CHALMERS

Open Source Mesh Generation and CFD Simulations for Francis Turbine

Master's Thesis in Sustainable Energy Systems

HIMANSHU KAPOOR

Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
Göteborg, Sweden 2014
Master's Thesis 2014:71

MASTER'S THESIS 2014:71

Open Source Mesh Generation and CFD Simulations for Francis Turbine

Master's Thesis in Sustainable Energy Systems

HIMANSHU KAPOOR

Department of Applied Mechanics
Division of Fluid Dynamics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2014

Open Source Mesh Generation and CFD Simulations for Francis Turbine

Master's Thesis 2014:71
ISSN 1652-8557
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-41296 Göteborg
Sweden

Tel. +46-(0)31 772 1000

Göteborg, Sweden 2014

Open Source Mesh Generation and CFD Simulations for Francis Turbine
Master's Thesis in the Sustainable Energy Systems
HIMANSHU KAPOOR
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology

Abstract

Hydro power can be recognised as a relatively mature technology of all renewable energies. Efficiency around 90 % are commonly observed in Francis turbines. Since hydro power is relatively cheaper among other power sources, it is used as base power in energy systems. Additionally hydro-power plants are often operated at off design conditions, which may lead to cavitation. Accordingly design improvements for improving the efficiency of the turbine at off design conditions are of great interest to designers. Computational fluid dynamics is used often for this analysis for predicting fluid flows as per modified design.

In the field of Computational Fluid Dynamics, open source software have developed a significant role in past few years due to economical reasons and providing user access to the source code. In many published works it has been successfully shown that the OpenFOAM solver produces comparable results to commercial codes. Most of the published work on OpenFOAM uses a mesh generated by a commercial software, most commonly on ICEM. As a step further in this thesis work, mesh has been generated on open source software for a typical Francis turbine. The geometry used for this work has been used from experimental rig for Francis turbine at Applied Mechanics Department.

Further possibilities of parametrisation in terms of geometry and mesh distribution have been explored. For diffuser complete parametrisation of geometry and mesh is achieved, for guide vane and Runner a structured mesh of acceptable quality has been made. To validate the mesh and mesh generation procedure simulations were carried out various operating conditions to analyse the flow. Two equation RANS models, standard k-ϵ and it's variant RNG k-ϵ model have been used for turbulence modelling. The efficiency of turbine did not vary much with choice of turbulence models. Overall it can be concluded that Open Source Software for mesh generation and CFD simulations can be successfully used for R&D in academia and industry.

Keywords: Hydro power, Francis Turbine, Parametrisation, Open Source

Contents

Acknowledgements

First and foremost my biggest thanks to my supervisor Dr Håkan Nilsson, for giving me a chance to do this thesis. I cannot thank him enough for allowing me to begin my thesis mid term. I started this thesis with a very little knowledge about open source software, linux and programming languages but his immense patience helped me sail through. I would also like to thank Ph.D. student Ardalan Javadi for his prompt guidance throughout the thesis work.

Lastly, I would like to thank my friend Ansuman Pradhan at the Applied Mechanics Department for sharing his very valuable experience & knowledge in this field and having discussions about this work time to time.

Himanshu Kapoor, Göteborg 22/10/14

Chapter 1

Introduction

Computational fluid dynamics (CFD) is a well established tool for analysing flow physics for engineering problems. The basic procedure for CFD can be divided into three steps : pre-processing, simulations and post-processing. Pre-processing involves steps like creating geometry, creating mesh and defining boundary conditions. Generally when the simulations are run for the very first time, there is a little knowledge about the flow and it is difficult to say weather the mesh will be able to capture the desired flow physics. Hence steps like mesh refinement in the near wall region make the regeneration of the mesh inevitable. This is a highly time consuming and a iterative procedure, see Figure 1.1. In order to reduce the time spent in case setup, it's necessary that this bottleneck is removed. One way of doing this would be to create scripts for mesh generation and then simply change the parameters as per requirement. The next challenge is recreating mesh after carrying out minor design modifications. Design modifications are required to be done in the designing phase to study the effect of changes in design on global parameters such as efficiency, in case of a turbine.

This kind of automation in case setup requires certain set of parameters on which automation is based upon. A case setup in which a given geometry and it's mesh, can be altered on basis of a certain set of parameters may be called as parametrised case. In hydro turbines, parametrisation holds a high significance because for a complete efficiency analysis of a hydro turbine, large number of simulations must be carried out at various operating conditions and at different guide vane angles in order to note the efficiency of turbine at various operating conditions in order to obtain a contour chart which is known as Hill Chart. This poses a time consuming, repetitive exercise of mesh generation at various guide vane angles. A parametrised case will allow reduction in total time consumed.

While designing a new turbine and it's other components, parametrisation can be used for : changing diffuser angle for maximum pressure recovery in order to improve efficiency, optimisation of leading and trailing edge of guide vane to minimise the drag

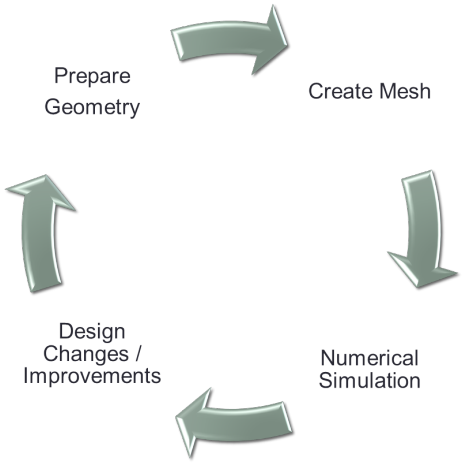


Figure 1.1: Iterative Procedure

coefficient, variation of inlet and outlet angles of runner blades based on velocity triangles. These examples are only few of the many such applications, again automation in the case setup will reduce the time and also allow for some optimisation techniques to be used.

The dominance of commercial software for mesh generation and parametrisation is well acknowledged, and known. As a step further, open source software was chosen in this work. Open source software come without any license cost and provide user a chance to look and modify the source code which is useful in case user wants to implement new functions.

1.1 Previous Work

This work has been carried out on the the Francis Turbine rig at the Department of Applied Mechanics, Chalmers university of Technology, Göteborg, Sweden. The turbine consists of a spiral casing, six guide vanes and ten runner blades followed by a draft tube. Detailed description (in Swedish language) of the experimental rig can be found in bachelor thesis report by Nilsson *et al.* [19]. In author's experience no work has been published for open source mesh generation for turbo machinery applications. Parametrisation of mesh and optimisation of design for GAMM Francis turbine runner using artificial neural networks and genetic algorithms using a commercial software is shown in a study by Derakhshan *et al.* [8]. OpenFOAM solver has been used in the ERCOFTAC centrifugal pump case study by Petit *et al.* [24] and for studying flow in Kaplan turbine by Petit [23].

Development of tools in OpenFOAM for steady state computations have been presented and tested by Page *et al.* [21]. An exhaustive overview of tools in OpenFOAM for turbo machinery and OpenFOAM solver itself are presented by Jasak *et al.* [13].

1.2 Aim and Scope

The aim of this work is to first to investigate various open source mesh software and based on this investigation, create a mesh of all the components of a Francis turbine. In addition the aim is also explore the possibilities of parametrisation. The next step is to carry out CFD simulations using OpenFOAM solver in which coupling between rotating and stationary parts is carried out using frozen rotor and mixing plane interface. After considerable number of simulations have been carried out, a qualitative analysis is done to have an insight of the flow and mesh is validated for various operating conditions.

