

Radial Inflow Turbine Meshing Rev. 1

Mechanical Engineering Technical Report 2015/03

Ingo JAHN

School of Mechanical and Mining Engineering

The University of Queensland.

June 12, 2015

Abstract

This report describes meshing utilities to support the mesh generation for Radial Inflow Turbines. This includes two tools:

- (a) parameterised mesh generator for Nozzle Guide Vanes;
- (b) a geometry and mesh generation tool for turbine rotors.

A key feature of the turbine rotor mesh and geometry generation tool is that it allows a parametric definition of the geometry based on properties of the aerodynamic passage. For example desired flow direction and evolution of flow area. This is in contrast to alternative methods, which start by defining the physical features of the rotor (e.g. hub and shroud shape), in which case the aerodynamic passage becomes an output.

In addition to providing the description of the two tools, usage instructions and examples are provided.

1 Introduction / Overview

Description and Userguide for `Inlet_Vane_Round_Square.py`, an automated job script for `e3prep.py` [1] to generate Nozzle Guide Vane (NGV) meshes and `Rotor_Passage.py`, a piece of code that can be used to generate parametric rotor passages for radial in-flow turbines or as an input to `e3prep.py` to generate appropriate rotor meshes. These tools allow the automated generation of geometries and structured meshes, based on a small number of parametric inputs. A key advantage, particularly for the rotor passage, is that the mesh is defined by a small number of input parameters that directly define the aerodynamic properties of the passage, thereby allowing optimisation by directly altering these properties. This is in contrast to alternative methods, which start by defining the physical features of the rotor (e.g. hub and shroud shape), in which case the aerodynamic passage becomes and output. The suitability for optimisation is further enhanced as all code is written in python, thereby allowing easy incorporation into scripted optimisation processes. NGV meshes can be 2-D and 3-D in the Eilmer format and the rotor meshes are in 3-D in the Eilmer format. The mesh can be used directly for simulations in Eilmer [3] or can be converted to the `foam` format used by OpenFoam [2] using the `e3preToFoam.py` utility [4].

The typical block structure used for a NGV mesh is shown in Figure 1(a). The parametrically defined rotor shape, shown for two adjacent passages in shown in Figure 1(b).

The report is split into the following sections.

- Section 2 provides information about obtaining and tool and associated code.
- Section 3 describes tools that have been developed to mesh Nozzle Guide Vanes.
- Section 4 describes the approach used to generate rotor passage geometries and associated boundary conforming meshes.
- Section 6 provides some brief conclusions of the work and an outline of future improvements and additions that are in preparation.

2 Distribution and Installation

2.1 Compatibility

The tools used to generate the meshes use functions from the *CFCFD Group* code collection Eilmer3 [3], python, and C++. The following dependencies exist:

Eilmer3 `e3prepToFoam` has been included as part of Eilmer code distribution from November 2014 onwards.

python The code has a number of python and C++ dependencies. It is recommended to install the dependencies list from the CFCFD webpage <http://cfcfd.mechmining.uq.edu.au/getting-started.html>

2.2 Citing this tool

When using the tool in simulations that lead to published works, it is requested that the following works are cited:

- This report to cover the mesh generation tools.
Ingo Jahn (2015), “Radial Inflow Turbine Meshing”, Mechanical Engineering Technical Report 2015/03, The University of Queensland

- The following report which covers `e3prep.py` the underlying code used to generate the mesh.
PA Jacobs, RJ Gollan, DF Potter (2014), “The Eilmer3 code: user guide and example book”, Mechanical Engineering Technical Report 2014/04, pp 1-447, The University of Queensland

2.3 License

The tools described within this report are distributed as part of the code collection maintained by the *CFCFD Group* at the University of Queensland [3]. This collection is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. This program collection is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details <http://www.gnu.org/licenses/>.

The code will be automatically installed during a typical build of Eilmer3. Download and build instructions are available from the CFCFD webpage <http://cfcfd.mechmining.uq.edu.au/>.

2.4 Modifying the code and Contributing

If you perform modifications or improvements to the code please submit an updated version together with a short description of the changes to the authors. Once reviewed the changes will be included in future versions of the code.

3 NGVs

3.1 Running the Tool

Follow these steps:

1. Define the NGV geometry in the `Inlet_Vane_Round_Square.py`
2. Modify settings in file `Inlet_Vane_Round_Square.py`
3. Run following command to create `e3prep` mesh:
`e3prep.py --job=Inlet_Vane_Round_Square.py --do-svg`
4. 2-D mesh topography is now available as `.svg`
5. Convert mesh to other format for further processing
6. Run following commands to convert mesh to `Vtk` format and view in `paraview`:
`e3post.py --job=Inlet_Vane_Round_Square.py --vtk-xml`
`paraview`
7. If using `Eilmer` for flow simulation, the job file `Inlet_Vane_Round_Square.py` needs to be adjusted to set appropriate simulation parameters.

3.2 Geometry Definition

The geometry of the NGV and the fluid domain is constructed using `### Defining the Geometry ###` section of the code. Figure 2 provides a graphical definition of the corresponding variables. The following variables define fluid domain:

```
# Dimensions defining the fluid Domain
R_in = 35.e-3           # (m)
R_out = 24.8e-3        # (m)
N_blade = 8            # Number of blades
Height = 0.003        # (m)
```

The domain is modelled as an arc segment of a ring. The arc width is defined by the blade number `N_blade`, the inner and outer edge of the domain are set by radii `R_in` and `R_out`. The left and right faces (periodic boundaries in typical simulations) are inclined to the respective radial directions at an angle that matches the angle of the NGV. In 3-D the mesh is extruded out of the page by the distance `Height`.

The actual NGV geometry is defined by:

```
# Dimensions defining the Vane
R_leading = 30e-3      # (m) position of Vane leading edge centre
alpha_trailing = 50./180*np.pi # (rad) Vane angle at trailing edge.
r_leading = 1.5e-3     # (m) radius of vane leading edge
r_trailing = 0.5e-3   # (m) thickness/radius of vane trailing edge
```

The NGV geometry is constructed as follows:

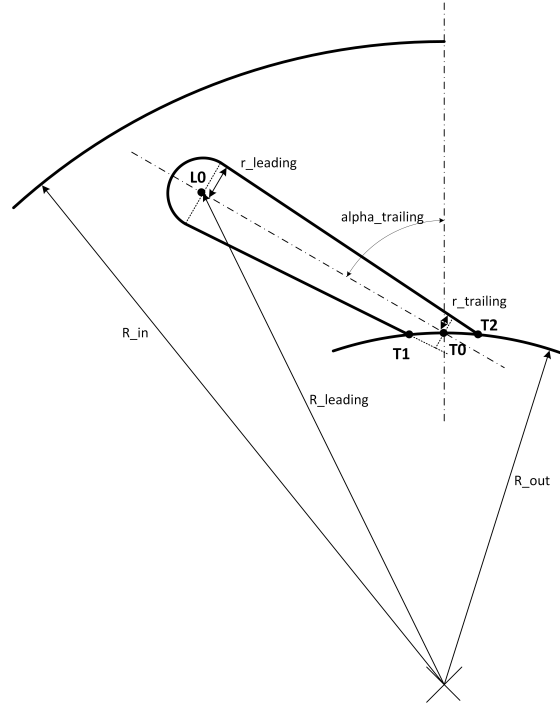


Figure 2: Variables defining the geometry of the NGV and fluid domain.

- T0, NGV trailing edge centre is placed at the centre of the domain outlet, defined by R_{out} .
- L0, NGV leading edge centre is placed at radius, $R_{leading}$, so that a NGV centreline with angle, $\alpha_{trailing}$ is created.
- NGV leading edge is constructed from a circle with radius, $r_{leading}$, centred at T0.
- NGV sides are constructed as the tangents to two circles. The with radius, $r_{leading}$, centred at L0 and the second with radius, $r_{trailing}$, centred at T0.
- The intersection of the NGV sides with the inner radius of the fluid domain (T1 and T2) are used as the ends of the sides.

3.3 Mesh Definition

The mesh is created in two parts. There is a NGV wrapping boundary layer mesh consisting of the blocks BL0, BL1, BL2, BL3, and BL4 and a far-field mesh consisting of the remaining blocks. The mesh topography together with the nodes at block corners is shown in Figure 3.

3.3.1 Mesh Construction and built-in constraints

Boundary layer mesh To create a wall conforming boundary layer mesh the The Boundary layer mesh is split into 5 sections as a consequence of the geometry definition as follows:

- L1 is placed at the end of the straight section of the NGV upstream side.

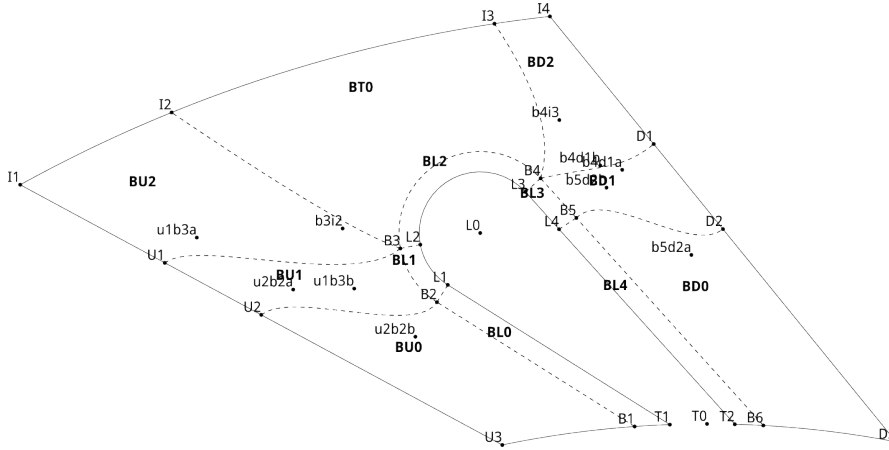


Figure 3: Mesh Topography.

- L4 is placed on the NGV downstream side, so that the distances T1L1 and T2L2 are matched. This ensures equal cell spacing on both sides of the NGV.
- L3 is placed at the end of the straight section of the NGV downstream side.
- L2 is placed on the NGV upstream side, so that the distances L3L4 and L1L2 are matched.

To define the upper limit of the boundary layer mesh, the control nodes B2, B3, B4, and B5 are placed at a distance, $b1$ normal to the wall. The control nodes B1 and B6 are placed such that the **BL0** and **BL4** have parallel edges.

Far-field mesh The far field mesh topography is predefined by the need to match cells between the two periodic boundaries at the left and right side of the domain. To ensure good mesh quality tuneable control points U1 and U2 (paired with D1 and D2) exist on the domain sides and control points I2 and I3 exist at the domain inlet (see 3.3.2). The lines linking the domain edges to the top of the boundary layer blocks are constructed as follows:

- $U2 \rightarrow B2$, 4 point bezier curve with intermediary control points $u2b2a$, $u2b2b$, which ensure line is perpendicular at both ends
- $U1 \rightarrow B3$, 4 point bezier curve with intermediary control points $u1b3a$, $u1b3b$, which ensure line is perpendicular at domain edge and forms 60° angle at B3. ($3 \times 60^\circ + 2 \times 90^\circ$)
- $B3 \rightarrow I2$, 3 point bezier curve with intermediary control point $b3i2$, which ensures line forms 60° angle at B3. ($3 \times 60^\circ + 2 \times 90^\circ$) The angles are chosen to maximise mesh quality, firstly in the boundary layer blocks and then the surrounding blocks.
- $B4 \rightarrow I3$, 3 point bezier curve with intermediary control point $b4i3$, which ensures line forms 60° angle at B4. ($3 \times 60^\circ + 2 \times 90^\circ$)
- $B4 \rightarrow D1$, 4 point bezier curve with intermediary control points $b4d1b$, $b4d1a$, which ensure line forms 60° angle at B4 and is perpendicular at domain edge. ($3 \times 60^\circ + 2 \times 90^\circ$)
- $B5 \rightarrow D2$, 4 point bezier curve with intermediary control points $b5d2b$, $b5d2a$, which ensure line is perpendicular at both ends

3.3.2 Tuning nodes

To obtain high grid qualities, especially for highly inclined NGVs, the mesh has 4 user definable mesh tuning nodes. These are U1 and U2 (paired with D1 and D2) on the domain sides and I2 and I3 at the domain inlet. The position of these is defined using fractions of the respective lines where the nodes are positioned. E.g. U1D1_f defines the position of the U1 (and D1) as a fraction of the distance along the line $U3 \rightarrow I1$. A low value will place U1 close to inner radius of the domain and a high value will place U1 close the outer radius of the domain. The fractions are set in the following part of the code:

```
# Dimensions defining the mesh
bl = 0.5e-3           # (m) thickness of boundary refined layer around
                    vane
U1D1_f = 0.7         # fractiond defining position of U1 and D1
U2D2_f = 0.5         # fractiond defining position of U2 and D2
I2_f = 0.3           # fractiond defining position of I2
I3_f = 0.9           # fractiond defining position of I3
```

- U1D1_f, position of U1 (and D1) along line $U3 \rightarrow I1$.
- U2D2_f, position of U2 (and D2) along line $U3 \rightarrow I1$.
- I2_f, position of I2 along line $I1 \rightarrow I4$.
- I3_f, position of I3 along line $I1 \rightarrow I4$.

3.3.3 Cell numbers

The cell numbers are defined by the following variables:

```
# Define the number of cells:
Nu = 32
Nd = 32
Nt = 12
NBL = 7
Nb1 = 53
Nb2 = 13
Nb3 = 18
Nz = 10
```

The variables correspond to the following:

- Nu, number of cells in the tangential direction for blocks BU0, BU1, and BU2.
- Nd, number of cells in the tangential direction for blocks BD0, BD1, and BD2.
- Nt, number of cells in the tangential direction for block BT0.
- NBL, number of cells in the wall normal direction in the boundary layer blocks BL0, BL1, BL2, BL3, and BL4.

- **Nb1**, number of cells used to discretise the NGV surface in the wall parallel direction for BL0 and BL4.
- **Nb2**, number of cells used to discretise the NGV surface in the wall parallel direction for BL1 and BL3.
- **Nb3**, number of cells used to discretise the NGV surface in the wall parallel direction for BL2.
- **Nz** (3-D only), number of cells used of the page normal direction.

3.3.4 Cell recommendations

The code includes a simple utility to recommend cell numbers for the mesh generation. Cell number recommendations are automatically generated when running the code and provided as part of the on-screen output:

```

#####
The following numbers of cell settings are recommended
Adjust absolute number by changing N_mult
Current setting:      N_mult = 150 (number of cells along blade
                        surface)

Recommendations:
Nu = 32.0
Nd = 31.0
Nt = 30.0
NBL = ???, user selected
Nb1 = 50.0
Nb2 = 10.0
Nb3 = 30.0
Nz = ???, user selected

```

The recommendations are based on the value of the variable `N_mult`, which is set in the line:
`N_mult = 150 # use this N_multiplier to adjust number of cells.`

Using the above numbers, results in mesh with the following properties:

- mesh with `N_mult` cells equally spaced around the perimeter of the NGV
- mesh with cells in the inter NGV gap having same cell length as NGV perimeter cells
- cell numbers for boundary layer blocks need to be adjusted manually to correct ensure smooth transition (see 3.3.5)

By adjusting `N_mult` and re-running the code, new recommendations for finer or coarser meshes can be generated.

