3 = v5p3
```

```
I_cor_p4 = v5c47.eval(Nodes_on_inlet_suction[-1])
I_cor_p0 = v5c03.eval(Nodes_on_inlet_suction[-1])

I_cor_c47 = Arc(I_cor_p4, I_cor_p7, Vector(0,0,I_cor_p4.z))
I_cor_c03 = Arc(I_cor_p0, I_cor_p3, Vector(0,0,I_cor_p0.z))
I_cor_c37 = Line(I_cor_p3, I_cor_p7)
I_cor_c04 = Line(I_cor_p0, I_cor_p4)

I_cor_west = CoonsPatch(I_cor_c03, I_cor_c47, I_cor_c04, I_cor_c37)

def I_cor_north_py(r,s):
    r_coordinate = Nodes_on_periodic_suction[0]*r
    point = suction_periodic.eval(r_coordinate,s)
    x = point.x
    y = point.y
    z = point.z
    return (x,y,z)

I_cor_north = PyFunctionSurface(I_cor_north_py)
I_cor_c32 = PathOnSurface (I_cor_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
I_cor_c26 = PathOnSurface (I_cor_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
I_cor_c37 = PathOnSurface (I_cor_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
I_cor_c76 = PathOnSurface (I_cor_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))

I_cor_p2 = suction_periodic.eval(Nodes_on_periodic_suction[0],0)
I_cor_p6 = suction_periodic.eval(Nodes_on_periodic_suction[0],1)

Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0),
    LinearFunction(0.0,1.0))
Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0),
    LinearFunction(0.0,0.0))

I_cor_p5 = Ogridcurve_top.eval(Nodes_on_o_grid_line_suction[Inflow_suction_corner_node])
I_cor_p1 = Ogridcurve_bottom.eval(Nodes_on_o_grid_line_suction[Inflow_suction_corner_node])

I_cor_c56= simple_connector(I_cor_p5,I_cor_p6,Nodes_on_o_grid_line_suction[
    Inflow_suction_corner_node],Ogridcurve_top,camber_curves,-1)
I_cor_c12 = simple_connector(I_cor_p1,I_cor_p2,Nodes_on_o_grid_line_suction[
    Inflow_suction_corner_node],Ogridcurve_bottom,camber_curves,0)
I_cor_c45 = simple_connector(I_cor_p4,I_cor_p5,Nodes_on_o_grid_line_suction[
    Inflow_suction_corner_node],Ogridcurve_top,camber_curves,-1)
I_cor_c01 = simple_connector(I_cor_p0,I_cor_p1,Nodes_on_o_grid_line_suction[
    Inflow_suction_corner_node],Ogridcurve_bottom,camber_curves,0)

I_cor_c15 = PathOnSurface(O_grid_line_surface_suction,LinearFunction(0.0,
    Nodes_on_o_grid_line_suction[Inflow_suction_corner_node]), LinearFunction(1.0,0.0))

I_cor_top = CoonsPatch(I_cor_c45,I_cor_c76,I_cor_c47,I_cor_c56)
print "I_cor_top+=============================================+"
#
I_cor_bottom = CoonsPatch(I_cor_c01,I_cor_c32,I_cor_c03,I_cor_c12)
print "I_cor_bottom+=============================================+"

I_cor_south = CoonsPatch(I_cor_c01,I_cor_c45,I_cor_c04,I_cor_c15)
print "I_cor_south+=============================================+"

I_cor_east = CoonsPatch(I_cor_c12,I_cor_c56,I_cor_c15,I_cor_c26)
print "I_cor_east+=============================================+"

I_cor_volume = ParametricVolume(I_cor_north,I_cor_east,I_cor_south,I_cor_west,I_cor_top,
    I_cor_bottom)

###A=[]
A.append(I_cor_c01)
A.append(I_cor_c32)
A.append(I_cor_c12)
A.append(I_cor_c03)
A.append(I_cor_c45)
A.append(I_cor_c56)
A.append(I_cor_c76)
A.append(I_cor_c47)
A.append(I_cor_c26)
A.append(I_cor_c15)
A.append(I_cor_c04)
A.append(I_cor_c37)
```