1.3 Problem Description

One of the key issues in this work can be identified as parametrised mesh generation on open source software. The first step towards which is to create a mesh.

1.3.1 Cleaning Imported CAD Geometry

In this work, the geometry was provided in form of IGES files and STP files. The IGES format preserves the actual dimensions and units of the geometry. The problem however arises that in several parts of the geometry extra edges show up (for unknown reasons). By extra edges author means, for example a face is made up of four edges, a fifth edge of zero dimension would also appear alongside. These edges have to be removed and geometry has to be cleaned for several such parts to proceed for meshing. In author's experience, STEP files work much better for CAD data exchange than the former one. Salome provides direct import of geometry with these two formats where it's native format is brep type.

1.3.2 Create mesh

Mesh can be classified in three categories:

Unstructured Mesh : This type of mesh are probably easiest ones to make, and require lest effort from user. Since the flow in turbo machines is fairly complex and most importantly there has to be a coupling between various parts of turbine, this strategy was left unexplored in this work.

Structured Mesh : This type of mesh are comparatively difficult to make, because there are a lot of trade offs which must be considered. It requires a lot of effort from

the user side to carefully relate the mesh to the physical flow. This strategy offers clear advantage over unstructured mesh in the following aspects.

- Utilise less memory storage
- Computationally faster

Hybrid Mesh : The underlying principal of this strategy is to make a structured mesh at walls and at interface of domains (if there are multiple domains as in case of turbines) and leave the rest of domain as unstructured. Hybrid meshes are very useful when geometry is too complex to make a structured mesh, and at the same time flow is complex.
In this work the focus was kept on creating hexahedral meshes because of the fore mentioned reasons and for all the components except spiral casing, a structured mesh of reasonable quality has been produced. To assess the quality of mesh, *checkMesh* utility of OpenFOAM can be used, which gives a comprehensive information about the mesh quality. It gives details on number of cells, bounding box of the mesh etc. Some key parameters which give a quick idea about mesh quality are:

- Aspect Ratio : Ratio of longest to shortest edge.
- Non-orthogonality : Angle between face normal and line formed between two cell centres.

A much more comprehensive mesh check can be done by adding *allGeometry allTopology* after the checkMesh command. A good explanation of the results obtained by this command can be found at open wiki page for check Mesh.

1.3.3 Parametrisation of Mesh

The next step is parametrisation which can have two meanings. One would be to parametrise the mesh on basis of total number of cells, boundary layer growth factor, number of layers in near wall region. The second would be to parametrise the geometry itself, for example change in the guide vane angle, would change the volume of the flow domain. The latter requires more generalised algorithms and some experience of the flow to decide and choose a set of parameters. Parameters for both geometrical variations and mesh distribution can be simply controlled by a python script.

Geometry parametrisation for diffuser in terms of diffuser angle, length of conical part has been done. For the rest of geometrical components in the given time, geometrical parametrisation was not done as for all the variations, simulations have to be done, to validate the procedure and it was considered that this would not be achievable in given time frame.

1.4 Description of Geometry

1.4.1 Spiral Casing

Spiral Casing or Scroll casing (Figure 1.2) is used to deliver water from the penstock to the guide vanes. The cross sectional area of scroll case continuously decreases so that magnitude of velocity of water remains constant at each cross section as it enters the guide vane. One interesting feature in this case is absence of stay vanes whereas usually scroll case in power plants have stay vanes. The purpose of stay vanes is to carry pressure loads in spiral case [9].

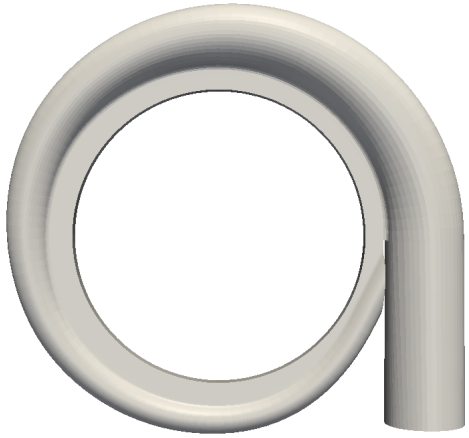


Figure 1.2: Spiral Case

1.4.2 Guide Vanes

Guide vanes are used for controlling the flow entering the runner, hence controlling the power output of Francis turbine. Their function is also to turn the flow at an appropriate angle before it enters the runner. The inner diameter of the flange is 82.4 mm, outer diameter is 140 mm, and height of vane is 8mm. The image can be seen in Figure 1.3. The area occupied by guide vanes is 32.8% of total area. This is different than what is usually observed, as in typical guide vane cascade for a Francis turbine, the guide vanes are very densely placed. Since the vanes are non symmetrical, the cross sectional area is lesser on the what is typically called pressure side of the aerofoil (hydrofoil in case of water turbine). In such case the fluid will accelerate on the pressure side of the vane, and according to Bernoulli principle, the pressure will be less in this region.

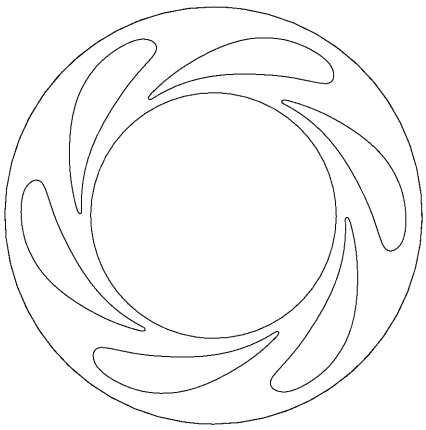


Figure 1.3: Guide Vane

1.4.3 Runner

The Runner of this experimental rig is a radial runner as the inlet flow is will have only radial component to it. The original Francis runner developed by inventor James B. Francis was of this type. Now days runner of Francis turbines are of mixed flow type in which flow enters the turbine at some angle between radial and axial [9].

The runner consists of ten blades, rotating in counter clockwise direction. The blades are not full length meaning that they do not go all the way down on the shroud up to outlet of runner. This may lead to reduced efficiency of turbine because if blades are not full in length it means that fluid leaving the runner will still have some energy which would not have been converted into useful mechanical energy at shaft. Figure 1.4 shows top view of runner geometry.

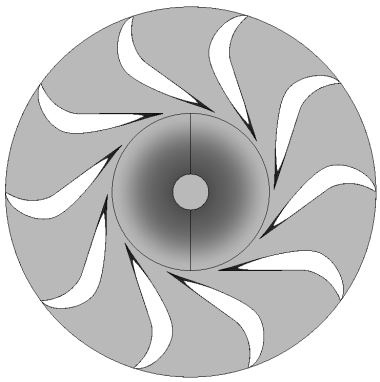


Figure 1.4: Runner

1.4.4 Draft Tube

The role of draft tube is to convert the kinetic energy at the outlet of runner to useful pressure energy. The diffuser or draft tube (Figure 1.5) is cylindrical at the inlet and conical afterwards. The diameter of cylindrical part is 0.038 metres and is 0.23 metres long. The conical region is 0.27 metres long with outlet diameter being 0.095 metres. The complete turbine assembly is shown in Figure 1.6.