3.3.5 Changing cell clustering

Clustering can be adjsuted in the `### Set Cluster Functions ###` part of the code.

NGV Boundary Layer The clustering of the boundary layer blocks wrapped around the NGV can be adjusted by adjusting the `XX` parameter in the following two lines:

```
CF_b10 = RobertsClusterFunction(0,1,XX)
CF_b11 = RobertsClusterFunction(1,0,XX)
```

The value of `XX` for both lines needs to be identical, as the directions change between adjacent blocks in the boundary layer. Varying this parameter in conjunction with the number of cells in the boundary layer, `N_b1`, the boundary layer block height, `b1` allows the boundary layer mesh to be refined and integrated into the far-field mesh.

Z-direction (3-D only) Clustering in the Z-direction, i.e. close to the top and bottom wall can be adjusted by varying the `XX` parameter in the following line:

```
CF_h = RobertsClusterFunction(1,1,XX)
```

3.3.6 Defining Flow Properties

This is not covered here. See specific simulation code instructions for details. If using `Eilmer` the boundary conditions can be set in under the `### define B/C ###` section of the code (see [1]). If using `OpenFoam` the boundary faces are pre-labeled as follows for use with `e3prepToFoam` [4]:

`OF_outlet_00` Boundary at inner radius. Typically outlet.

`OF_inlet_00` Boundary at outer radius. Typically inlet.

`OF_inlet_01` Boundary on left side. First periodic interface.

`OF_inlet_02` Boundary on right side. Second periodic interface. This is fully matched to `OF_inlet_01`.

`OF_wall_00` Surface of NGV.

`OF_wall_01` (3-D only). Top wall of channel.

`OF_wall_02` (3-D only). Bottom wall of channel.

3.4 Example

The following section shows an example mesh generated by for a radial inflow turbine NGV. The geometry and mesh properties of the NGV are summarised in table 1. The example can be generated by executing `./run_NGV.sh`.

3.4.1 Set-up file

The mesh is initially generated in 2-D to ensure computationally efficient generation of and easy review of the main mesh topography. Once the 2-D topography has been set, as a final step the mesh is extruded in the z-direction to generate a 3-D mesh (see 3.4.2). The relevant code from the set-up file, to set the properties from Table 1 is:

Table 1: NGV geometry used for example case

Fluid Domain			
Domain outer radius	0 mm	Domain inner radius	0 mm
Number of blades	0		
NGV Geometry			
Leading edge centre radius	0 mm	Leading edge radius	0 mm
Trailing edge width	0 mm	Height	3 mm
Mesh			
Thickness of boundary layer blocks	0 mm	Nu	32
Nd	32	Nt	12
NBL	7	Nb1	53
Nb2	13	Nb3	18
Nz	10		
Tuning Parameters			
U1D1_f	0.7	U1D1_f	0.5
I2_f	0.3	I3_f	0.9

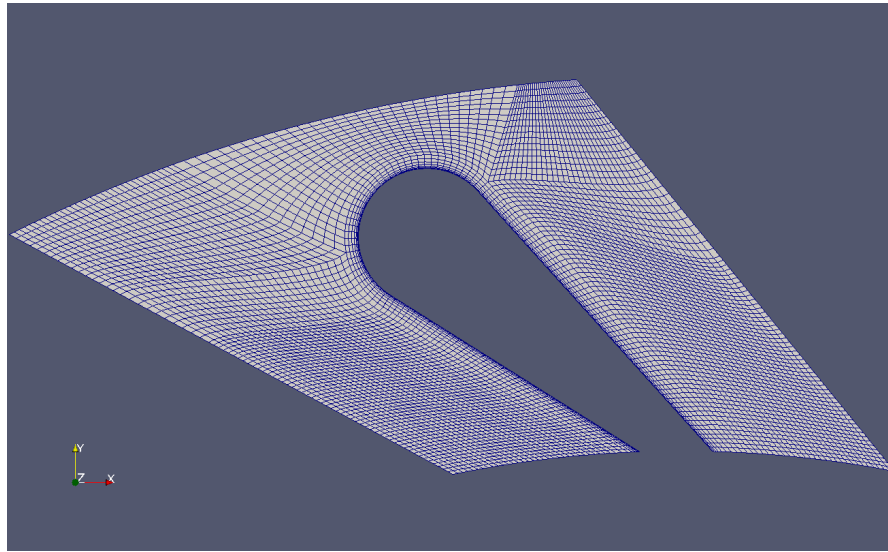
```
#####
### Defining the Geometry and Mesh ###
#####

# Dimensions defining the fluid Domain
R_in = 33.e-3      # (m)
R_out = 24.8e-3   # (m)
N_blade = 8       # Number of blades
Height = 0.003   # (m)

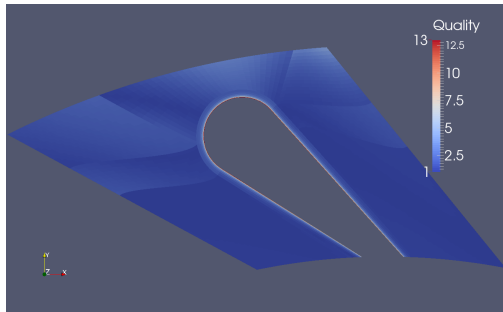
# Dimensions defining the Vane
R_leading = 30e-3 # (m) position of Vane leading edge centre
alpha_trailing = 50./180*np.pi # (rad) Vane angle at trailing edge.
r_leading = 1.5e-3 # (m) radius of vane leading edge
r_trailing = 0.5e-3 # (m) thickness/radius of vane trailing egde

# Dimensions defining the mesh
bl = 0.3e-3      # (m) thickness of boundary refined layer around
                 vane
U1D1.f = 0.75    # fractiond defining position of U1 and D1
U2D2.f = 0.55    # fractiond defining position of U2 and D2
I2_f = 0.3       # fractiond defining position of I2
I3_f = 0.9       # fractiond defining position of I3
# bez...        # definition of Bezier control points.

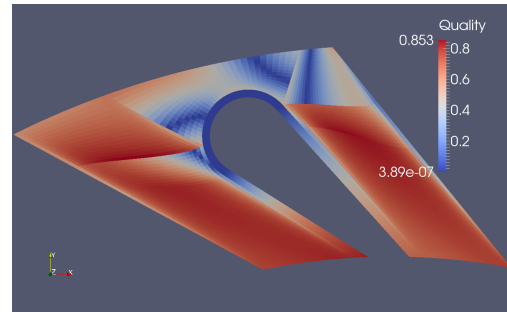
# Define the number of cells:
Nu = 32
Nd = 31
```



(a) 2-D mesh



(b) Cell Aspect Ratio



(c) Mesh skewness

Figure 4: 2-D mesh generated for NGV using example data.

$N_t = 16$
 $N_{b1} = 7$
 $N_{b1} = 50$
 $N_{b2} = 10$
 $N_{b3} = 30$
 $N_z = 10$
 $N_{mult} = 150$ # use this $N_multiplier$ to adjust number of cells.

The resulting mesh, mesh aspect ratio, and mesh skewness are shown in Figure 4. While overall a good quality, the mesh shows high skewness close to the inner edge, of the domain, which is caused by the high angle between the NGV and the inner edge. However with a maximum skewness parameter of 0.85 the mesh is still within acceptable limits, This mesh can be further improved by increasing the overall cell numbers.

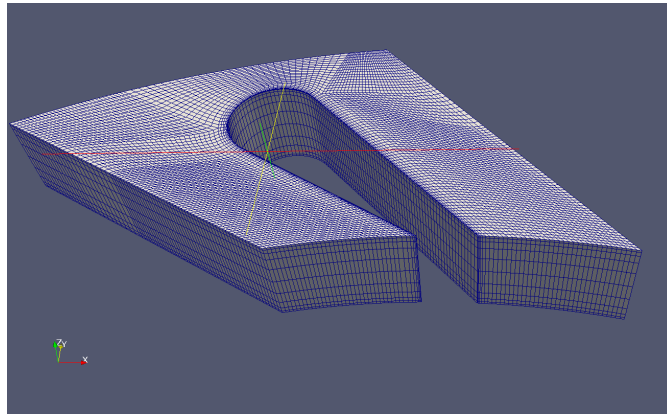


Figure 5: 3-D mesh

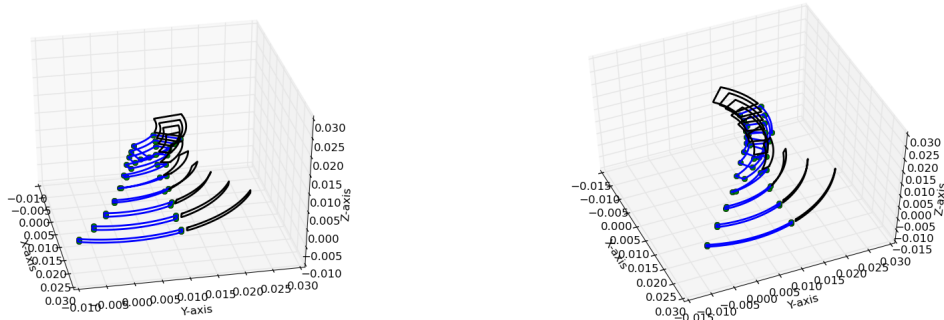
3.4.2 Conversion to 3-D

To generate a 3-D mesh the variable `gdata.dimensions` needs to be set to 3.

For grid development, set `gdata.dimensions = 2`, this will create the 2-D projection of the mesh.

```
gdata.dimensions = 3  
gdata.axisymmetric_flag = 0
```

Once this has been changed, executing `./run_NGV.sh` will generate the 3-D mesh shown in figure 5.



(a) Low Wrap Angle, $\theta_{wrap} = 40^\circ$

(b) High Wrap Angle, $\theta_{wrap} = 80^\circ$

Figure 6: Comparison of two identical rotor geometries with different wrap angles. ($\theta_{in} = -21^\circ$, $\theta_{out} = 65^\circ$, $R_{in} = 28.44 \times 10^{-3} \text{ m}$, $R_{out} = 12.5 \times 10^{-3} \text{ m}$, $Z = 12 \times 10^{-3} \text{ m}$, $A_{in} = 3.9 \times 10^{-4} \text{ m}^2$, $A_{out} = 5.6 \times 10^{-4} \text{ m}^2$)

4 ROTOR

4.1 New approach to Definition of Rotor Shape

Traditionally the shape of aerodynamic passages on a radial turbine are defined using a hub contour, a shroud contour, wrap angle, and local blade angle β , defined with respect to the passage meridional length. While such a geometry definition is logical from a structural, manufacturing, and mechanical design point of view (i.e. blades features are attached to the hub), this approach is not ideal for aerodynamics design. When designing and optimising the aerodynamic design parameters such as local flow direction, flow area (normal to flow direction), wetted rotor area, wetted shroud area, and how these evolve along the meridional length of the passage are the parameters of interest.

Particularly when comparing performance of turbomachinery, the traditional approach can create significant challenges. For example two designs with identical hub and shroud contours and inlet and exit flow angles can be generated with two different wrap angles as shown in Fig. 6. Naturally their performance will be different and one may conclude that this is a consequence of wrap angle. However, the underlying physical effects causing this effect are the differences in passage length (affecting frictional losses) and the changes in flow area evolution (affecting rate of expansion) between the two different designs. Thus to get a good understanding of the interaction between the fluid flow, the turbine rotor, and the resulting performance it is more intuitive and direct to work with aerodynamic parameters. The direct approach is also favourable from an optimisation point of view, as it allows a more direct parameterisation of the design space using parameters that directly affect performance.

The advantages outlined above in combination with recent advances in manufacturing, that allow even the most abstract geometries to be manufactures, for example by using precision 5-axis machining or advanced powder deposition methods (e.g. laser sintering), make geometry definition based on aerodynamic shapes an appealing alternative. The following sections outline a new geometry definition approach, which defines rotor geometry based on three features, also illustrated in Fig. 7:

Meridional Streamline This line in 3-D defines the direction the flow has to follow as it

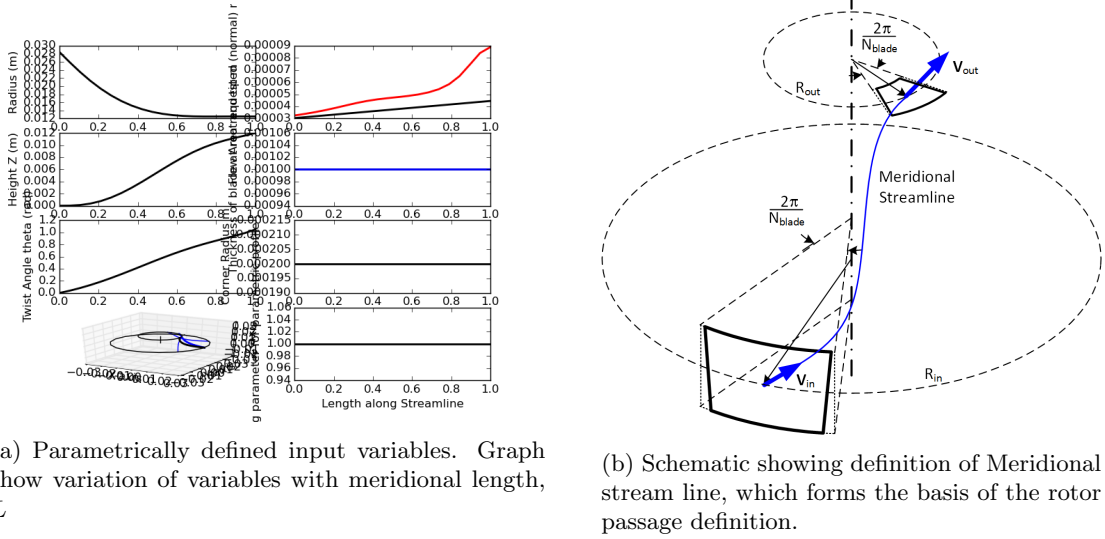


Figure 7: Definition of Parameters used to define rotor aerodynamic shape.

passes through the rotor passage. Effectively it defines r , $theta$, and z , the coordinates of the passage centre as a function of position along the passage. The position in the passage is defined by the normalised meridional variable L . $L = 0$ corresponds to the rotor inlet, $L = 1$ corresponds to the rotor outlet.

Evolution of Area The change in area normal to the flow direction, A_n , along the the streamline defines how quickly the flow expands and thus how the velocity evolves along the rotor passage.