```
#for n in range(0,len(A)):
#    string = "curve_file%g.txt" %n
#    print string , n , "of", len(A)
#    with open(string,"w") as thisfile:
#        thisfile.write("")
#    with open(string,"a") as thisfile:
#        for i in range (0,101):
#            B = A[n].eval(i*0.01)
#            C = "%g %g %g \n" %(B.x,B.y,B.z)
#            #print C
#            thisfile.write(C)
##
##
#I_cor_block = (Block3D(label="TEST-BLOCK", nni=30, nnj=5, nnk=5,
#      parametric_volume=I_cor_volume,
#      fill_condition=initialCond))
##
##
# ############################################################################################################
#### Suction side: -> periodic
for i in range (0,len(Nodes_on_periodic_suction)-1):
    print create_H_Block(Nodes_on_periodic_suction[i:i+2],Nodes_on_o_grid_line_suction[Inflow_suction_corner_node+i:i+2+Inflow_suction_corner_node],-1), "this is H block number", i
    #Blocks.append(create_H_Block(Nodes_on_periodic_suction[i:i+2],Nodes_on_o_grid_line_suction[i:i+2],-1))
##
# ############################################################################################################
####Suction side: -> outflow corner :

Out_cor_p1 = v3c12.eval(Nodes_on_outlet_suction[-1])
Out_cor_p5 = v3c56.eval(Nodes_on_outlet_suction[-1])
Out_cor_p6 = v3p6
Out_cor_p2 = v3p2
#
Out_cor_c56 = Arc(Out_cor_p5, Out_cor_p6, Vector(0,0,Out_cor_p5.z))
Out_cor_c12 = Arc(Out_cor_p1, Out_cor_p2, Vector(0,0,Out_cor_p1.z))
Out_cor_c15 = Line(Out_cor_p1, Out_cor_p5)
Out_cor_c26 = Line(Out_cor_p2, Out_cor_p6)
#
Out_cor_east = CoonsPatch(Out_cor_c12, Out_cor_c56, Out_cor_c15, Out_cor_c26)
#
#
def Out_cor_north_py(r,s):
    r_coordinate = (1-Nodes_on_periodic_suction[-1])*r+Nodes_on_periodic_suction[-1]
    point = suction_periodic.eval(r_coordinate,s)
    x = point.x
    y = point.y
    z = point.z
    return (x,y,z)
#
Out_cor_north = PyFunctionSurface(Out_cor_north_py)
Out_cor_c32 = PathOnSurface (Out_cor_north, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
#Out_cor_c26 = PathOnSurface (Out_cor_north, LinearFunction(0.0,1.0), LinearFunction(1.0,0.0))
Out_cor_c37 = PathOnSurface (Out_cor_north, LinearFunction(0.0,0.0), LinearFunction(1.0,0.0))
Out_cor_c76 = PathOnSurface (Out_cor_north, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
#
Out_cor_p3 = suction_periodic.eval(Nodes_on_periodic_suction[-1],0)
Out_cor_p7 = suction_periodic.eval(Nodes_on_periodic_suction[-1],1)
#
Ogridcurve_top = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,1.0))
Ogridcurve_bottom = PathOnSurface(O_grid_line_surface_suction, LinearFunction(1.0,0.0), LinearFunction(0.0,0.0))
#
Out_cor_p4 = Ogridcurve_top.eval(Nodes_on_o_grid_line_suction[Outflow_suction_corner_node])
Out_cor_p0 = Ogridcurve_bottom.eval(Nodes_on_o_grid_line_suction[Outflow_suction_corner_node])
#
Out_cor_c45 = simple_connector(Out_cor_p4,Out_cor_p5,Nodes_on_o_grid_line_suction[Outflow_suction_corner_node],Ogridcurve_top,camber_curves,-1)
```

```
Out_cor_c47 = simple_connector(Out_cor_p4,Out_cor_p7,Nodes_on_o_grid_line_suction[
    Outflow_suction_corner_node],Ogridcurve_top,camber_curves,-1)
Out_cor_c01 = simple_connector(Out_cor_p0,Out_cor_p1,Nodes_on_o_grid_line_suction[
    Outflow_suction_corner_node],Ogridcurve_bottom,camber_curves,0)
Out_cor_c03 = simple_connector(Out_cor_p0,Out_cor_p3,Nodes_on_o_grid_line_suction[
    Outflow_suction_corner_node],Ogridcurve_bottom,camber_curves,0)

#
Out_cor_c04 = PathOnSurface(O_grid_line_surface_suction,LinearFunction(0.0,
    Nodes_on_o_grid_line_suction[Outflow_suction_corner_node]), LinearFunction(1.0,0.0))