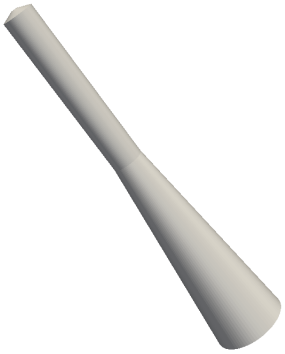


Figure 1.5: Draft Tube

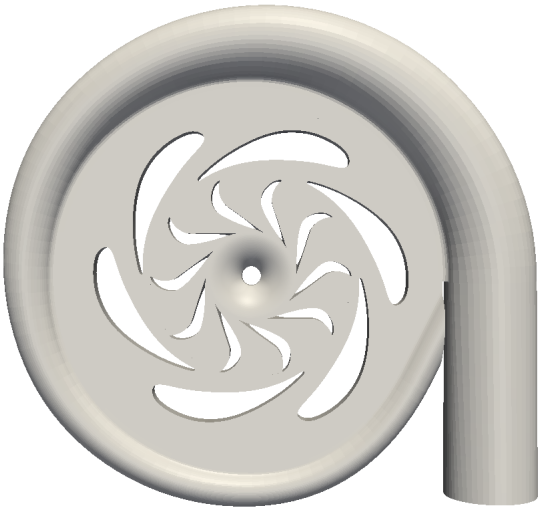


Figure 1.6: Complete Assembly (without draft tube)

Chapter 2

Methodology

2.1 Choice of Software

This section describes the motivational reasons for all the software used.

2.1.1 Mesh Generation

There are a lot of open source meshing software, each having there own advantages and disadvantages, pertaining to a given case. A good and thorough analysis on the most common software for mesh generation can be found in a technical report by Kortelainen [15]. In this report, analysis has been done on three software namely : Gmsh, SnappyHexMesh and Salome. Based on the conclusions of this report, it was decided that Salome was the most appropriate software for this work. The motivating reasons for this choice can be noted as:

- Python console for executing scripts
- Powerful geometry module which is useful for manipulating imported CAD geometries
- Mesh module with various algorithms for creating structured meshes
- Well documented user guide, and active forum of users

2.1.2 Simulations & Post Processing

OpenFoam extend 3.0 Turbo is used for carrying out flow simulations. OpenFOAM has been successfully tested and compared with commercial software for carrying out simulations of turbo machinery. Features such as generalised grid interface and mixing plane have been rigorously tested and validated results are available.For post processing, open source software ParaView is used, which is the main post processing tool supplied

with OpenFOAM. It is executed in the standard case directory by command paraFoam which in fact is a script that launches, ParaView.

In order to extract engineering quantities like efficiency, function objects can be used. SIG (Special Interest Group) Turbo machinery provides a library (turboPerformance) of functionObjects which are useful for calculating following parameters which are useful for CFD analysis on turbomachinery.

- Efficiency (%)
- Head (m)
- Forces and moments on rotating patches.

A detailed description on calculation of quantities, compiling, and using the library can be found on Sig Turbomachinery Library turboPerformance wiki page. A script for plotting the above mentioned quantities is also supplied which is useful for studying the evolution of these parameters.

2.2 Numerical Studies

This section describes in detail the various intricacies involved in setting up the simulations for a Francis turbine case.

2.2.1 Coupling of Non Conformal Mesh

By virtue of complexity of the geometrical components, a standard practice is to create a separate mesh of the rotating and stationary components of turbine. In principle, the mesh for the components can be conformal, meaning the cell size and distribution on the interface of a rotor and stator can match perfectly. This is however, very difficult to achieve due to various practical reasons pertaining to the complexity of geometry and compromises on mesh quality may have to be done. This has lead to development of a numerical procedure called Generalised Grid Interface (GGI) to transfer the flow variables between a given set of non conformal meshes. GGI is developed and implemented in OpenFOAM by Beaudoin *et. al* [2]. Beyond coupling two non conformal meshes, ggi finds it's use in form of cyclic ggi which is used as periodic boundary condition for non planar patches. A very useful tool called *ggiCheck* function object can be used to check the flux imbalance across a ggi interface. The values reported by this must be very low, which will indicate that there is very low imbalance of flux, larger values indicate vice verse. In most cases according to author's experience, large flux imbalance values indicate poor quality of mesh.

2.2.2 Coupling Rotating and Stationary Domains

The Next question is : How to transfer flow information between two domains out of which one is stationary and other is rotating. The most accurate choice would be to use transient simulations by using sliding grid technique. Since these simulations are time consuming, two common steady state techniques, Frozen Rotor and Mixing Plane have been used in this work.

Multiple Reference Frame : In multiple reference frame, runner is solved in rotating frame and stationary zones like guide vane and draft tube is solved in stationary frame. In rotating frame Coriolis force is added to governing equations and flux is calculated from relative velocity. In OpenFOAM there is a dedicated solver for carrying out these kind of simulations called MRFSimpleFoam. For setting up the case for MRFSimpleFoam the procedure can be found under validation test cases of SIG turbomachinery. A very thorough and comprehensive information about implementation of MRF, SRF (single reference frame) in OpenFOAM and other ways to set up cases using moving mesh, for turbo machinery applications is given in a presentation by Nilsson [20].

Frozen Rotor Technique : In this technique, relative position of rotor and stator is fixed. The disadvantage is that the flow is captured only at one single position and not at all the positions which is unphysical and hence this technique is inaccurate as the wakes get unphysical. This technique gives only a general behaviour of flow and only used at the initial design phase of a turbine [23].

Mixing Plane Technique : In this technique the flow is circumferentially averaged between rotor and stator components and only one rotor stator passage is required to be simulated. It was originally proposed by Denton [7]. Numerically this is achieved by creating virtual patches of rotor and stator interface. The flow variables on each side are interpolated on to this patch using GGI and are then circumferentially averaged. This method predicts the overall performance of the turbine reasonably well but the wakes at the interface are inherently eliminated. Detailed description and comparison with commercial codes CFX and Numeca can be seen in a study by Blaim *et al.* [5].

2.3 Boundary Conditions and Initial Conditions

In order to specify the radial and circumferential components of velocity as well as inlet turbulent conditions a boundary condition called profile1DfixedValue is used in which the flow values are specified in a file with *CSV* extension in constant folder. Detailed instructions for compiling and usage are explained under wiki page of Sig Turbomachinery

Library OpenFoamTurbo. Table 2.1 presents the boundary conditions at inlet of guide vane, draft tube outlet.

Variable	Inlet	Outlet
p	zeroGradient	fixedMeanValue (0)
U	profile1DfixedValue	zeroGradient
k	fixedValue	zeroGradient
ϵ	fixedValue	zeroGradient

Table 2.1: Boundary Conditions

In case of frozen rotor simulations ggi boundary condition is used at the rotor stator interface. For the other set of simulations i.e. stage interface mixing plane boundary condition is used in which at runner inlet discretisation is done along z axis and at runner outlet discretisation is done along radial direction. The boundary condition at the inlet, the face (patch) in the centre of diffuser which will be connected to the shaft of the runner must have a rotating wall boundary condition but in the simulations this was assumed as a stationary wall. The operating conditions used for this case are given in Table 2.2 and simulations are carried out at constant guide vane angle 14.4 °.

No.	Volume Flow(litres/s)	Rotational Speed(RPM)
1	4.1	2650
2	4.4	2300
3	4.6	2000

Table 2.2: Operating Conditions

2.4 Solver Settings

In Table 2.3 and Table 2.4 settings used for mixing plane and frozen rotor simulations are given respectively. Since OpenFOAM is a segregated solver, for pressure velocity coupling SIMPLE algorithm is used [25]. The main procedure of SIMPLE algorithm is described in [29]. Both *relTol* and *tolerance* can be used as measures of convergence. Value of tolerance specifies the value to which initial residual must reach and entry *reltol*

specifies ratio of current residual to final residual which solver uses as a measure of convergence. There is also a possibility of specifying maximum number of iterations by specifying *maxiter* in *fvSolution* file in system directory.