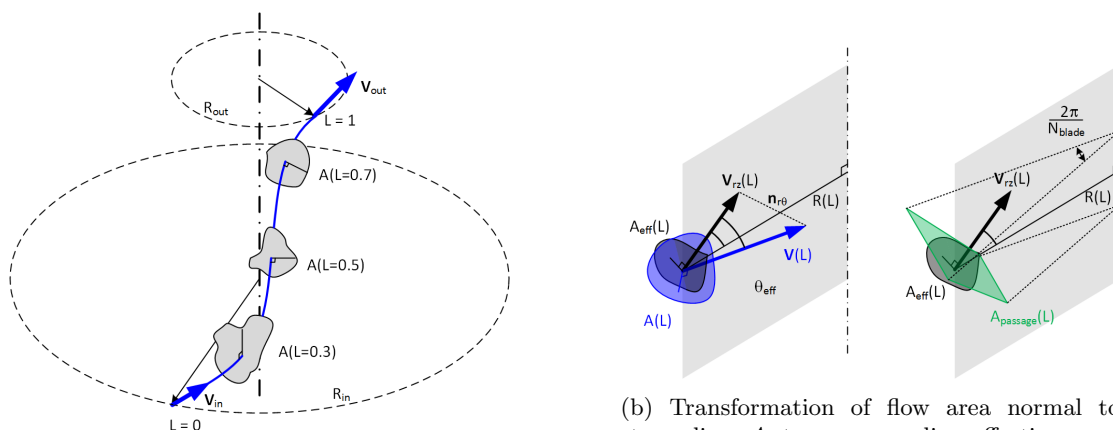
Parametric Passage Cross-section This features defines a parameterised shape used to generate the passage shape based on the streamline and normal area, A_n . Fig. 7 shows a simple template. More complex templates as outlined in section 4.4 are possible to enhance performance.

A further advantage of this approach is that the number of design variables is reduced to two design variables that can be described by functions (e.g. polynomials or Bezier curves) and a third variable with discrete values. The resulting reduction in variables, compared to more traditional ways of rotor geometry definition and the fact that variables are directly linked to aerodynamic effects, make this approach highly suitable for optimisation.

4.2 Process Overview

This section provides details on how the rotor and shroud geometry is generated. The stages of the process are summarised in the following list.

1. Define shape of meridional streamline. (Defined by parametric curve in 3-D)
2. Define flow normal area, A_n . (Defined by parametric function)
3. Construct streamline and calculate relative projection, $theta_{eff}$, which allows effective flow area, A_{eff} to be calculated.



(a) Flow Area, A defined at various positions along the Meridional Streamline.

(b) Transformation of flow area normal to streamline, A , to corresponding effective area, A_{eff} , normal to vector V_{rz} .

Figure 8: Construction of parameterised passage. The streamline normal area, A is transformed into an equivalent effective area, A_{eff} , which must be accommodated in the available passage area, $A_{passage}$.

4. Perform area correction to account for blade thickness. (Blade thickness is defined by parametric function)
5. Select parametric passage cross-section (see section 4.4). (Parametric shape specific variables can be defined by parametric functions)
6. Use parametric passage shape to construct rotor hub and blade shapes.
7. Generate shroud shape. (Rotor-stator clearance is defined by parametric function)
8. Export rotor and shroud shape.
9. Meshing.
 - (a) Define boundary layer refinement blocks.
 - (b) (optional) Add inlet mesh.
 - (c) (optional) Add NGV-rotor cavity mesh

The key steps of the above list are stages 1, 2, and 6. Based on the meridional streamline shape and desired evolution of flow area along the parametric passage shape is used to generate the rotor geometry.

4.3 Definition of Passage Shape

The passage shape is constructed piece-wise along the meridional streamline. Fig. 8(a) shows the meridional streamline and the corresponding normal flow area, A_n , and the local direction vector. To utilise the parametric passage shapes, the normal flow area is projected into a revolved surface as shown in Fig. 8(b). The advantage of using this revolved surface space, is that the available annular area can easily be distributed between multiple blade passages that exist around the

rotor. To construct this area, first the radial vector \mathbf{V}_{radial} is found and then the vector normal to the r, θ plane, $\mathbf{n}_{r\theta}$ is calculated as

$$\mathbf{n}_{r\theta} = |\mathbf{n}_z \times \mathbf{V}_{radial}| \quad (1)$$

Next the vector, $\mathbf{V}_{r\theta}$, the projection of the local streamline directional vector on the local r, z plane is found as

$$\mathbf{V}_{r\theta} = \mathbf{V} - |\mathbf{V} \cdot \mathbf{n}_{r\theta}| \mathbf{n}_{r\theta}. \quad (2)$$

This vector is the plane normal for the revolved surface. Once known, the inclination angle between the normal flow area, A_n (normal to streamline) and the projected flow area, A_{eff} (projection onto revolves surface) is calculated as

$$\theta_{eff} = \cos^{-1}(\mathbf{V} \cdot \mathbf{V}_{r\theta}) \quad (3)$$

Using θ_{eff} the required flow area A_{eff} on the revolved surface is calculated

$$A_{eff} = \frac{A_n}{\cos \theta_{eff}} = \frac{A_n}{\mathbf{V} \cdot \mathbf{V}_{r\theta}}. \quad (4)$$

At this stage the parametric passage cross-sections (see section 4.4), defined on the revolved surface are used to construct a local slices of the blade passage. The parametric passage shape is centred on the local streamline coordinate and the angular segment that can be used to construct the passage is defined by the number of blades ($\theta_{passage} = \frac{2\pi}{N_{blade}}$). Geometric features, such as blade thickness and corner radii that affect the available flow area within a given angular segment are incorporated in the definition of the parametric shape. Effectively in this stage a parametric shape, centred on the local streamline coordinate is fitted into the available angular segment. By appropriately defining the parametric shape, the full passage geometry is constructed.

As will be apparent in the next section, the normal flow area, A_n defined on a flat surface is transformed to an area defined on a revolved surface. This is particularly apparent close to the inlet of a radial in-flow turbine, where the flow direction is purely radial. This treatment, which implies that not all the flow crossing the revolved surface is parallel to the meridional streamline is appropriate as particularly close to the inlet, the actual flow purely radial and thus perpendicular to the revolved surface.

4.4 Parametric passage Cross-sections

As described above, core step of the current approach is to define the a parametric passage cross-section shape, which can be used to convert the effective flow area, A_{eff} into a corresponding rotor and blade shape at various points along the meridional streamline. For this purpose a number of pre-defined parametric geometries have been generated. These are listed in Table 2. The following sections give further details about the various parametric shapes.

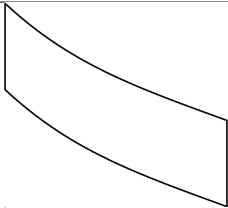
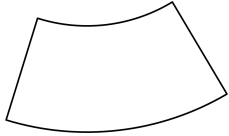
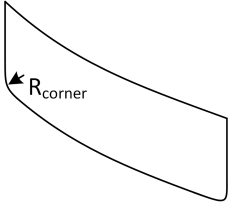
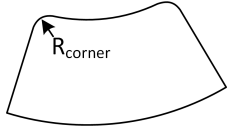
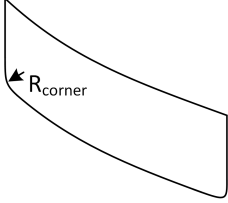
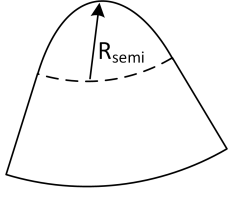
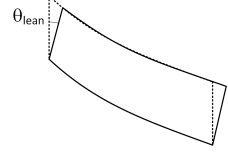
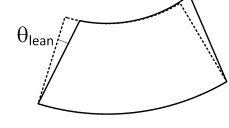
4.4.1 Rectangle to annular segment

This parametric surface is defined relative to its centre, (black dot), as shown in Fig. 9(a) The shape is constructed as follows:

1. Area occupied by rotor blade, defined in terms of passage height, H_{pass} is added to flow area

$$A_{passage} = A_{eff} + \frac{1}{2} (t_{root} + t_{tip}) H_{pass} = \frac{A_n}{\cos \theta_{eff}} + \frac{1}{2} (t_{root} + t_{tip}) H_{pass} \quad (5)$$

Table 2: Options for defining the passage cross-section between inlet and outlet

Option	Passage shape	
	Rotor inlet (looking radially inwards)	Rotor outlet (looking in axial direction)
Rect → Anul		
Rect → Anul (with corner radii)		
Semi-circle		
Lean option		

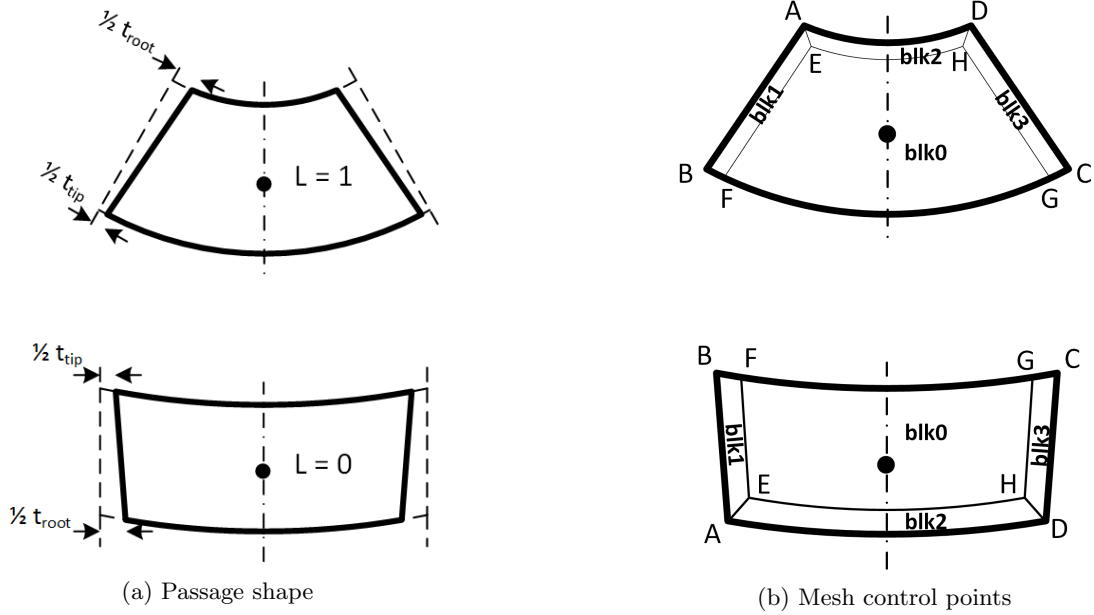


Figure 9: Rect \rightarrow Anul, definition of parametric surface geometry.

2. Required height of the passage is calculated by solving the following the equation

$$A_{passage} = \int_0^{\theta_{passage}} \int_{h=-\frac{H_{pass}}{2}}^{h=+\frac{H_{pass}}{2}} r(h) dr d\theta, \quad (6)$$

$$\text{where } \theta_{passage} = \frac{2\pi}{N_{blades}}.$$

In addition to defining the passage shape, internal control points to define the region of the mesh used for boundary layer refinement are also created as shown in the Fig. 9(b). The control points are placed a distance BL , defined by a parametric function, from the outer surfaces.

Function:

$\text{Rect}(\text{STREAMLINE}, \text{AREA}, \text{T_ROOT}, \text{T_TIP}, \text{BL}, \text{N_BLADE}, \text{LEAN})$, where STREAMLINE is a 3-d parametric path object, AREA , T_ROOT , T_TIP , BL , and LEAN are 1-d parametric path objects, and N_BLADE is a constant.

4.4.2 Rectangle to annular segment with corner radii

This parametric surface is defined relative to its centre, (black dot), as shown in Fig. 10(a) The shape is constructed as follows:

1. Area occupied by rotor blade, defined in terms of passage height, H_{pass} and area occupied by corners is added to flow area

$$A_{passage} = A_{eff} + \frac{1}{2} (t_{root} + t_{tip}) H_{pass} + R_c^2 \left(1 - \frac{\pi}{4}\right) = \frac{A_n}{\cos \theta_{eff}} + \frac{1}{2} (t_{root} + t_{tip}) H_{pass} \cos \theta_{eff} + R_c^2 \left(1 - \frac{\pi}{4}\right) \quad (7)$$

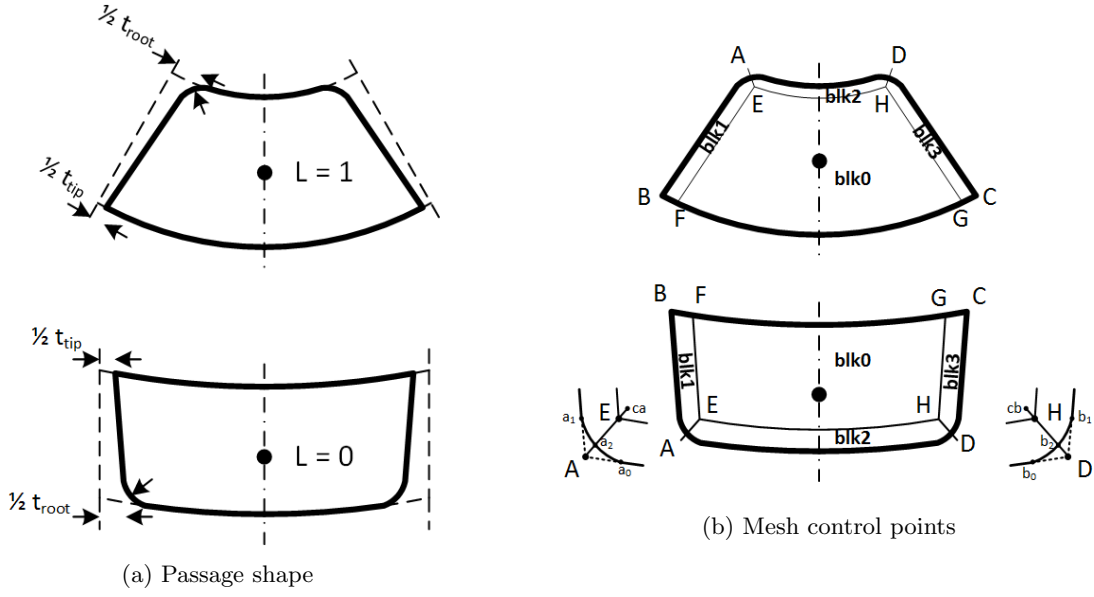


Figure 10: Rect \rightarrow Anul (with corner radii), definition of parametric surface geometry.

2. Required height of the passage is calculated by solving the following the equation

$$A_{passage} = \int_0^{\theta_{passage}} \int_{h=-\frac{H_{pass}}{2}}^{h=+\frac{H_{pass}}{2}} r(h) dr d\theta, \quad (8)$$

where $\theta_{passage} = \frac{2\pi}{N_{blades}}$.

In addition to defining the passage shape, internal control points to define the region of the mesh used for boundary layer refinement are also created as shown in the Fig. 10(b). The control points are placed a distance BL , defined by a parametric function, from the outer surfaces.