Out_cor_top = CoonsPatch(Out_cor_c45,Out_cor_c76,Out_cor_c47,Out_cor_c56)
print "Out_cor_top+================================================+"
#
Out_cor_bottom = CoonsPatch(Out_cor_c01,Out_cor_c32,Out_cor_c03,Out_cor_c12)
print "Out_cor_bottom+================================================+"

Out_cor_south = CoonsPatch(Out_cor_c01,Out_cor_c45,Out_cor_c04,Out_cor_c15)
print "Out_cor_south+================================================+"

Out_cor_west = CoonsPatch(Out_cor_c03,Out_cor_c47,Out_cor_c04,Out_cor_c37)
print "Out_cor_east+================================================+"

Out_cor_volume = ParametricVolume(Out_cor_north,Out_cor_east,Out_cor_south,Out_cor_west,
    Out_cor_top,Out_cor_bottom)

##A=[]
A.append(Out_cor_c01)
A.append(Out_cor_c32)
A.append(Out_cor_c12)
A.append(Out_cor_c03)
A.append(Out_cor_c45)
A.append(Out_cor_c56)
A.append(Out_cor_c76)
A.append(Out_cor_c47)
A.append(Out_cor_c26)
A.append(Out_cor_c15)
A.append(Out_cor_c04)
A.append(Out_cor_c37)
###for n in range(0,len(A)):
###    string = "curve_file%g.txt" %n
###    print string , n , "of", len(A)
###    with open(string,"w") as thisfile:
###        thisfile.write("")
###    with open(string,"a") as thisfile:
###        for i in range (0,101):
###            B = A[n].eval(i*0.01)
###            C = "%g %g %g \n" %(B.x,B.y,B.z)
###            #print C
###             thisfile.write(C)
#
#
###Out_cor_block = (Block3D(label="TEST-BLOCK", nni=5, nnj=5, nnk=5,
###      parametric_volume=Out_cor_volume,
###      fill_condition=initialCond))
#
#
    ##############################################################################################
###### Suction side:  -> outflow blocks :
#
for i in range (0,len(Nodes_on_outlet_suction)-1):
    print create_Out_Block(Nodes_on_outlet_suction[i:i+2],[Nodes_on_o_grid_line_suction[
     Outflow_suction_corner_node+i],Nodes_on_o_grid_line_suction[Outflow_suction_corner_node+i
     +1]],-1), "this is Out block number", i
     #Blocks.append(create_Out_Block(Nodes_on_outlet_suction[i:i+2],[
     Nodes_on_o_grid_line_suction[Outflow_suction_corner_node+i],Nodes_on_o_grid_line_suction[
     Outflow_suction_corner_node+i+1]],-1))
#
    ############################################################################################
#### Suction side O-grid blocks
###for i in range (len(Nodes_on_o_grid_line_suction)-2,len(Nodes_on_o_grid_line_suction)-1):
#for i in range (0,len(Nodes_on_o_grid_line_suction)-1):
```

```
#     print create_O_Block(Nodes_on_o_grid_line_suction[i:i+2],Nodes_on_Blade_suction[i:i
      +2],-1), "this is O block number", i
#     Blocks.append(create_O_Block(Nodes_on_o_grid_line_suction[i:i+2],Nodes_on_Blade_suction[i
      :i+2],-1))
#identify_block_connections()

for n in range(0,len(A)):
    string = "curve_file%g.txt" %n
    print string , n , "of", len(A)
    with open(string,"w") as thisfile:
        thisfile.write("")
    with open(string,"a") as thisfile:
        for i in range (0,101):
            B = A[n].eval(i*0.01)
            C = "%g %g %g \n" %(B.x,B.y,B.z)
            ###print C
            thisfile.write(C)
#Blocks = []
#for i in range (0,len(Nodes_on_periodic_pressure)-1):
#     Blocks.append(create_H_Block(Nodes_on_periodic_pressure[i:i+2],
      Nodes_on_o_grid_line_pressure[i:i+2],1))
```

Bibliography

[1] J. Garnaut. *Scientists get hot rocks off over green nuclear power*, April 2007. URL http://www.smh.com.au/news/environment/hot-rock-power-the-way-ahead/2007/04/11/1175971183212.html.

[2] C. A. de Miranda Ventura. *Aerodynamic Design and Performance Estimation of Radial Inflow Turbines for Renewable Power Generation Applications*. PhD thesis, The University of Queensland, 2012.

[4] P. A. Jacobs and R. J. Gollan. The Eilmer3 code: User guide and example book. Technical report, The University of Queensland, 2014. URL https://espace.library.uq.edu.au/view/UQ:331606

[5] URL https://www.blender.org.