Variable	Solver	Preconditioner	Tolerance	Rel Tol
p	BiCGStab	DILU	1e-7	0.01
U	BiCGStab	DILU	1e-6	0.1
k	BiCGStab	DILU	1e-6	0.1
ϵ	BiCGStab	DILU	1e-6	0.1

Table 2.3: Solver settings for mixing plane simulations

Variable	Solver	Smoother	Tolerance	Rel Tol
p	GAMG	Gauss Seidal	1e-7	0.01
U	smoothSolver	Gauss Seidal	1e-6	0.1
k	smoothSolver	Gauss Seidal	1e-6	0.1
ϵ	smoothSolver	Gauss Seidal	1e-6	0.1

Table 2.4: Solver settings for frozen rotor simulations

GAMG stands for generalised geometric-algebraic multi-grid solver and BiCGStab solver stands for biconjugate gradient stabilised method. Using GAMG solver results in faster computation times. The main idea is to use different levels of grid spacing. In this method a quick solution is generated on a coarse mesh and then the solution is mapped onto a finer mesh [1]. More information on GAMG and BiCGStab solvers and preconditioners can be found in a study by Sickel [26] and in a report by Behrens [3], detailed description of multi-grid solver in OpenFOAM is given.

2.5 Turbulence Modelling

The flow in a turbine is strongly affected by streamline curvature of blades and the flow is of swirling type. Accordingly the choice of turbulence model depends on these two features.

The two equation models, mainly variants of standard k-ϵ model, have been quite often used in analysis of Francis turbines. In a study by Maruzewski *et al.* [17] wake of turbine blades in a Francis turbine is studied using standard k-ϵ model and SST Shear Stress model in steady state. The effect of streamline curvature on turbulent flow properties is studied by Pathak *et al.* [22] for a jet flow by streamline curvature modifications in standard k-ϵ model. The streamline curvature modifications are based on work by Cheng *et al.* [6].

In the present work standard k-ϵ model and RNG k-ϵ model was used for all the operating conditions. In both turbulence models, transport equation for k and ϵ is solved. Transport equation for ϵ is given by equation 2.1 [1]. In this equation $C_{\epsilon 1}$,$C_{\epsilon 2}$, σ_ϵ are constants and ν_t is turbulent viscosity. RNG model differs from standard k-ϵ model due to addition of a source term S_ϵ as shown in equation 2.1.

The source term is given by equation 2.2 where S_{ij} is strain rate tensor and ν is viscosity. The source term is modelled as equation 2.3 where η_0, β are constants and η is given by equation 2.4. In geometries with strong streamline curvature the effect of strain rate will be better modelled by RNG k-ϵ model because as per modified transport equation of ϵ, there will be less destruction of ϵ and hence reducing k. This will in turn reduce effective viscosity given by equation 2.5. For this reason better results are expected with RNG k-ϵ model [1].

The value of the coefficients used in both these turbulence models in OpenFOAM are listed in Table 2.5. Description of implementation of standard and RNG k-ϵ turbulence models in OpenFOAM is discussed in a study by Furbo [10].

$$\begin{aligned}\frac{\partial \epsilon}{\partial t} + \langle U_j \rangle \frac{\partial \epsilon}{\partial x_j} = C_{\epsilon 1}\nu_T \frac{\epsilon}{k}\left[\left(\frac{\partial \langle U_i \rangle}{\partial x_j} + \frac{\partial \langle U_j \rangle}{\partial x_i}\right)\frac{\partial \langle U_i \rangle}{\partial x_j}\right] \\ -C_{\epsilon 2}\frac{\epsilon^2}{k} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right) + \frac{\partial \epsilon}{\partial x_j}\right] - S_\epsilon\end{aligned} \tag{2.1}$$

$$S_\epsilon = 2\nu S_{ij}\left\langle \frac{\partial U_l}{\partial x_i}\frac{\partial U_l}{\partial x_j}\right\rangle \tag{2.2}$$

$$S_\epsilon = \frac{C_\mu \eta^3 (1 - \frac{\eta}{\eta_0}) \epsilon^2}{(1 + \beta \eta^3)\, k} \tag{2.3}$$

$$\eta = \frac{k}{\epsilon} \sqrt{2 S_{ij} S_{ij}} \tag{2.4}$$

$$\nu_T = C_\mu \frac{k^2}{\epsilon} \tag{2.5}$$

Coefficient	Value
C_μ	0.09
C_1	1.44
C_2	1.92
α_ϵ	0.76923

(a) Coefficients used in Standard k-ϵ model

Coefficient	Value
C_μ	0.0845
C_1	1.42
C_2	1.68
α_k	1.39
α_ϵ	1.39
η_0	4.38
β	0.012

(b) Coefficients used in RNG k-ϵ model

Table 2.5: Coefficients used in turbulence models in OpenFOAM

Chapter 3

Results

This chapter first introduces results from mesh generation in Salome highlighting the procedure and the weakness, possible improvements for further studies. The detailed description of mesh generation is given in Appendix. In the second part results from the simulations are reported along with a discussion.

3.1 A look at the Mesh

3.1.1 Spiral Casing

Even after multiple trials, mesh for the spiral case was not generated. The geometry comprises of seven sections from inlet to the last cross section with the least area circumeferentially. The mesh for the last section could not be made.

3.1.2 Guide Vanes

The mesh is created using pressure side of one hydrofoil and suction side of the consecutive one. Then the periodic boundaries on leading edge and trailing edge are created by drawing lines to centre and partitioning them using the inner radius and outer radius. The same procedure can be scripted so as to automatically generate the mesh at different guide vane angles, however, there is a disadvantage to this strategy, at full opening of the guide vanes, the periodicity at the trailing edges will vanish, thus inhibiting the robustness of the script. For future works this can be taken care of by probably using

some other strategy. The Figure 3.1 shows the mesh used for mixing plane simulations. There are 96,950 cells in the single domain of guide vane passage mesh. The results from checkMesh utility do not show any error in terms of aspect ratio, skewness and cell determinant values.

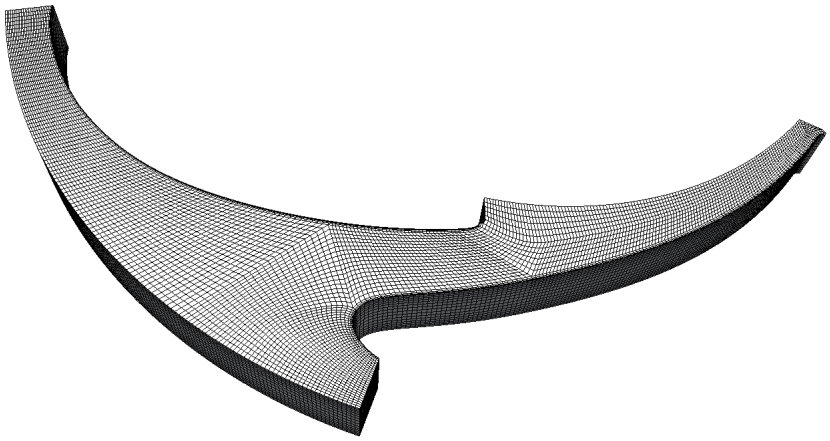


Figure 3.1: Mesh for single guide vane passage domain

3.1.3 Runner

The similar procedure as described for guide vane was used for making the mesh of runner and the complete procedure is detailed in appendix. It can be observed in Figure 3.2 that topology of the blade on the suction side is such, that at the leading and trailing edge, mesh quality will be poor. This almost kills the possibility of making a O Grid mesh so as to resolve boundary layer flow. This is because a very fine mesh in near wall region would cause mesh to be highly non orthogonal, which would induce more errors. The checkMesh utility has a threshold vale of 70 set for non orthogonality and all the faces above this limit are added into separate set of non orthogonal faces in the constant/polymesh/sets folder. For carrying out simulations with OpenFOAM in multiple reference frames, cell zones must be specified, which define the part of mesh on which rotation will be applied. This can be directly done in Salome by specifying Group of Volumes in mesh module. After this while converting the mesh using *ideasUnvToFoam* cell zone will be directly reported. The checkmesh results show that there are in total 438043 faces in the mesh, and 9670 faces amongst them are non orthogonal with maximum non orthogonality of 82.21 and average value of 27.65. These are located mainly at the leading edge and trailing edge of the suction side of the blade. Other quality checks such as skewness and