Function:

Rect(STREAMLINE, AREA, T_ROOT, T_TIP, BL, RC, N_BLADE, LEAN), where STREAMLINE is a 3-d parametric path object, AREA, T_ROOT, T_TIP, BL, RC, and LEAN are 1-d parametric path objects, and N_BLADE is a constant.

4.4.3 Rectangle to semi-circular segment

This parametric surface is defined relative to its centre, (black dot), as shown in Fig. 11(a) The shape is constructed as follows:

1. A circle is constructed, which forms tangent lines with both the passage side walls.
2. Area occupied by rotor blade, defined in terms of passage height, H_{pass} and area occupied by corners is added to flow area

$$tbc \quad (9)$$

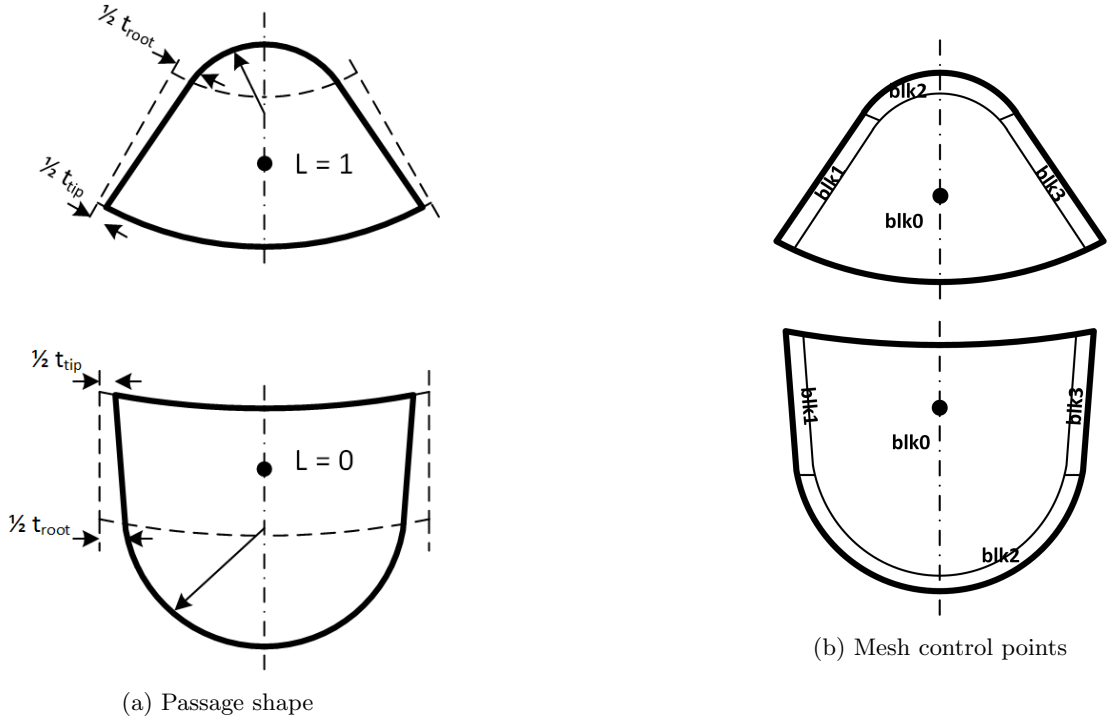


Figure 11: Semi-circle, definition of parametric surface geometry.

3. Required height of the passage is calculated by solving the following the equation

$$tbc \tag{10}$$

where $\theta_{passage} = \frac{2\pi}{N_{blades}}$.

In addition to defining the passage shape, internal control points to define the region of the mesh used for boundary layer refinement are also created as shown in the Fig. 11(b). The control points are placed a distance BL , defined by a parametric function, from the outer surfaces.

Function:

SemiCirc(STREAMLINE, AREA, T_ROOT, T_TIP, BL, N_BLADE, LEAN)

4.4.4 Blade lean

A further feature of all the above parametric shapes is the inclusion of blade lean. This allows the blade profiles to be inclined relative to the radial direction as shown in Fig. 12(a). Blade lean is incorporated by applying the angle θ_{lean} to the rotor blades. The angle is defined using a 1-D path object which defines the change in lean along the passage.

4.4.5 Profile Blending

To generate more complex rotor geometries it is desirable to create combinations of the parametric shapes defined in section 4.4. To allows this a blending function has been generated. This

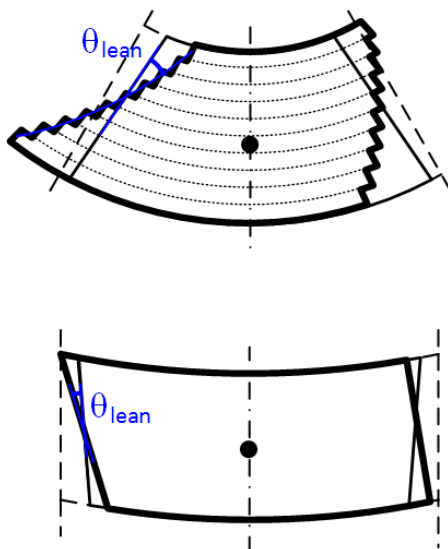


Figure 12: Sketch showing incorporation of blade lean into geometry generation

function is based on a blending parameter, $blend$, defined in terms of position along the meridional streamline and creates a linear combination of the respective parametric profiles as follows:

$$Profile_{out} = (1 - blend) \times Profile_1 + blend \times Profile_2 \quad (11)$$

By defining $blend$ as a continuous function this ensures a smooth transition of the shape.

Function:

`Blended2Dsurface(Profile_1,Profile_2,blend)`, where `Profile1` and `Profile2` are parametric surface objects and `blend` is a 1-d path object.

4.5 Running the Tool

Follow these steps:

1. Define the passage geometry in the file `Rotor_example.py`
2. To display the rotor passage geometry and to generate a geometry for export to CAD
 - (a) Run `Rotor_Profile.py --job=Rotor_example.py`
 - (b) View displayed rotor passage properties
 - (c) View generated passage shape
 - (d) To generate geometry files, set appropriate variable inside `Rotor_example.py`
3. To generate the mesh
 - (a) Adjust mesh property and mesh clustering parameters in `Rotor_Profile.py`
 - (b) Generate mesh by running `e3prep.py --job=Rotor_Passage.py`
(if renaming `Rotor_example.py`, ensure correct file is referenced in `Rotor_Passage.py`)

- (c) Run following commands to convert mesh to Vtk format and view in paraview:


```
e3post.py --job=Rotor_Passage.py --vtk-xml
paraview
```
 - (d) Iterate to improve mesh quality by adjusting thickness of boundary layer blocks inside `Rotor_example.py`, or by adjusting clustering and number of cells in `Rotor_Passage.py`
4. Convert mesh to other format for further processing
 5. If using Eilmer for flow simulation, the job file `Rotor_Passage.py` needs to be adjusted to set appropriate simulation parameters.

4.6 Geometry Generation

Generation of the rotor and shroud geometry is performed using the standalone python programme `Rotor_Profile.py`, which use geometry parameters defined in the geometry definition file `Rotor_example.py` to generate multiple slices along a single (or multiple) rotor passage. This allows efficient generation of candidate rotor passages, which can either be turned into output files for import to CAD packages or which can be further processed into a CFD mesh as described in section 4.7.

The rotor geometry is generate by executing:

```
Rotor_Profile.py --job=Rotor_example.py
```

where `Rotor_example.py` is a geometry definition file that is used to set up the rotor geometry and to define parameters for the output files. The following sections describe how the rotor geometry is defined and the corresponding outputs that are provided.

4.6.1 Setting up the Rotor_example.py

The rotor geometry is defined in the `Rotor_example.py` file. A typical file consists of the following three main parts:

Streamline Definition

```
#####
# Setting Streamline at Passage Centre
#####

# Define Central Streamline that is used to set blade passage shape.
# STREAMLINE must be a 3-D path function as described at the end
# STREAMLINE = Topgen2Bezier(R_in, theta_in, R_out, theta_out, Z_out,
    Twist, L_in2, L_in3, L_out4, L_out5)
STREAMLINE = Bezier_3D ([ (x0,y0,z0), (x1,y1,z1), ... (x6,y6,z6)])
```

Here the a line in 3-D space is defined, which forms the streamline at the centre of the passage. This line can be a 3-D Bezier curve, or a function to read-in an output from TopGen [5], `Topgen2Bezier` has also been set-up. The use of this function is illustrated and explained in the example provided in section 4.8.

As a minimum, a 3-D path object with the name `STREAMLINE` has to be created.

Parametric Curve Definition

```
#####  
# Setting Parametric curves to define passage  
#####  
  
# Define parametric evolution of area  
# AREA must be a 1-D path function  
AREA = Poly_1D((A0,A1))  
  
# set corner radius and boundary layer refinement height  
# RC must be a 1-D path function  
RC = Const_1D(0.001)  
# BL must be a 1-D path function  
BL = Const_1D(0.0002)  
  
# set Blade thickness at root and tip  
# T_ROOT and T_TIP must be a 1-D path function  
T_ROOT = Const_1D(0.001)  
T_TIP = Const_1D(0.001)  
  
# set lean of rotor blades  
# LEAN must be a 1-D path function  
LEAN = Const_1D(0.)  
  
# set number of Blades  
N_BLADE = 9
```

Here 1-D line objects are defined, which set the evolution of various line parameters along the length of the passage.

As a minimum, 1-D path objects with the following names: AREA, T_ROOT, T_TIP, BL and the integer variable N_BLADE have to be created. The objects RC, BLEND and E_RATIO are optional and depend on the type of parametric cross-section that is used.

Passage Shape Creation

```
#####  
# Setting Parametric Profile  
#####  
  
# Define parametric profile used to generate passage shape  
# PROFILE must be a 2-D profile object, which contains sub-division  
  into 4 grid-able blocks  
Surf1 = Rect(STREAMLINE,AREA,T_ROOT,T_TIP,BL,N_BLADE,LEAN)  
Surf2 = RectCorner(STREAMLINE,AREA,T_ROOT,T_TIP,BL,RC,N_BLADE,LEAN)  
  
# define blending function  
# BLEND must be a 1-D path function varying between 0. and 1.
```

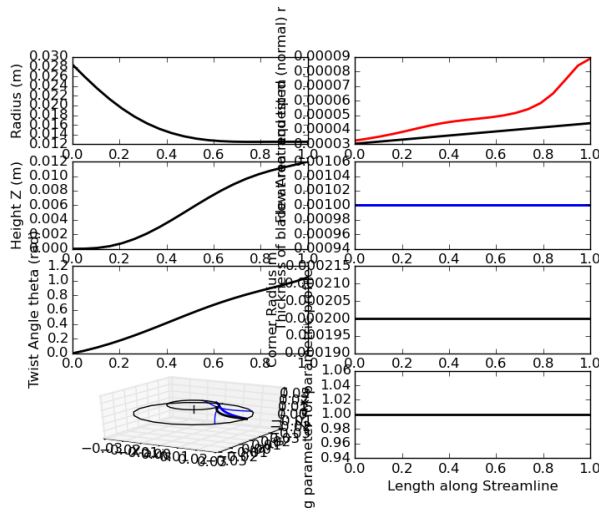



Figure 13: Parametrically defined input variables. Graph show variation of variables with meridional length, L

```
# BLEND = Poly_1D((0., 1.))
BLEND = Const_1D(1.)
```

```
# Assemble Profile
PROFILE = Blended2Dsurface(Surf1 , Surf2 , BLEND)
```

This last section is used to combine the streamline and various parametric objects that have been combined to create one of the pre-defined parametric passage cross-sections defined in section 4.4. In the above case two surface objects **Surf1** and **Surf2** are created and blended to form the final cross-section object **PROFILE**.

The final profile object that is created must be named **PROFILE**.

4.6.2 Graphical Displays

After running the program `Rotor_Profile.py --job=Rotor_example.py` a number of graphical outputs are generated. These are collated in three figures.

Figure 1 13 This displays eight graphs showing the input parameters as well as the calculated inout parameters. The left column shows inputs for the streamline defining the rotor passage. These are the evolution of radius, angular position, and height z with meridional position L , as well as 3-D and 2-D projection of the streamline in 3-D. The right column shows the requested flow area A_n , and the calculated flow area A_{eff} , together with other setting parameters and how they vary with meridional position.

Figure 2 14 This 3-D projection of multiple cross-sectional slices through the passage. In addition to showing the perimeter of the slices, the position of the nodes used to define the boundary layer blocks is marked also. The number of slices and the number of adjacent channels that is plotted can be defined in `Rotor_example.py`

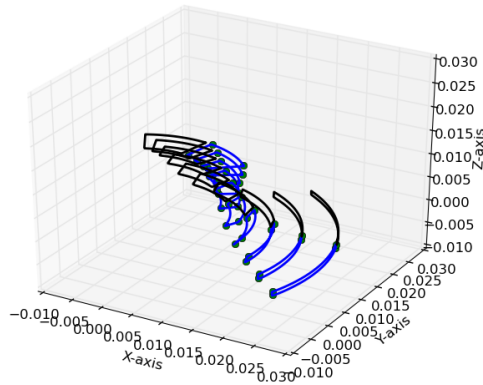


Figure 14: Rotor Shape, illustrated by multiple slices through passage.

Figure 3 15 This shows a 2-D slice through the rotor in the r, z plane. The lines show the profile of the hub, blade tips, and shroud.

Graphical Display Settings The settings for the display are adjusted in the following section of `Rotor_example.py`

```
#####
# Setting Visulisation Properties
#####
# set Visulisation Flag
VISUAL.flag = 1

# Set Properties for Showing Profile in python window
VISUAL.slices = 10
VISUAL.channels = 2
VISUAL.nodes = 40
```

`VISUAL.flag` defines if data will be displayed;
`VISUAL.slices` sets how many channel cross-sections will be displayed;
`VISUAL.channles` defines how many side-by-side channels are displayed;
`VISUAL.nodes` sets the resolution used when creating the perimeter lines.

4.6.3 Output Files for CAD

Optionally the following section of code can be added to the `job_rotor.py` file. By setting `FILES.flag = 1` this will invoke the creation of a number of output files, which can be used to generate coordinate files for import to CAD software.

```
#####
```

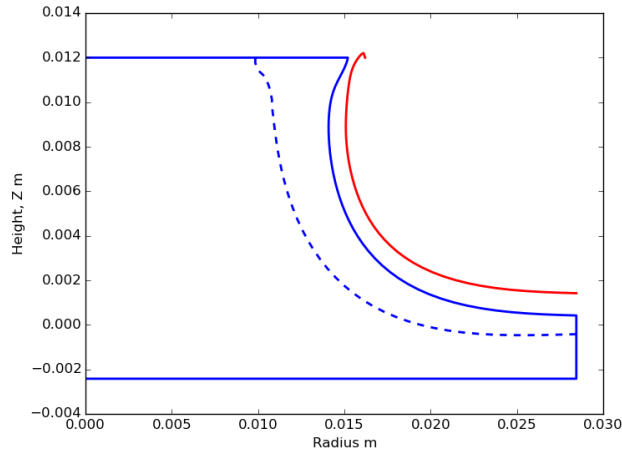


Figure 15: Generate Rotor and Shroud Profile

```
# Define Output Files
#####
# set file writing Flag
FILES.flag = 0

# set Filenames
FILES.nameroot = "Dat_"
FILES.Slices = 10
FILES.Points1 = 20
FILES.Points2 = 5
```

The following files are generated to define the axi-symmetric profile of the rotor and the shroud:

Dat_hub.txt a file containing the r and z coordinates along the perimeter of the rotor blank. This consists of rotor back-side, defined by 1-D object **R_THICK**, inlet, blade tips, outlet and hub.