aspect ratio are alright. The non orthogonality error is expected to inhibit the quality of results, but to remove these errors and topological changes are inevitable. For future studies this topology may be improved so as to have a better mesh.

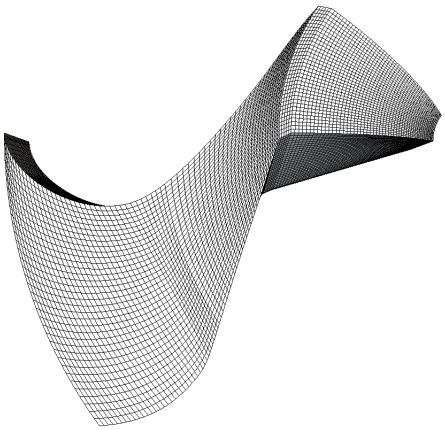


Figure 3.2: Mesh for single runner blade passage domain

3.1.4 Draft Tube

Figure 3.3 shows the mesh for diffuser. There were no errors reported in the mesh.

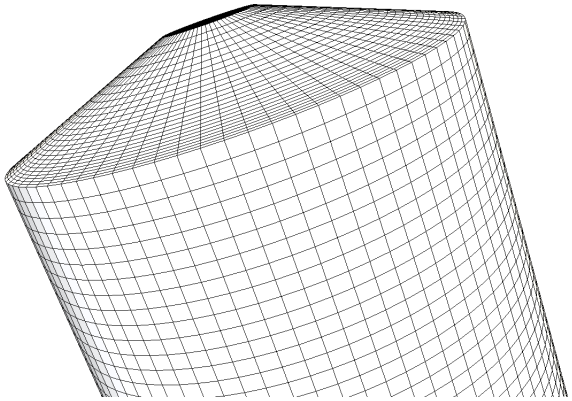


Figure 3.3: Mesh for Draft tube

3.2 Results of Simulations and Qualitative Analysis

Overview of simulations is presented in the following section. There is no availability of detailed experimental results for this case hence, the analysis is of a qualitative nature. The experimental efficiencies are compared with efficiencies obtained with mixing plane simulations using RNG k-ϵ model, in Table 3.1. It must be noted that experimental efficiencies listed in Table 3.1 are obtained by students in the course work.

	Experiment	Mixing plane
BEP	69.3%	85.33%
Part Load	67.2%	80.3%
High Load	62.02%	74.5%

Table 3.1: Efficiency Comparison

While it is difficult to pin point exact sources of error, some possible reasons which collectively contribute to discrepancies and disagreement with experiment values, could be that the mesh of spiral case was not used because of which the losses in spiral case are not accounted. In future studies, spiral case mesh must be used and first results at inlet of guide vanes must be obtained to be used as initial conditions for mixing plane simulations, in case simulation time is to be reduced by avoiding use of spiral case mesh. Also the experimental rig uses an inlet bend (penstock) for delivery of water to spiral case, the losses in it are also not accounted. It must also be taken in account that the mesh quality for runner is only reasonable due to non orthogonality errors specially at the leading edge. This would definitely reduce accuracy of flow prediction at the interface.

In all the operating conditions a counter clock wise swirl was observed. This is different than what it should be at the best efficiency point. This could be due to blade angles at outlet which turn the flow so much that it enters into reverse swirl. As shown in the geometrical Figure the blades are not full length, this would cause water to leave with unconverted kinetic energy from runner in any operating condition. Hence this design must be improved in future studies.

To evaluate the performance of diffuser in terms of pressure recovery, coefficient of pressure is plotted in the conical region with an assumption that the cylindrical region at the inlet of draft tube will not contribute much to pressure recovery. Equation 3.1 is used to calculate the value of coefficient of pressure. The points for calculation are

equidistant and this choice is completely arbitrary. In this equation p_2 is pressure at the point where coefficient of pressure is being plotted, ρ is density of water and p_1, U_1 are static pressure and mean velocity at the reference point. Ideally, pressure recovery factor in draft tube is 0.95 [18].

$$C_p = \frac{p_2 - p_1}{\frac{1}{2}\rho U_1^2} \tag{3.1}$$

There is a significant role of turbulence in inlet region of draft tube and results indicate that in steady state, turbulence and it's effect is not being completely resolved, as at the best efficiency point there must be no back flow at all, where as both back flow and counter swirl was observed. In Table 3.2 average wall y+ values with mixing plane simulations at best efficiency point are listed. Similar values were observed at other operating conditions also. These values are considered as reasonably acceptable [1]. Although there is a scope for improvement in the values of y+ in runner but it was considered that, improving mesh quality in this region would make mesh more non-orthogonal because of the reasons mentioned earlier in section 3.1.3.

Patch	y+ value
Guide vane hydrofoil	10.15
Blade pressure side	27.21
Blade suction side	49.70
Runner Hub	46.73
Runner Shroud	59.14

Table 3.2: Wall y+ values

3.2.1 Best Efficiency Point

At best efficiency point counter rotating swirl and back flow was observed in the inlet region of diffuser as can be seen in the Figure 3.4 which is not a expected. This is because at BEP the whirl component of absolute flow must be zero, which implies that all the angular momentum has been utilized. Figure 3.5 shows the distribution of static pressure on pressure side of runner blades. It can be observed that pressure is evenly distributed on the blade surface which is expected behaviour at best efficiency point. At the trailing edge the negative pressure can be explained by sudden increase in velocity as flow leaves the trailing edge and enters the swirling region. The swirl causes formation of a very low pressure region at the inlet of draft tube, which causes back flow. This can be observed in Figure 3.6, where it is seen that tangential velocity is symmetrically distributed with positive values in near wall region and axial velocity has positive values at the central region of draft tube.

Distribution of static pressure at the cut plane in runner can be seen in Figure 3.7 where it can be observed that, pressure distribution is almost similar but at the same time minor perturbations are visible in case of frozen rotor especially at the leading edge of the suction side of the blade. This can be understood by considering, that frozen rotor simulations present the snapshot of flow at a particular relative position between guide vane and runner.

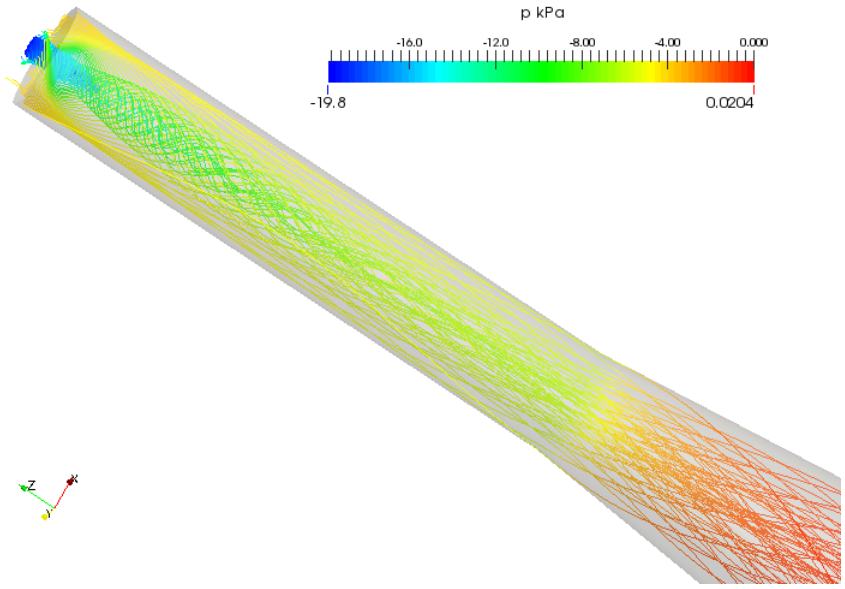


Figure 3.4: *Streamlines originating from runner outlet, at BEP coloured with pressure, with mixing plane interface*

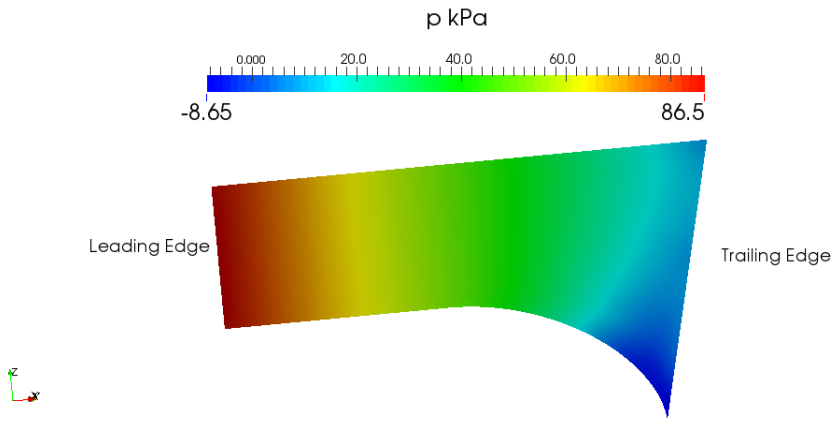


Figure 3.5: *Distribution of static pressure on blade pressure side at BEP, with mixing plane interface*

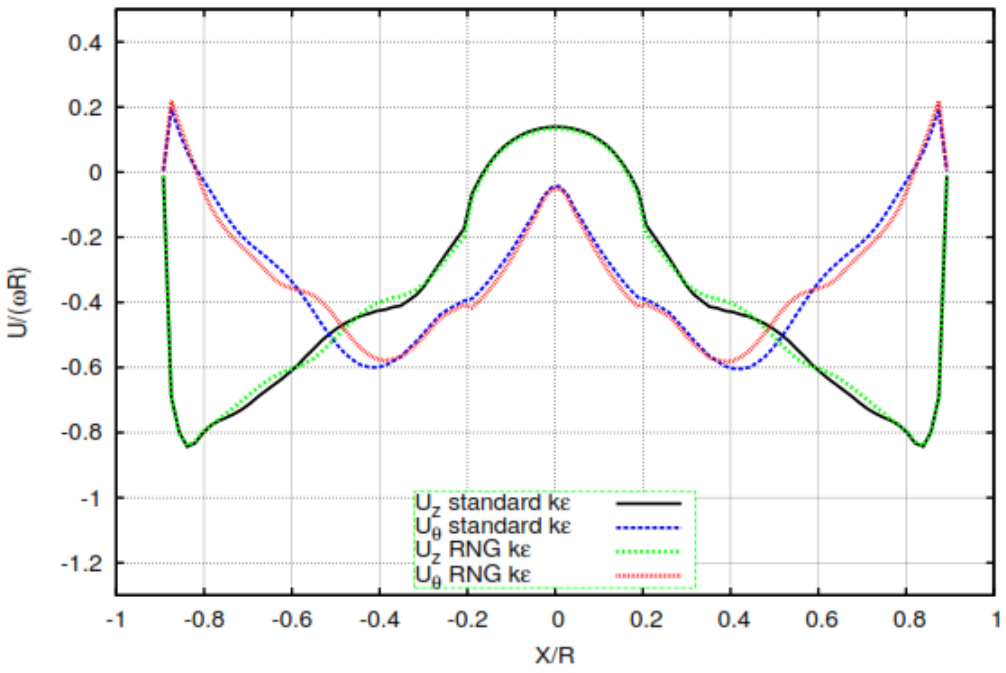
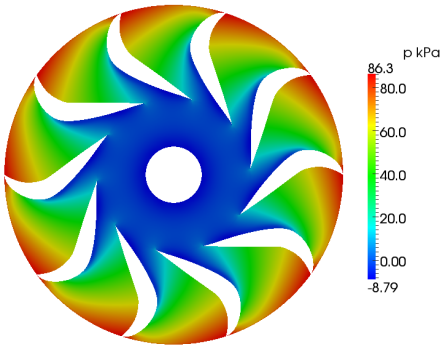
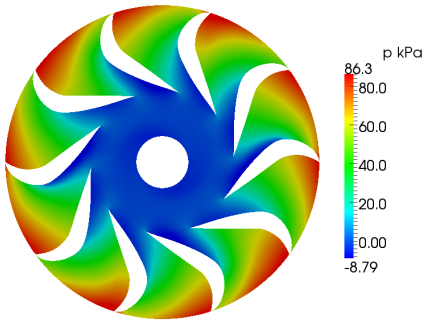


Figure 3.6: *Axial (Uz) and Tangential Velocity (U_θ) profiles at Inlet of Draft Tube on diametrical line perpendicular to stream wise direction, X axis is normalised by radius and Y axis is normalised by peripheral velocity at BEP, with mixing plane interface*



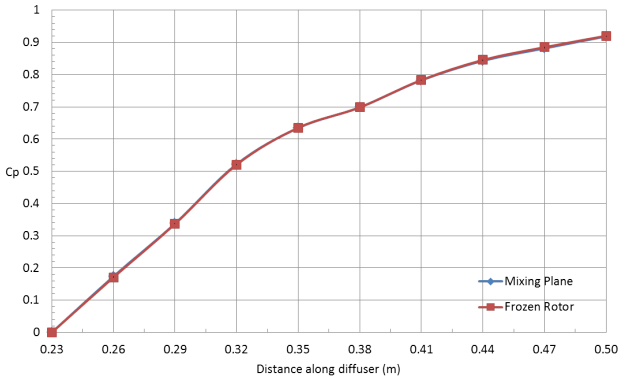
(a) Mixing Plane



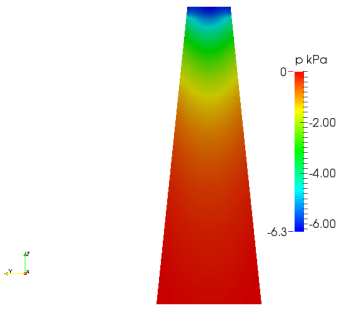
(b) Frozen Rotor

Figure 3.7: *Pressure distribution in runner (cut plane at z = 0.004) at best efficiency point*

In Figure 3.8a the plot of coefficient of pressure is given (corresponding to conical region, as shown in Figure 3.8b). It is seen that most of pressure recovery of about 70 % occurs in early region of the conical section of draft tube and after that slope of the curve reduces and ends at a value of 0.92. This is a justified value as the best efficiency point, pressure recovery should be high in draft tube.