Dat_shroud.txt a file containing the r and z coordinates for the turbine shroud, positioned with a perpendicular off-set relative to the blade tip defined by the 1-D path object **CLEARANCE**

And the following files are the generated to define the lines in 3-D space, which resemble the geometry of a single rotor passage. The value of **FILES.Slices** is used to set the number of slices used for extracting the rotor shape.

Dat_line0.txt a file containing the coordinates of the shroud facing edge of a passage profile. **Points2** defines the number of points used to discretise the boundary layer block segments (west and east edge of blk1 and blk3). **Points1** defines the points along the central block. Total number of points per line is **Points2 + Points1 + Points2**.

Dat_line1.txt points along the left hand side blade. Discretised **Points1** points.

Dat_line2.txt points along the bottom of the passage. Discretised **Points1** points.

Dat_line3.txt points along the right hand side blade. Discretised **Points1** points.

Once imported into a CAD package, the respective line segments can be used to reconstruct surfaces, which in turn form the full 3-D geometry of the rotor. Care must be taken to discretise the passage profiles sufficiently, so that small features, such as the corner radii are correctly re-created.

4.7 Mesh Generation

The mesh is generated using the tool **e3prep** using the pre-prepared job-file **Rotor_Passage.py**. This creates a structured mesh, based on the rotor geometry defined in the file **Rotor_example.py**.

The mesh generation is performed by running **e3prep.py --job=Rotor_Passage.py**. Details on using the **e3pre** and how to perform general modifications to the file **Rotor_Passage.py** can be obtained from the Eilmer User Guide [1]. The following paragraphs describe modifications to the file of specific relevance for using in conjunction with the rotor meshing tool **Rotor_Profile.py**.

4.7.1 Rotor Profile Definition

The rotor shape is generated by linking importing the **Rotor_Profile** as a module and then executing the file containing the rotor shape definition. This is the same file as can be used as the *job* file when executing **Rotor_Profile.py --job=job**. In the code below this is **Rotor_example.py**. The subsequent lines are used to generate anonymous functions, which can be called by **PyFunctionPath()** when constructing the blocks.

```
from Rotor_Profile import *

#####
### Set Rotor Properties      ###
#####

# set filename used to store rotor data. Data can be previewed using
  Rotor_Profile.py --job=name
RotorFileName = "Rotor_example.py"

# execute file containing rotor data to define geometry
execfile(RotorFileName, globals())
# create anonymous functions that can be used by PyFunctionVolume()
pyfunction_blk0 = lambda r,s,t: PROFILE.eval(r,s,t,0)
pyfunction_blk1 = lambda r,s,t: PROFILE.eval(r,s,t,1)
pyfunction_blk2 = lambda r,s,t: PROFILE.eval(r,s,t,2)
pyfunction_blk3 = lambda r,s,t: PROFILE.eval(r,s,t,3)
```

4.7.2 Mesh Definition and Clustering

The cell numbers and mesh clustering are defined in this section

```
#####  
### Set Number of Cells ###  
#####  
n1 = 60 # cells along passage  
nt = 60 # cells of main passage in circumferential (tangential)  
direction  
nz = 60 # cells of main passage axial (at inlet) or radial (at  
outlet) direction  
nbl = 10 # cells in boundary layer region.  
  
#####  
### Set Cluster Functions ###  
#####  
CF_bl0 = RobertsClusterFunction(1,0,1.05)  
#CF_bl1 = RobertsClusterFunction(1,0,1.05)  
#CF_h = RobertsClusterFunction(1,1,1.05)
```

Here **n1**, **nt**, and **nz** defines the number of cells in the meridional, tangential and hub towards shroud direction of the block located in the passage centre (BLK0). The value of **nbl** sets the number of used to for boundary layer refinement in the wall normal direction.

Clustering can be achieved using the cluster functions. Currently only clustering in the boundary blocks in the wall normal direction is active. Other clustering can be added by defining further cluster functions and adding these to the respective block definitions.

4.8 Example

The following sections describes the generation of an example mesh, based on a geometry defined by the meanline turbine design code TopGen [5]. In addition to the meanline geometry parameters, defined by the TopGen, the twist angle needs to be selected by the user. For the current example the twist angle was determined by iteratively generating geometries with `Rotor_Profile.py` until a smooth profile was created that results in a continuous increase in properties of the streamline at the passage centre. The geometry defining properties are defined in table 3. In the table the extra parameters that have to be defined in addition to the outputs from meanline analysis are shown in **bold**.

The corresponding geometry definition file is given below. In this case the passage defining streamline, **STREAMLINE** is defined using the function `Topgen2Bezier`, which constructs a **Bezier_3D** curve based on the provided inputs. This function takes the geometry variables and the following tuning parameters to define the Bezier control points:

L_in2 and **L_in3**, are two control points positioned on a straight line defined by the inlet flow vector.

L_out4 is a control point at the same radius as the outlet, but rotated such that the line follows a cylindrical path.

L_out5 is a control point on the vector defined by the outlet flow direction. This ensures the correct flow direction is achieved at the outlet.

Table 3: Rotor geometry, obtained from meanline analysis, used for example case

Overall Dimensions			
Inlet Radius	$28.44 \times 10^{-3} \text{ m}$	Outlet Radius	$12.5 \times 10^{-3} \text{ mm}$
Rotor Height	12.00	Number of Blades	9
Flow Angles			
Inlet relative Angle	-21°	Outlet relative Angle	60°
Twist Angle	60°		
Flow Areas and Passage Features			
Inlet Area	$3.9 \times 10^{-4} \text{ m}^2$	Outlet Area	$6.2 \times 10^{-4} \text{ m}^2$
Inlet Normal Area	$4.17 \times 10^{-4} \text{ m}^2$	Outlet Normal Area	$18.13 \times 10^{-4} \text{ m}^2$
Area evolution	linear increase	Parametric Area Type	RectCorner
Blade Thickness (root)	1.0 m	Blade Thickness (tip)	1.0 m
Blade-Hub corner radius	0.2 m	Blade Lean	None
Mesh			
Thickness of boundary layer blocks			$0.2 \times 10^{-3} \text{ m}$
nl	60	nt	60
nz	60	nbl	10

For all control points the values correspond to the fractional distance (based on meridional length L), that these points are away from the inlet.

It should be noted that this file actually defines two parametric geometries, **Surf1** and **Surf2**. However by setting `BLEND = 1` using the `Const_1D` class the resulting output is **Surf2**.

```

# Rotor_example.py
# Ingo Jahn 11/05/2015
# Example Job file for creating Mesh

Name = "Rotor_meshing_example"
print "Running", Name, "_to_generate_rotor_mesh"

#####
# Setting Streamline at Passage Centre
#####
# Streamline defining passage (take values from TOPGEN)
R_in = 28.443e-3    # (m) radius at inlet
    TOPGEN --> radius_inlet
R_out = 12.5e-3    # (m) radius at outlet
    TOPGEN --> radius_outlet (mean)
Z_out = 12e-3    # (m) height at outlet (only used for ellipse) -->
    design variable
Twist = 60./180. * np.pi    # (rad) twist angle of streamline -->
    design variable
theta_in = -21./180. * np.pi    # (rad)
    TOPGEN --> beta_inlet
theta_out = 60./180. * np.pi    # (rad)
    TOPGEN --> beta_outlet

```

```

L_in2 = 0.3      # position of 2nd control point (fraction along
                 streamline from inlet) —> design variable
L_in3 = 0.6      # position of 3rd control point (fraction along
                 streamline from inlet) —> design variable
L_out4 = 0.8     # position of 4th control point (fraction along
                 streamline before outlet) —> design variable
L_out5 = 0.9     # position of 5th control point (fraction along
                 streamline before outlet) —> design variable

# Define Central Streamline that is used to set blade passage shape.
# STREAMLINE must be a 3-D path function as described at the end
STREAMLINE = Topgen2Bezier(R_in, theta_in, R_out, theta_out, Z_out,
                          Twist, L_in2, L_in3, L_out4, L_out5)

#####
# Setting Parametric curves to define passage
#####
# Area of Passage
A0 = 3.9e-4/12.*np.cos(21./180. * np.pi) # (m2) Area at inlet
A1 = 6.2e-4/12.*np.cos(70./180. * np.pi) # (m2) Area at outlet
A1 = 0.9*A1

# Define parametric evolution of area
# AREA must be a 1-D path function
AREA = Poly_1D((A0,A1))

# set corner radius and boundary layer refinement height
# RC must be a 1-D path function
RC = Const_1D(0.0002)
# BL must be a 1-D path function
BL = Const_1D(0.0002)

# set Blade thickness at root and tip
# T_ROOT and T_TIP must be a 1-D path function
T_ROOT = Const_1D(0.001)
T_TIP = Const_1D(0.001)

# set lean of rotor blades
# LEAN must be a 1-D path function
LEAN = Const_1D(0./180.*np.pi)

# set number of Blades
NBLADE = 9

#####
# Setting Parametric Profile
#####

# Define parametric profile used to generate passage shape

```

```

# PROFILE must be a 2-D profile object, which contains sub-division
  into 4 grid-able blocks
Surf1 = Rect(STREAMLINE,AREA,T_ROOT,T_TIP,BL,N_BLADE,LEAN)
Surf2 = RectCorner(STREAMLINE,AREA,T_ROOT,T_TIP,BL,RC,N_BLADE,LEAN)

# define blending function
# BLEND must be a 1-D path function varying between 0. and 1.
# BLEND = Poly_1D((0.,1.))
BLEND = Const_1D(1.0)

# Assemble Profile
PROFILE = Blended2Dsurface(Surf1 ,Surf2 ,BLEND)

#####
# Defining Rotor blank and Stator
#####
# set Clearance
CLEARANCE = Const_1D(0.001)

# Rotor back thickness
R_THICK = Const_1D(0.002)

#####
# Define Output Files
#####
# set file writing Flag
FILES.flag = 0

# set Filenames
FILES.nameroot = "Dat_"
FILES.Slices = 10
FILES.Points1 = 20
FILES.Points2 = 5

#####
# Setting Visulisation Properties
#####
# set Visulisation Flag
VISUAL.flag = 1

# Set Properties for Showing Profile in python window
VISUAL.slices = 10
VISUAL.channels = 2
VISUAL.nodes = 40

```

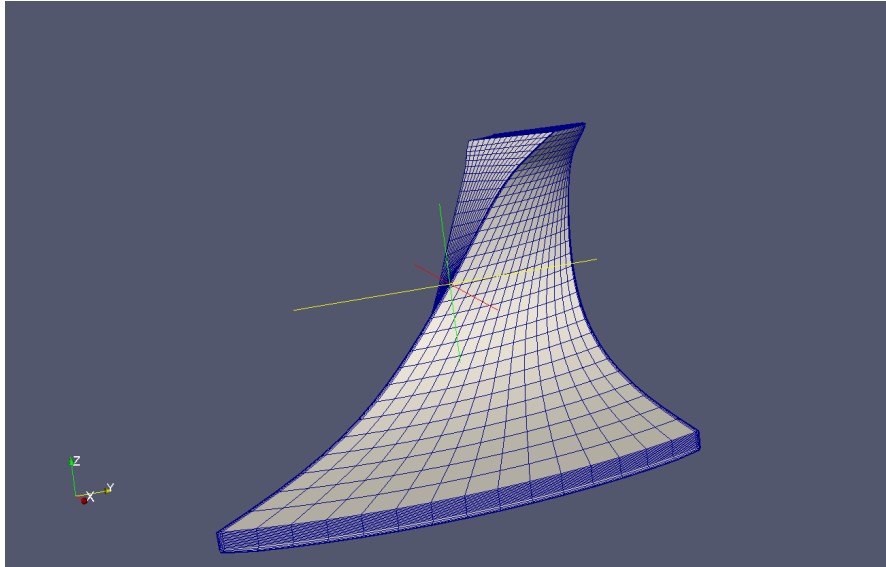
The results from `Rotor_Profile.py` are shown in Figs 13, 14, and 15. Even these simple results reveal some interesting features in relation to rotor design, such as the rapid increase in effective flow area, A_{eff} as $L \rightarrow 1$. This is caused by the fact that the effective flow area is a function of $\frac{1}{\cos\theta_{eff}}$ and that at the exit θ_{eff} equals the relative flow angle. Consequently as

the exit flow angle becomes large ($> 60^\circ$), the required effective flow area increases rapidly. To accommodate this, the rotor actually increases in outer radius at the exit, as shown in the rotor profile plot (Fig. 15). In contrast rotors designed with conventional design tools, which define a shroud and hub profile, the flow area evolution along meridional position would be highly nonlinear.

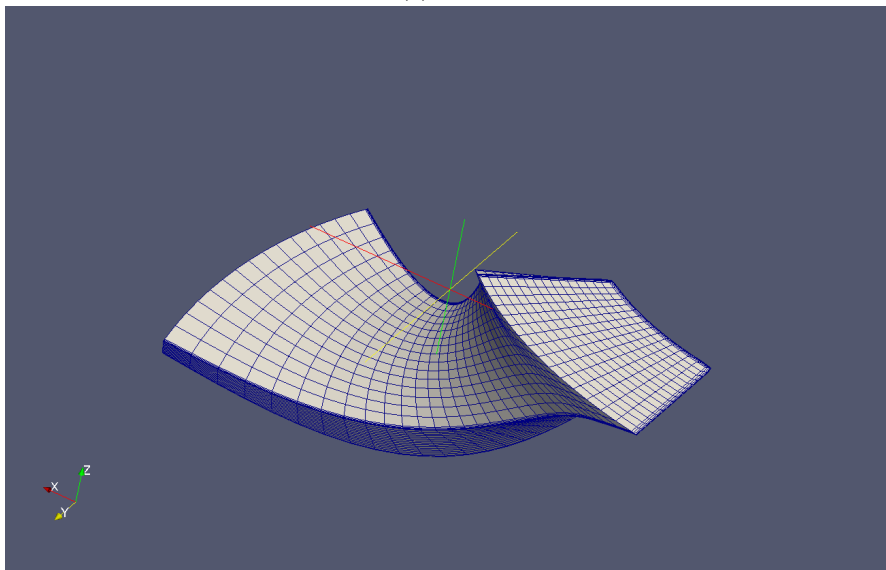
The corresponding mesh, generated using `e3prep.py` is shown in Fig. 16. This is a preliminary mesh without the implementation of significant mesh refinements.