(a) *Graph showing variation of Cp*



(b) Pressure Distribution

Figure 3.8: *Performance of draft tube at BEP*

3.2.2 Part Load

A counter clockwise swirl is observed at part load also, as can be seen by axial and tangential velocity profiles in Figure 3.9 and also in streamlines in Figure 3.10. In this case back flow is predicted higher in case of standard k ϵ than RNG model. The curve for tangential velocity has a lesser tangential component in positive direction than at BEP. Figure 3.11 shows the distribution of static pressure on pressure side of runner blades. It can be seen that pressure is no longer as evenly distributed as at BEP. Now pressure drops from value of 97.8 kPa to approximately 70 kPa in around one third of blade and rest of pressure drop occurs in the rest two thirds of the region.

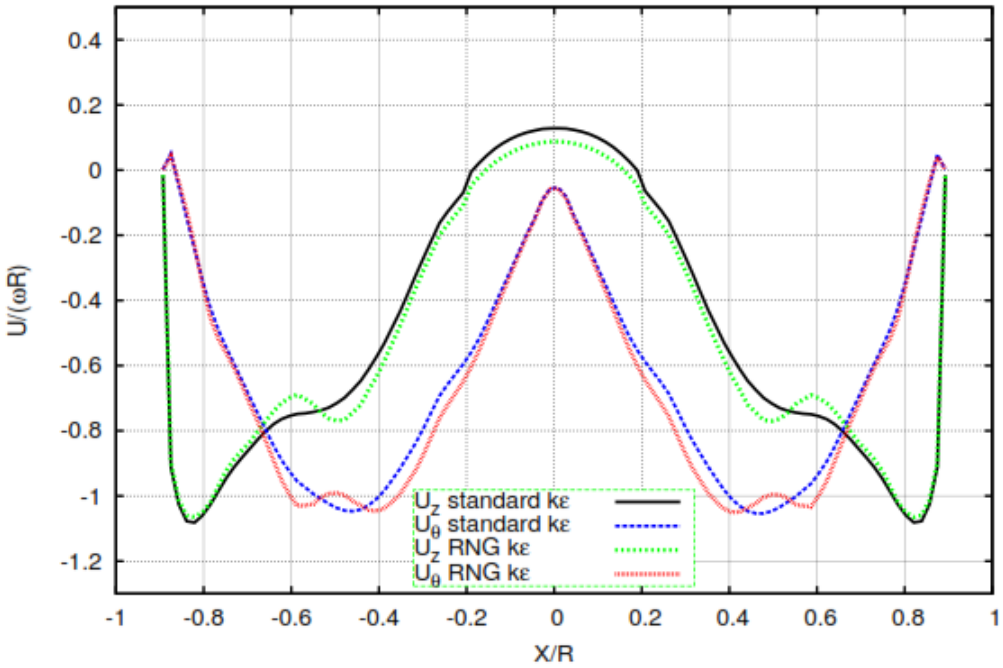


Figure 3.9: *Axial (Uz) and Tangential Velocity (U_θ) profiles at Inlet of Draft Tube on diametrical line perpendicular to stream wise direction, X axis is normalised by radius and Y axis is normalised by peripheral velocity at part load, with mixing plane interface*

Difference in pressure distribution in runner with mixing plane and frozen rotor simulations is now more pronounced as can be observed in Figure 3.12a and 3.12b. It can be seen that pressure distribution on leading edges of blades is highly asymmetric in case of frozen rotor simulations. This is a typical part load characteristic as the wakes are formed due to shift from optimal angle of attack of fluid on runner blades at part load, this is visible with frozen rotor simulations (although unphysical) whereas with mixing plane, these wakes are averaged out at the interface of runner and guide vanes.

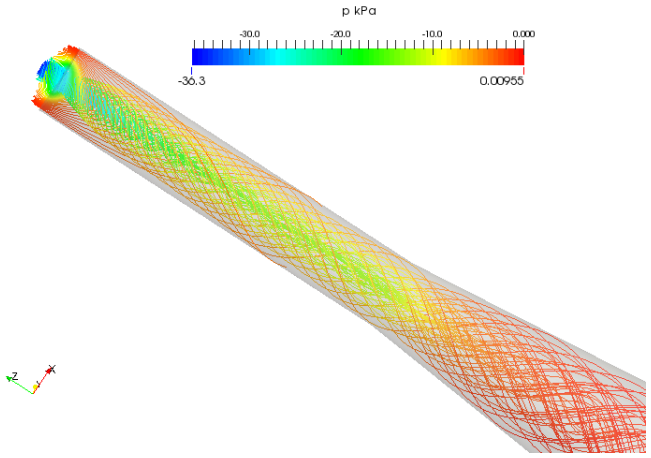


Figure 3.10: *Streamlines originating from runner outlet, from line perpendicular to rotation axis, coloured by pressure at part load, with mixing plane interface*

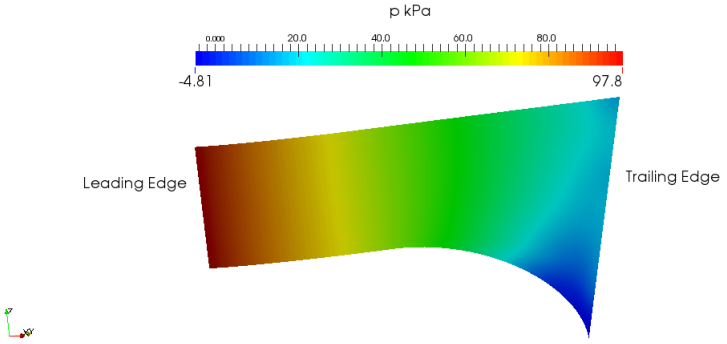
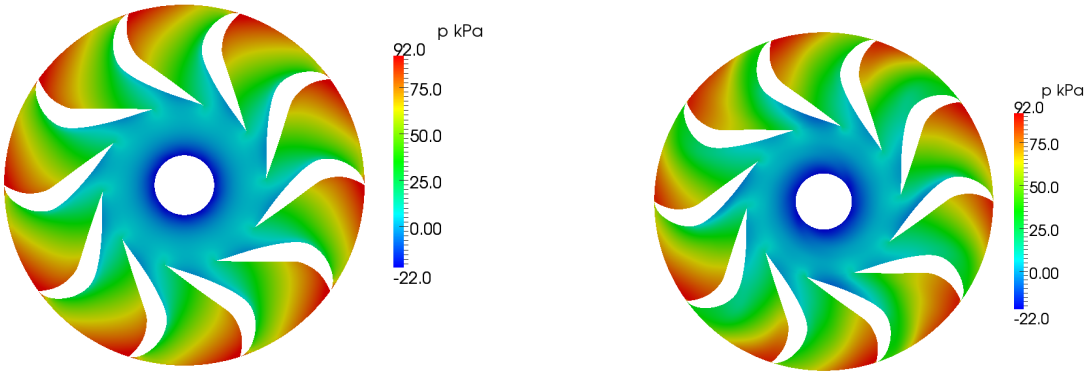