5 Combined geometry

To conduct fully coupled and possibly unsteady CFD simulations of a full Radial-Inflow turbine the NGV mesh and the rotor mesh need to be coupled. Currently a range of coupling methods exist, such as mixing planes, or universal grid interfaces, that allow unsteady coupling between the two sections. To allow the user the most flexibility for constructing this mesh interface, currently no mesh is defined for the NGV - rotor gap. Future work will explore the options of extending both the NGV and rotor mesh towards the middle of the gap in order to create a mesh interface at mid position.



(a) Text1



(b) Text2

Figure 16: Preliminary Mesh generated using `e3prep`. Fine tuning of boundary layer clustering and position of control points that define boundary layer blocks can be used for further mesh improvements.

6 Conclusion and Planned Work

A new radial inflow turbine geometry generation and mesh definition tool has been developed. The Nozzle Guide Vane (NGV) part of the tool allows the generation of a specific vane geometry with a circular leading edge and a straight cut trailing edge. For the rotor two tools have been generated. First, a tool that allows the generation of a rotor geometry based on aerodynamic requirements. This tool can be used to rapidly generate desirable rotor geometries and these geometries can be extracted to CAD for further processing. Second, the same tool can be used to generate structured meshes suitable for CFD.

The key advantage of the approach taken for rotor geometry generation, is that the geometry is defined as a function of aerodynamic parameters, such as evolution of flow area (defined normal to flow direction) along the passage length. This allows a direct investigation of the rotor performance in terms of aerodynamic inlet parameters.

The following tasks to improve the capability of this open-source meshing tool are currently planned and in progress. Please contact the author if you want to support the development of these modules

- Development of additional parametric passage shapes. Planned passages currently under consideration are:
 - Upgrade of SemiCircle Profile to allow an elliptical base. (effectively a variable root radius)
- Addition of rotor-shroud clearance to mesh
- Addition of rotor blade inlet edges to mesh
- implementation of sliding grid interface for Eilmer
- other suggestions

References

- [1] P.A. Jacobs, R.J. Gollan, D.F. Potter, 2014, *The Eilmer3 Code: User Guide and Example-Book*, Mechanical Engineering Report 2014/05, The University of Queensland
- [2] OpenFOAM The Open Source CFD Toolbox, *Userguide*, Version 2.3.1, 3r December 2014, www.foam.sourceforge.net/docs/Guides-a4/UserGuide.pdf OpenFOAM Foundation
- [3] CFCFD, *The Compressible Flow Project* <http://cfcfd.mechmining.uq.edu.au> The University of Queensland
- [4] Ingo Jahn, Kan Qin, 2015, *e3prepToFoam: a mesh generator for OpenFOAM*, Mechanical Engineering Report 2015/04, The University of Queensland
- [5] Carlos A.M. Ventura, Peter A. Jacobs, Andrew S. Rowlands, Paul Petrie-Repar, Emilie Sauret, 2012, *Preliminary Design and Performance Estimation of Radial Inflow Turbines: An Automated Approach*, Journal of Fluids Engineering, MARCH 2012, Vol. 134, ASME

7 Code

7.1 Inlet_Vane_Round_Square.py

```
1### \Inlet_Vane_Round_Square.py
2#
3"""
4Script to create a structured mesh for an inlet guide vane as may be used
   for a radial in-flow turbine.
5The current file is designed for a a guide vane with a rounded leading edge
   and a trauling edge, cut at the exit of the domain.
6
7The grid is generated largely automatically. See XXX for corresponding
   diagrams and definitions.
8Currently the following locations have to be adjusted manually:
9- Positions of U1, U2, D1, D2 (as fraction of radial position)
10- Positions of I2, I3 (as fraction of angular position)
11- Bezier control points for the corresponding lines.
12
13Author: Ingo Jahn
14Last modified: 23/03/2015
15"""
16
17import numpy as np
18
19#####
20### Setting up Basic Information ###
21#####
22
23# For grid development, set gdata.dimensions = 2, this will create teh 2-D
   projection of the mesh.
24gdata.dimensions = 2
25gdata.axisymmetric_flag = 0
```

```

26
27 # Set some fluid propertied to allow e3prep to solve
28 # These only need to be correct if using Eilmer as solver.
29 select_gas_model(model='ideal gas', species=['air'])
30 initial = FlowCondition(p=5955.0, u=0.0, v=0.0, T=304.0)
31 inflow = FlowCondition(p=95.84e3, u=1000.0, v=0.0, T=1103.0)
32
33
34 #####
35 ### Defining the Geometry and Mesh ###
36 #####
37 # Dimensions defining the fluid Domain
38 R_in = 33.e-3 # (m)
39 R_out = 24.8e-3 # (m)
40 N_blade = 8 # Number of blades
41 Height = 0.003 # (m)
42
43 # Dimensions defining the Vane
44 R_leading = 30e-3 # (m) position of Vane leading edge centre
45 alpha_trailing = 50./180*np.pi # (rad) Vane angle at trailing edge.
46 r_leading = 1.5e-3 # (m) radius of vane leading edge
47 r_trailing = 0.5e-3 # (m) thickness/radius of vane trailing egde
48
49 # Dimensions defining the mesh
50 bl = 0.3e-3 # (m) thickness of boundary refined layer around vane
51 U1D1_f = 0.75 # fractiond defining position of U1 and D1
52 U2D2_f = 0.55 # fractiond defining position of U2 and D2
53 I2_f = 0.3 # fractiond defining position of I2
54 I3_f = 0.9 # fractiond defining position of I3
55 # bez... # definition of Bezier control points.
56
57 # Define the number of cells:
58 Nu = 32
59 Nd = 31
60 Nt = 16
61 NBL = 7
62 Nb1 = 50
63 Nb2 = 10
64 Nb3 = 30
65 Nz = 10
66 N_mult = 150 # use this N_multiplier to adjust number of cells.
67
68 #####
69 ### Helper Functions ###
70 #####
71 def M_calc(R_out, R_leading, alpha_trailing):
72     """
73     function to calculate distance M
74     """
75     theta_x = np.arcsin( R_out * np.sin( np.pi - alpha_trailing ) /
76                          R_leading )
77     theta_m = np.pi - (np.pi - alpha_trailing) - theta_x
78     M = R_leading * np.sin(theta_m) / np.sin(np.pi - alpha_trailing)

```

```

78     return M
79 ##
80 def d_alpha_trailing_calc(r_leading , r_trailing , M):
81     """
82     function to calculate delta_alpha_trailing
83     """
84     return np.arcsin( ( r_leading-r_trailing )/M)
85 ##
86 def theta_T1_calc(R_out , r_trailing , alpha_trailing , d_alpha_trailing):
87     """
88     function to calculate theta_T1
89     """
90     d_x = r_trailing / (np.cos(np.pi/2.-alpha_trailing-d_alpha_trailing))
91     theta_x = np.pi - alpha_trailing - d_alpha_trailing
92     theta_temp = np.arcsin( (R_out - d_x ) * np.sin(theta_x) / R_out )
93     return np.pi - theta_x - theta_temp
94 ##
95 def theta_T2_calc(R_out , r_trailing , alpha_trailing , d_alpha_trailing):
96     """
97     function to calculate theta_T2
98     """
99     d_x = r_trailing / np.sin(alpha_trailing-d_alpha_trailing)
100    theta_temp = np.pi - np.arcsin( (R_out + d_x) * np.sin(alpha_trailing-d_alpha_trailing) / R_out )
101    return np.pi - (alpha_trailing-d_alpha_trailing) - theta_temp
102 ##
103 def theta_d_I_calc(R_out , R_in , alpha_trailing):
104     """
105     function to calculate theta_I1
106     """
107     theta_x = np.pi - alpha_trailing
108     theta_y = np.arcsin(R_out/R_in * np.sin(theta_x) )
109     return np.pi - theta_x - theta_y
110 ##
111 def perimeter_calc(T1,L1,T2,L3,r_leading):
112     """
113     function to calculate positions
114     """
115     P1 = ((T1.x - L1.x)**2. + (T1.y - L1.y)**2.)**0.5
116     P45 = ((T2.x - L3.x)**2. + (T2.y - L3.y)**2.)**0.5
117     P4 = P45-P1
118     alpha = P1 / P45
119     theta_L2 = P4 / r_leading
120 #     print P1, P45, P4, theta_L2
121     return alpha , theta_L2
122 ##
123 def distance_calc(T1,L1,L3,L4,T2,theta_L2 , d_alpha_trailing , r_leading ,
124     theta_U3 , theta_B1 , theta_B6 , theta_D3):
125     """
126     function to calculate distances along edges of domain
127     """
128     P1 = ((T1.x - L1.x)**2. + (T1.y - L1.y)**2.)**0.5
129     P2 = r_leading * theta_L2

```

```

129     P3 = r_leading * (np.pi + 2*d_alpha_trailing - theta_L2)
130     P4 = ((L3.x - L4.x)**2. + (L3.y - L4.y)**2.)**0.5
131     P5 = ((L4.x - T2.x)**2. + (L4.y - T2.y)**2.)**0.5
132     P6 = R_out * (-theta_U3-theta_T1)
133     P7 = R_out * (theta_D3-theta_T2)
134     #print theta_U3, theta_D3, theta_T1, theta_T2
135     return P1,P2,P3,P4,P5,P6,P7
136
137 #####
138 ### Generation of Nodes ###
139 #####
140 # Calculate derived variables
141 theta_blade = np.pi / N_blade
142 M = M_calc(R_out, R_leading, alpha_trailing)
143 d_alpha_trailing = d_alpha_trailing_calc(r_leading, r_trailing, M)
144 theta_T1 = theta_T1_calc(R_out, r_trailing, alpha_trailing, d_alpha_trailing)
145 theta_B1 = theta_T1_calc(R_out, r_trailing+bl, alpha_trailing,
146                          d_alpha_trailing)
146 theta_T2 = theta_T2_calc(R_out, r_trailing, alpha_trailing, d_alpha_trailing)
147 theta_B6 = theta_T2_calc(R_out, r_trailing+bl, alpha_trailing,
148                          d_alpha_trailing)
148 theta_D3 = (theta_blade - (theta_B1+theta_B6)) / 2. + theta_B6
149 ### Setting up Basic Information ###
150 theta_U3 = -(theta_blade - (theta_B1+theta_B6)) / 2. - theta_B1
151 theta_I1 = theta_U3 - theta_d_I_calc(R_out, R_in, alpha_trailing)
152 theta_I4 = theta_D3 - theta_d_I_calc(R_out, R_in, alpha_trailing)
153
154 print "alpha_trailing", alpha_trailing, "d_alpha_trailing",
155       d_alpha_trailing, "M", M
156
157 # Define some Nodes (directly calculated)
158 Origin = Node(0.0,0.0, label = "Origin")
159 T0 = Node(0.0, R_out, label = "T0")
160 L0 = Node(-M * np.sin(alpha_trailing), R_out + M * np.cos(alpha_trailing),
161          label = "L0")
162 L1 = Node(L0.x- r_leading* np.cos(alpha_trailing+d_alpha_trailing), L0.y-
163          r_leading* np.sin(alpha_trailing+d_alpha_trailing), label="L1")
164 L3 = Node(L0.x+ r_leading* np.cos(alpha_trailing-d_alpha_trailing), L0.y+
165          r_leading* np.sin(alpha_trailing-d_alpha_trailing), label="L3")
166 B2 = Node(L0.x- (r_leading+bl)* np.cos(alpha_trailing+d_alpha_trailing), L0
167          .y- (r_leading+bl)* np.sin(alpha_trailing+d_alpha_trailing), label="B2"
168          )
169 B4 = Node(L0.x+ (r_leading+bl)* np.cos(alpha_trailing-d_alpha_trailing), L0
170          .y+ (r_leading+bl)* np.sin(alpha_trailing-d_alpha_trailing), label="B4"
171          )
172 T1 = Node(-R_out* np.sin(theta_T1), R_out* np.cos(theta_T1), label = "T1")
173 B1 = Node(-R_out* np.sin(theta_B1), R_out* np.cos(theta_B1), label = "B1")
174 T2 = Node(R_out* np.sin(theta_T2), R_out* np.cos(theta_T2), label = "T2")
175 B6 = Node(R_out* np.sin(theta_B6), R_out* np.cos(theta_B6), label = "B6")
176 U3 = Node(R_out* np.sin(theta_U3), R_out* np.cos(theta_U3), label = "U3")
177 D3 = Node(R_out* np.sin(theta_D3), R_out* np.cos(theta_D3), label = "D3")
178 I1 = Node(R_in* np.sin(theta_I1), R_in* np.cos(theta_I1), label = "I1")
179 I4 = Node(R_in* np.sin(theta_I4), R_in* np.cos(theta_I4), label = "I4")

```

```

172
173 # Do some more calcs based on already defined points
174 L4_f, theta_L2 = perimeter_calc(T1,L1,T2,L3,r.leading)
175 L4 = Node( (1.-L4_f)*T2.x+L4_f*L3.x, (1.-L4_f)*T2.y+L4_f*L3.y, label = "L4"
)
176 B5 = Node( (1.-L4_f)*B6.x+L4_f*B4.x, (1.-L4_f)*B6.y+L4_f*B4.y, label = "B5"
)
177 L2 = Node( L0.x - r.leading * np.sin(np.pi/2 - alpha_trailing -
d_alpha_trailing + theta_L2), L0.y - r.leading * np.cos(np.pi/2 -
alpha_trailing - d_alpha_trailing + theta_L2), label = "L2")
178 B3 = Node( L0.x - (r.leading+bl) * np.sin(np.pi/2 - alpha_trailing -
d_alpha_trailing + theta_L2), L0.y - (r.leading+bl) * np.cos(np.pi/2 -
alpha_trailing - d_alpha_trailing + theta_L2), label = "B3")
179
180 # Define some Nodes (weighted average)
181 U1 = Node( (1.-U1D1_f)*U3.x+U1D1_f*I1.x, (1.-U1D1_f)*U3.y+U1D1_f*I1.y,
label = "U1")
182 D1 = Node( (1.-U1D1_f)*D3.x+U1D1_f*I4.x, (1.-U1D1_f)*D3.y+U1D1_f*I4.y,
label = "D1")
183 U2 = Node( (1.-U2D2_f)*U3.x+U2D2_f*I1.x, (1.-U2D2_f)*U3.y+U2D2_f*I1.y,
label = "U2")
184 D2 = Node( (1.-U2D2_f)*D3.x+U2D2_f*I4.x, (1.-U2D2_f)*D3.y+U2D2_f*I4.y,
label = "D2")
185
186 # Define Bezier Control Points
187 # points are generated, so that Bezier curves form a junction with 4 * 90
degree
188 ra = 2. * bl
189 rb = 2. * bl
190 tua = alpha_trailing + theta_U3
191 tda = alpha_trailing - theta_D3
192 tub = np.pi/2. - alpha_trailing - d_alpha_trailing
193 tdb = np.pi/2. - alpha_trailing + d_alpha_trailing
194 u2b2a = Node(U2.x + ra*np.cos(tua), U2.y + ra*np.sin(tua), label = "u2b2a")
195 u2b2b = Node(B2.x - rb*np.sin(tub), B2.y - rb*np.cos(tub), label = "u2b2b")
196 u1b3a = Node(U1.x + ra*np.cos(tua), U1.y + ra*np.sin(tua), label = "u1b3a")
197 b4d1a = Node(D1.x - ra*np.cos(tda), D1.y - ra*np.sin(tda), label = "b4d1a")
198 b5d2a = Node(D2.x - ra*np.cos(tda), D2.y - ra*np.sin(tda), label = "b5d2a")
199 b5d2b = Node(B5.x + rb*np.sin(tdb), B5.y + rb*np.sin(tdb), label = "b5d2b")
200
201 # points are generated, so that Bezier curves form a junction with 2 * 90
degree and 3 * 60 degree
202 rr = 3. * bl
203 tubb = np.pi/2. - alpha_trailing - d_alpha_trailing + theta_L2
204 tdbb = np.pi/2. - alpha_trailing + d_alpha_trailing
205 u1b3b = Node(B3.x - rr*np.sin(tubb - np.pi/6.), B3.y - rr*np.cos(tubb - np.
pi/6.), label = "u1b3b")
206 b3i2 = Node(B3.x - rr*np.sin(tubb + np.pi/6.), B3.y - rr*np.cos(tubb + np.
pi/6.), label = "b3i2")
207 b4d1b = Node(B4.x + rr*np.sin(tdbb + np.pi/6.), B4.y + rr*np.cos(tdbb + np.
pi/6.), label = "b4d1b")
208 b4i3 = Node(B4.x + rr*np.sin(tdbb - np.pi/6.), B4.y + rr*np.cos(tdbb - np.
pi/6.), label = "b4i3")

```



```

209 I2 = Node( R_in*np.sin( (1.-I2_f)*theta_I1 + I2_f*theta_I4 ), R_in*np.cos(
      (1.-I2_f)*theta_I1 + I2_f*theta_I4 ), label = "I2")
210 I3 = Node( R_in*np.sin( (1.-I3_f)*theta_I1 + I3_f*theta_I4 ), R_in*np.cos(
      (1.-I3_f)*theta_I1 + I3_f*theta_I4 ), label = "I3")
211
212 #####
213 ### Calculated recommended grid number
214 #####
215 print "#####"
216 print "The following numbers of cell settings are recommended"
217 print "Adjust absolute number by changing N_mult"
218 print "Current setting:      N_mult = ",N_mult, " (number of cells along
      blade surface) \n"
219
220 P1,P2,P3,P4,P5,P6,P7 = distance_calc(T1,L1,L3,L4,T2,theta_L2 ,
      d.alpha_trailing ,r_leading ,theta_U3 ,theta_B1 ,theta_B6 ,theta_D3)
221 P_blade = P1+P2+P3+P4+P5
222 P_cell = P_blade / N_mult
223 print "Recommendations:"
224 print "Nu = ", round(P6/P_cell)
225 print "Nd = ", round(P7/P_cell)
226 #print "Nt = ", round((R_in - (R_leading+r_leading)) / P_cell)
227 print "Nt = ", round((R_in - (B4.x**2+B4.y**2)**0.5 ) / P_cell)
228 print "Nbl = ???, user selected"
229 print "Nbl = ", round(P1 / P_blade*N_mult)
230 print "Nb2 = ", round(P2 / P_blade*N_mult)
231 print "Nb3 = ", round(P3 / P_blade*N_mult)
232 #print "Nb4 = Nb2 = ", round(P2 / P_blade)*N_mult
233 #print "Nb5 = Nb1 = ", round(P1 / P_blade)*N_mult
234 print "Nz = ???, user selected"
235
236 #####
237 ### Generation of Lines      ###
238 #####
239 # Lines for 2-D grid
240 U3U2 = Line(U3, U2); U2U1 = Line(U2, U1); U1I1 = Line(U1,I1)
241 D3D2 = Line(D3, D2); D2D1 = Line(D2, D1); D1I4 = Line(D1,I4)
242 U3B1 = Arc(U3,B1, Origin); B1T1 = Arc(B1,T1, Origin); T2B6 = Arc(T2,B6, Origin
      ); B6D3 = Arc(B6,D3, Origin)
243 I1I2 = Arc(I1, I2, Origin); I2I3 = Arc(I2, I3, Origin); I3I4 = Arc(I3, I4, Origin
      )
244 T1L1 = Line(T1,L1); L1L2 = Arc(L1,L2,L0); L2L3 = Arc(L2,L3,L0); L4L3 = Line
      (L4,L3); T2L4 = Line(T2,L4)
245 B1B2 = Line(B1,B2); B2B3 = Arc(B2,B3,L0); B3B4 = Arc(B3,B4,L0); B5B4 = Line
      (B5,B4); B6B5 = Line(B6,B5)
246 B2L1 = Line(B2,L1); B3L2 = Line(B3,L2); L2B3 = Line(L2,B3); L3B4 = Line(L3,
      B4); L4B5=Line(L4,B5)
247 U2B2 = Bezier([U2, u2b2a, u2b2b, B2], "U2B2", 0.0, 1.0, 1)
248 U1B3 = Bezier([U1, u1b3a, u1b3b, B3], "U1B3", 0.0, 1.0, 1)
249 B3I2 = Bezier([B3, b3i2, I2], "B3I2", 0.0, 1.0, 1)
250 B4I3 = Bezier([B4, b4i3, I3], "B4I3", 0.0, 1.0, 1)
251 B4D1 = Bezier([B4, b4d1b, b4d1a, D1], "B4D1", 0.0, 1.0, 1)
252 B5D2 = Bezier([B5, b5d2b, b5d2a, D2], "B5D2", 0.0, 1.0, 1)

```

```

253
254 # Nodes for Extrusion (defined at corner 00). Only required in 3D
255 if gdata.dimensions == 3:
256     U3h = Node(U3.x,U3.y,Height , label = "U3h")
257     U2h = Node(U2.x,U2.y,Height , label = "U2h")
258     U1h = Node(U1.x,U1.y,Height , label = "U1h")
259     B1h = Node(B1.x,B1.y,Height , label = "B1h")
260     B2h = Node(B2.x,B2.y,Height , label = "B2h")
261     B3h = Node(B3.x,B3.y,Height , label = "B3h")
262     L2h = Node(L2.x,L2.y,Height , label = "L2h")
263     B6h = Node(B6.x,B6.y,Height , label = "B6h")
264     B5h = Node(B5.x,B5.y,Height , label = "B5h")
265     B4h = Node(B4.x,B4.y,Height , label = "B4h")
266     T2h = Node(T2.x,T2.y,Height , label = "T2h")
267     L4h = Node(L4.x,L4.y,Height , label = "L4h")
268
269 #####
270 ### Set Cluster Functions      ###
271 #####
272 CF_b10 = RobertsClusterFunction(0,1,1.05)
273 CF_b11 = RobertsClusterFunction(1,0,1.05)
274 CF_h = RobertsClusterFunction(1,1,1.05)
275
276 #####
277 ### Definitions of Blocks      ###
278 #####
279 if gdata.dimensions == 2:
280     BL0 = Block2D(make_patch(B2L1, T1L1, B1T1, B1B2), nni=Nbl, nnj=Nb1,
281                  cf_list = [CF_b10, None, CF_b10, None],
282                  fill_condition=initial, label="BL0")
283     BL1 = Block2D(make_patch(B3L2, L1L2, B2L1, B2B3), nni=Nbl, nnj=Nb2,
284                  cf_list = [CF_b10, None, CF_b10, None],
285                  fill_condition=initial, label="BL1")
286     BL2 = Block2D(make_patch(B3B4, L3B4, L2L3, L2B3), nni=Nb3, nnj=Nbl,
287                  cf_list = [None, CF_b11, None, CF_b11],
288                  fill_condition=initial, label="BL2")
289     BL3 = Block2D(make_patch(L3B4, B5B4, L4B5, L4L3), nni=Nbl, nnj=Nb2,
290                  cf_list = [CF_b11, None, CF_b11, None],
291                  fill_condition=initial, label="BL3")
292     BL4 = Block2D(make_patch(L4B5, B6B5, T2B6, T2L4), nni=Nbl, nnj=Nb1,
293                  cf_list = [CF_b11, None, CF_b11, None],
294                  fill_condition=initial, label="BL4")
295     BU0 = Block2D(make_patch(U2B2, B1B2, U3B1, U3U2), nni=Nu, nnj=Nb1,
296                  fill_condition=initial, label="BU0")
297     BU1 = Block2D(make_patch(U1B3, B2B3, U2B2, U2U1), nni=Nu, nnj=Nb2,
298                  fill_condition=initial, label="BU1")
299     BU2 = Block2D(make_patch(I1I2, B3I2, U1B3, U1I1), nni=Nu, nnj=Nt,
300                  fill_condition=initial, label="BU2")
301     BD0 = Block2D(make_patch(B5D2, D3D2, B6D3, B6B5), nni=Nd, nnj=Nb1,
302                  fill_condition=initial, label="BD0")
303     BD1 = Block2D(make_patch(B4D1, D2D1, B5D2, B5B4), nni=Nd, nnj=Nb2,
304                  fill_condition=initial, label="BD1")
305     BD2 = Block2D(make_patch(I3I4, D1I4, B4D1, B4I3), nni=Nd, nnj=Nt,

```

```

306         fill_condition=initial , label="BD2")
307     BT0 = Block2D(make_patch(I2I3 , B4I3 , B3B4 , B3I2) , nni=Nb3 , nnj=Nt ,
308         fill_condition=initial , label="BT0")
309 elif gdata.dimensions == 3:
310     BL0 = Block3D(WireFrameVolume(make_patch(B2L1 , T1L1 , B1T1 , B1B2) , Line(
311         B1,B1h)) , nni=Nbl , nnj=Nbl , nnk=Nz ,
312         cf_list = [CF_bl0 , None , CF_bl0 , None , CF_bl0 , None , CF_bl0 ,
313         None , CF_h , CF_h , CF_h , CF_h] ,
314         fill_condition=initial , label="BL0")
315     BL1 = Block3D(WireFrameVolume(make_patch(B3L2 , L1L2 , B2L1 , B2B3) , Line(
316         B2,B2h)) , nni=Nbl , nnj=Nb2 , nnk=Nz ,
317         cf_list = [CF_bl0 , None , CF_bl0 , None , CF_bl0 , None , CF_bl0 ,
318         None , CF_h , CF_h , CF_h , CF_h] ,
319         fill_condition=initial , label="BL1")
320     BL2 = Block3D(WireFrameVolume(make_patch(B3B4 , L3B4 , L2L3 , L2B3) , Line(
321         L2,L2h)) , nni=Nb3 , nnj=Nbl , nnk=Nz ,
322         cf_list = [None , CF_bl1 , None , CF_bl1 , None , CF_bl1 , None ,
323         CF_bl1 , CF_h , CF_h , CF_h , CF_h] ,
324         fill_condition=initial , label="BL2")
325     BL3 = Block3D(WireFrameVolume(make_patch(L3B4 , B5B4 , L4B5 , L4L3) , Line(
326         L4,L4h)) , nni=Nbl , nnj=Nb2 , nnk=Nz ,
327         cf_list = [CF_bl1 , None , CF_bl1 , None , CF_bl1 , None , CF_bl1 ,
328         None , CF_h , CF_h , CF_h , CF_h] ,
329         fill_condition=initial , label="BL3")
330     BL4 = Block3D(WireFrameVolume(make_patch(L4B5 , B6B5 , T2B6 , T2L4) , Line(
331         T2,T2h)) , nni=Nbl , nnj=Nbl , nnk=Nz ,
332         cf_list = [CF_bl1 , None , CF_bl1 , None , CF_bl1 , None , CF_bl1 ,
333         None , CF_h , CF_h , CF_h , CF_h] ,
334         fill_condition=initial , label="BL4")
335     BU0 = Block3D(WireFrameVolume(make_patch(U2B2 , B1B2 , U3B1 , U3U2) , Line(
336         U3,U3h)) , nni=Nu , nnj=Nbl , nnk=Nz ,
337         cf_list = [ None , None , None , None , None , None , None , None ,
338         CF_h , CF_h , CF_h , CF_h] ,
339         fill_condition=initial , label="BU0")
340     BU1 = Block3D(WireFrameVolume(make_patch(U1B3 , B2B3 , U2B2 , U2U1) , Line(
341         U2,U2h)) , nni=Nu , nnj=Nb2 , nnk=Nz ,
342         cf_list = [ None , None , None , None , None , None , None , None ,
343         CF_h , CF_h , CF_h , CF_h] ,
344         fill_condition=initial , label="BU1")
345     BU2 = Block3D(WireFrameVolume(make_patch(I1I2 , B3I2 , U1B3 , U1I1) , Line(
346         U1,U1h)) , nni=Nu , nnj=Nt , nnk=Nz ,
347         cf_list = [ None , None , None , None , None , None , None , None ,
348         CF_h , CF_h , CF_h , CF_h] ,
349         fill_condition=initial , label="BU2")
350     BD0 = Block3D(WireFrameVolume(make_patch(B5D2 , D3D2 , B6D3 , B6B5) , Line(
351         B6,B6h)) , nni=Nd , nnj=Nbl , nnk=Nz ,
352         cf_list = [ None , None , None , None , None , None , None , None ,
353         CF_h , CF_h , CF_h , CF_h] ,
354         fill_condition=initial , label="BD0")
355     BD1 = Block3D(WireFrameVolume(make_patch(B4D1 , D2D1 , B5D2 , B5B4) , Line(
356         B5,B5h)) , nni=Nd , nnj=Nb2 , nnk=Nz ,
357         cf_list = [ None , None , None , None , None , None , None , None ,
358         CF_h , CF_h , CF_h , CF_h] ,

```

```

339         fill_condition=initial , label="BD1")
340     BD2 = Block3D(WireFrameVolume(make_patch(I3I4 , D1I4 , B4D1 , B4I3) , Line(
        B4,B4h)), nni=Nd, nnj=Nt, nnk=Nz,
341         cf_list = [ None, None, None, None, None, None, None, None,
        CF_h, CF_h, CF_h, CF_h],
342         fill_condition=initial , label="BD2")
343     BT0 = Block3D(WireFrameVolume(make_patch(I2I3 , B4I3 , B3B4 , B3I2) , Line(
        B3,B3h)), nni=Nb3, nnj=Nt, nnk=Nz,
344         cf_list = [ None, None, None, None, None, None, None, None,
        CF_h, CF_h, CF_h, CF_h],
345         fill_condition=initial , label="BT0")
346
347 # link blocks
348 identify_block_connections()
349 # define B/C
350 BU0.bc_list [SOUTH] = ExtrapolateOutBC(label='OF_outlet_00')
351 BL0.bc_list [SOUTH] = ExtrapolateOutBC(label='OF_outlet_00')
352 BL4.bc_list [SOUTH] = ExtrapolateOutBC(label='OF_outlet_00')
353 BD0.bc_list [SOUTH] = ExtrapolateOutBC(label='OF_outlet_00')
354 BU2.bc_list [NORTH] = ExtrapolateOutBC(label='OF_inlet_00')
355 BT0.bc_list [NORTH] = ExtrapolateOutBC(label='OF_inlet_00')
356 BD2.bc_list [NORTH] = ExtrapolateOutBC(label='OF_inlet_00')
357
358 BU0.bc_list [WEST] = ExtrapolateOutBC(label='OF_inlet_01') # OF_inlet_01 and
        _02 are used to group periodic boundaries
359 BU1.bc_list [WEST] = ExtrapolateOutBC(label='OF_inlet_01')
360 BU2.bc_list [WEST] = ExtrapolateOutBC(label='OF_inlet_01')
361 BD0.bc_list [EAST] = ExtrapolateOutBC(label='OF_inlet_02')
362 BD1.bc_list [EAST] = ExtrapolateOutBC(label='OF_inlet_02')
363 BD2.bc_list [EAST] = ExtrapolateOutBC(label='OF_inlet_02')
364
365 BL0.bc_list [EAST] = ExtrapolateOutBC(label='OF_wall_00')
366 BL1.bc_list [EAST] = ExtrapolateOutBC(label='OF_wall_00')
367 BL2.bc_list [SOUTH] = ExtrapolateOutBC(label='OF_wall_00')
368 BL3.bc_list [WEST] = ExtrapolateOutBC(label='OF_wall_00')
369 BL4.bc_list [WEST] = ExtrapolateOutBC(label='OF_wall_00')
370
371 if gdata.dimensions == 3:
372     BU0.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
373     BU0.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
374     BU1.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
375     BU1.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
376     BU2.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
377     BU2.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
378     BD0.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
379     BD0.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
380     BD1.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
381     BD1.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
382     BD2.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
383     BD2.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
384     BL0.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')
385     BL0.bc_list [BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
386     BL1.bc_list [TOP] = ExtrapolateOutBC(label='OF_wall_01')

```

```

387 BL1.bc_list[BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
388 BL2.bc_list[TOP] = ExtrapolateOutBC(label='OF_wall_01')
389 BL2.bc_list[BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
390 BT0.bc_list[TOP] = ExtrapolateOutBC(label='OF_wall_01')
391 BT0.bc_list[BOTTOM] = ExtrapolateOutBC(label='OF_wall_02')
392
393 sketch.prefer_bc_labels_on_faces() # required to allow grouping of
    boundaries by e3prepToFoam.py
394
395 if gdata.dimensions == 2:
396     # This is to make a nice *.svg file of the 2-D projection of the mesh
397     sketch.xaxis(-0.05, 0.05, 0.02, 0.0)
398     sketch.yaxis(0.0, 0.1, 0.02, 0.0)
399     sketch.window(-0.05, 0., 0.05, 0.1, 0.05, 0.05, 2.05, 2.05)

```

7.2 Rotor_Passage.py

```

1 ### \Rotor_Passage.py
2 #
3 """
4 Script to create a structured mesh for an Radial inflow turbine passage.
5 The geometry and grid is defined in Rotor_Profile.py
6
7 Author: Ingo Jahn
8 Last modified: 23/03/2015
9 """
10
11 import numpy as np
12 from Rotor_Profile import *
13
14 #####
15 ### Setting up Basic Information ###
16 #####
17 # For grid development, set gdata.dimensions = 2, this will create the 2-D
    projection of the mesh.
18 gdata.dimensions = 3
19 gdata.axisymmetric_flag = 0
20
21 # Set some fluid properties to allow e3prep to solve
22 # These only need to be correct if using Eilmer as solver.
23 select_gas_model(model='ideal gas', species=['air'])
24 initial = FlowCondition(p=5955.0, u=0.0, v=0.0, T=304.0)
25 inflow = FlowCondition(p=95.84e3, u=1000.0, v=0.0, T=1103.0)
26
27 #####
28 ### Set Rotor Properties ###
29 #####
30 # set filename used to store rotor data. Data can be previewed using
    Rotor_Profile.py --job=name
31 RotorFileName = "Rotor_example.py"

```

```

32
33 # execute file containing rotor data to define geometry
34 execfile(RotorFileName, globals())
35 # create anonymous functions that can be used by PyFunctionVolume()
36 pyfunction_blk0 = lambda r,s,t: PROFILE.eval(r,s,t,0)
37 pyfunction_blk1 = lambda r,s,t: PROFILE.eval(r,s,t,1)
38 pyfunction_blk2 = lambda r,s,t: PROFILE.eval(r,s,t,2)
39 pyfunction_blk3 = lambda r,s,t: PROFILE.eval(r,s,t,3)
40
41 #####
42 ### Set Number of Cells ###
43 #####
44 #nl = 60 # cells along passage
45 #nt = 60 # cells of main passage in circumferential direction
46 #nz = 60 # cells of main passage axial (at inlet) or radial (at outlet)
    direction
47 #nbl = 10 # cells in boundary layer region.
48
49 nl = 30 # cells along passage
50 nt = 15 # cells of main passage in circumferential direction
51 nz = 15 # cells of main passage axial (at inlet) or radial (at outlet)
    direction
52 nbl = 5 # cells in boundary layer region.
53
54 #####
55 ### Set Cluster Functions ###
56 #####
57 CF_b10 = RobertsClusterFunction(1,0,1.05)
58 #CF_b11 = RobertsClusterFunction(1,0,1.05)
59 #CF_h = RobertsClusterFunction(1,1,1.05)
60
61 #####
62 ### Definitions of Blocks ###
63 #####
64 BL0 = Block3D(PyFunctionVolume(pyfunction_blk0), nni=nt, nnj=nz, nnk=nl,
65             cf_list = [None, None, None, None, None, None, None, None,
                None, None, None, None],
        fill_condition=initial, label="blk0")
66
67 BL1 = Block3D(PyFunctionVolume(pyfunction_blk1), nni=nz, nnj=nbl, nnk=nl,
68             cf_list = [None, CF_b10, None, CF_b10, None, CF_b10, None,
                CF_b10, None, None, None, None],
        fill_condition=initial, label="blk1")
69
70 BL2 = Block3D(PyFunctionVolume(pyfunction_blk2), nni=nt, nnj=nbl, nnk=nl,
71             cf_list = [None, CF_b10, None, CF_b10, None, CF_b10, None,
                CF_b10, None, None, None, None],
        fill_condition=initial, label="blk2")
72
73 BL3 = Block3D(PyFunctionVolume(pyfunction_blk3), nni=nz, nnj=nbl, nnk=nl,
74             cf_list = [None, CF_b10, None, CF_b10, None, CF_b10, None,
                CF_b10, None, None, None, None],
        fill_condition=initial, label="blk3")
75
76 # link blocks
77 identify_block_connections()
78

```

```

79 #####
80 ### define B/C ###
81 #####
82 ""
83 OF_inlet_00 --> Inlet
84 OF_outlet_00 --> Outlet
85
86 OF_wall_00 --> shroud
87 OF_wall_01 --> left blade
88 OF_wall_02 --> right balde
89 OF_wall_03 --> hub
90 ""
91 # Inlet & Outlet
92 BL0.bc_list [TOP] = ExtrapolateOutBC (label='OF_inlet_00')
93 BL1.bc_list [TOP] = ExtrapolateOutBC (label='OF_inlet_00')
94 BL2.bc_list [TOP] = ExtrapolateOutBC (label='OF_inlet_00')
95 BL3.bc_list [TOP] = ExtrapolateOutBC (label='OF_inlet_00')
96 BL0.bc_list [BOTTOM] = ExtrapolateOutBC (label='OF_outlet_00')
97 BL1.bc_list [BOTTOM] = ExtrapolateOutBC (label='OF_outlet_00')
98 BL2.bc_list [BOTTOM] = ExtrapolateOutBC (label='OF_outlet_00')
99 BL3.bc_list [BOTTOM] = ExtrapolateOutBC (label='OF_outlet_00')
100 # Shroud
101 BL1.bc_list [WEST] = ExtrapolateOutBC (label='OF_wall_00')
102 BL0.bc_list [NORTH] = ExtrapolateOutBC (label='OF_wall_00')
103 BL3.bc_list [EAST] = ExtrapolateOutBC (label='OF_wall_00')
104 # Left Blade
105 BL1.bc_list [SOUTH] = ExtrapolateOutBC (label='OF_wall_01')
106 # Hub
107 BL2.bc_list [SOUTH] = ExtrapolateOutBC (label='OF_wall_02')
108 # Right Blade
109 BL3.bc_list [SOUTH] = ExtrapolateOutBC (label='OF_wall_03')
110
111 # make sure labels are plotted in svg
112 sketch.prefer_bc_labels_on_faces ()

```

7.3 Rotor_example.py

```

1 # Rotor_example.py
2 # Ingo Jahn 11/05/2015
3 # Example Job file for creating Mesh
4
5 Name = "Rotor meshing example"
6
7 print "Running ", Name, " to generate rotor mesh"
8
9 #####
10 # Setting Streamline at Passage Centre
11 #####
12 # Streamline defining passge (take values from TOPGEN)

```

```

13 R_in = 28.443e-3      # (m) radius at inlet                TOPGEN —>
    radius_inlet
14 R_out = 12.5e-3      # (m) radius at outlet                TOPGEN
    —> radius_outlet (mean)
15 Z_out = 12e-3       # (m) height at outlet (only used for ellipse) —> design
    variable
16 Twist = 60./180. * np.pi # (rad) twist angle of streamline —> design
    variable
17 theta_in = -21./180. * np.pi # (rad)                    TOPGEN —>
    beta_inlet
18 theta_out = 60./180. * np.pi # (rad)                    TOPGEN —>
    beta_outlet
19 L_in2 = 0.3         # position of 2nd control point (fraction along streamline
    from inlet) —> design variable
20 L_in3 = 0.6         # position of 3rd control point (fraction along streamline
    from inlet) —> design variable
21 L_out4 = 0.8        # position of 4th control point (fraction along streamline
    before outlet) —> design variable
22 L_out5 = 0.9        # position of 5th control point (fraction along streamline
    before outlet) —> design variable
23
24 # Define Central Streamline that is used to set blade passage shape.
25 # STREAMLINE must be a 3-D path function as described at the end
26 STREAMLINE = Topgen2Bezier(R_in, theta_in, R_out, theta_out, Z_out, Twist,
    L_in2, L_in3, L_out4, L_out5)
27
28 #####
29 # Setting Parametric curves to define passage
30 #####
31 # Area of Passage
32 A0 = 3.9e-4/12.*np.cos(21./180. * np.pi) # (m2) Area at inlet
33 A1 = 6.2e-4/12.*np.cos(70./180. * np.pi) # (m2) Area at outlet
34 A1 = 0.8*A1
35
36 # Define parametric evolution of area
37 # AREA must be a 1-D path function
38 AREA = Poly_1D((A0,A1))
39
40 # set corner radius and boundary layer refinement height
41 # RC must be a 1-D path function
42 RC = Const_1D(0.0002)
43 # BL must be a 1-D path function
44 BL = Const_1D(0.0002)
45
46 # set Blade thickness at root and tip
47 # T_ROOT and T_TIP must be a 1-D path function
48 T_ROOT = Const_1D(0.001)
49 T_TIP = Const_1D(0.001)
50
51 # set lean of rotor blades
52 # LEAN must be a 1-D path function
53 LEAN = Const_1D(0./180.*np.pi)
54

```



```

55 # set number of Blades
56 N_BLADE = 9
57
58
59 #####
60 # Setting Parametric Profile
61 #####
62 # Define parametric profile used to generate passage shape
63 # PROFILE must be a 2-D profile object, which contains sub-division into 4
   grid-able blocks
64 Surf1 = Rect(STREAMLINE, AREA, T_ROOT, T_TIP, BL, N_BLADE, LEAN)
65 Surf2 = RectCorner(STREAMLINE, AREA, T_ROOT, T_TIP, BL, RC, N_BLADE, LEAN)
66
67 # define blending function
68 # BLEND must be a 1-D path function varying between 0. and 1.
69 # BLEND = Poly_1D((0., 1.))
70 BLEND = Const_1D(1.0)
71
72 # Assemble Profile
73 PROFILE = Blended2Dsurface(Surf1, Surf2, BLEND)
74
75 #####
76 # Defining Rotor blank and Stator
77 #####
78 # set Clearance
79 CLEARANCE = Const_1D(0.001)
80
81 # Rotor back thickness
82 R_THICK = Const_1D(0.002)
83
84 #####
85 # Define Output Files
86 #####
87 # set file writing Flag
88 FILES.flag = 0
89
90 # set Filenames
91 FILES.nameroot = "Dat_"
92 FILES.Slices = 10
93 FILES.Points1 = 20
94 FILES.Points2 = 5
95
96 #####
97 # Setting Visulisation Properties
98 #####
99 # set Visulisation Flag
100 VISUAL.flag = 1
101
102 # Set Properties for Showing Profile in python window
103 VISUAL.slices = 10
104 VISUAL.channels = 2
105 VISUAL.nodes = 40
106

```

```

107 # path functions:
108 """
109 Following path functions are required:
110 1-D path:
111     Function which returns single value (betwee 0 and 1) as a function of
        parametric input parameter t.
112     Function requires two sub-functions
113     self.eval(t) —> provides local value
114     self.gradeval(t) —> provides local gradient d/dt
115
116     Options:
117     - Const_1D(Value) —> will output a constant value
118     - Poly_1D((coeff0, coeff1, coeff2, ...)) —> creates polynomial of
        from  $y = \text{coeff0} + \text{coeff1} * t + \text{coeff2} * t^{**2} + \dots$ 
119     - Bezier_1D((coeff0, coeff1, coeff2, ...)) —> creates bezier curve
        through control points defined by coeff
120
121 3-D path:
122     Function which returns three value (betwee 0 and 1) as a function of
        parametric input parameter t.
123     Function requires two sub-functions
124     self.eval(t) —> provides local value (x,y,z)
125     self.gradeval(t) —> provides local gradient dx/dt, dy/dt, dz/dt
126
127     Options:
128     - Bezier_3D(( (coeff0_x, coeff0_y, coeff0_z), (coeff1_x, coeff1_y,
        coeff1_z), (coeff2_x, coeff2_y, coeff2_z), ...))
129     —> creates bezier curve through control points defined by coeff
130     - Topgen2Bezier(R_in, theta_in, R_out, theta_out, Z_out, Twist, L_in2,
        L_in3, L_out4, L_out5)
131     —> function that takes TopGen Outputs (R_in, theta_in, R_out,
        theta_out, Z_out, Twist) and generates a corresponding point
        Bezier curve.
132     (L_in2, L_in3, L_out4, L_out5) are used to control position of
        intermediary control points.
133
134 """

```