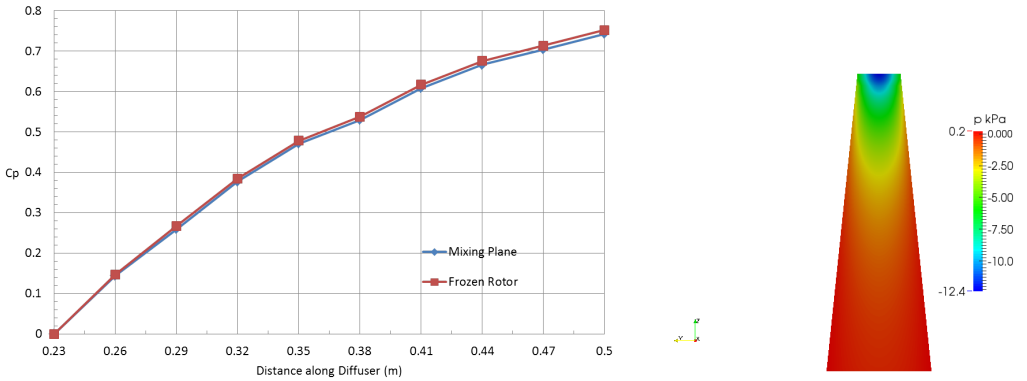
Figure 3.11: *Distribution of static pressure on pressure side of runner blade at part load , with mixing plane interface*



(a) Mixing Plane (b) Frozen Rotor

Figure 3.12: *Pressure distribution in runner (cut plane at z = 0.004) at part load*

In plot of coefficient of pressure, see Figure 3.13a (corresponding to conical region, as shown in Figure 3.13b) it can be observed that there are minor discrepancies in values of Cp calculated with frozen rotor and mixing plane simulations. The graph suggests that total pressure recovery is lower than in case of best efficiency point and more over it can also be observed that graph rises steeply up to a certain distance approximately 50 % of diffuser length and then it decreases, but the reduction in slope is not very much as in case of BEP. The final value is around 0.75 which can be expected at part load.



(a) *Graph showing variation of Cp* (b) Pressure Distribution

Figure 3.13: *Performance of draft tube at part load*

3.2.3 High Load

Like other operating points counter clock wise swirl is again observed in this condition as can be seen in Figure 3.14 and 3.15 . A interesting feature is pointed from the velocity profiles in Figure 3.14, the tangential velocity profile no more jumps to slightly positive values before becoming negative. The reason for this behaviour is unknown and could be possibly investigated in future studies.

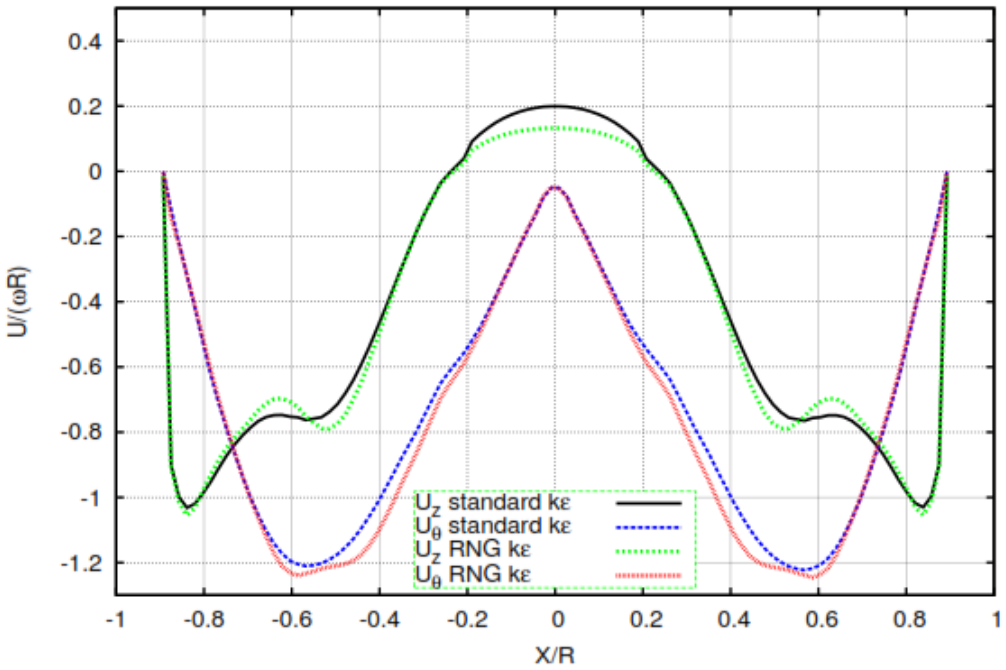


Figure 3.14: *Axial (Uz) and Tangential Velocity (U_θ) profiles at Inlet of Draft Tube on diametrical line perpendicular to stream wise direction, X axis is normalised by radius and Y axis is normalised by peripheral velocity at high load, with mixing plane interface*

Figure 3.16 shows the distribution of static pressure on pressure side of runner blades and it is similar to distribution at part load. Difference in pressure distribution in runner with mixing plane and frozen rotor simulations can be seen in Figure 3.17. The difference in minimum and maximum pressure is high as compared to previous operating conditions, this clearly indicates high load on runner. In this case the difference between pressure distribution with frozen rotor and mixing plane simulations is very strongly visible at the leading edge of blades.

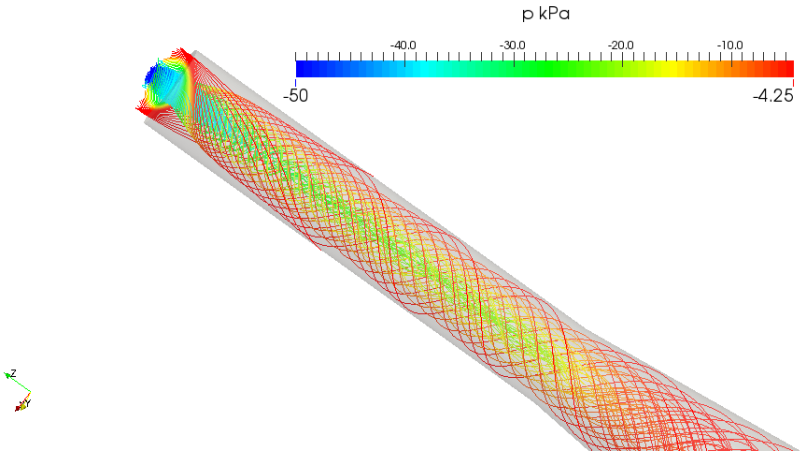


Figure 3.15: *Streamlines originating from runner outlet, perpendicular to rotation axis, coloured by pressure at high load, with mixing plane interface*

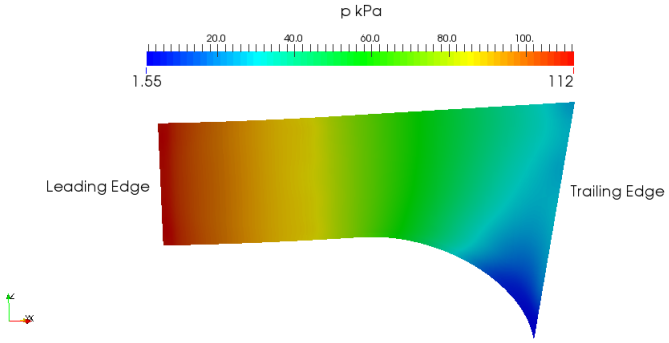
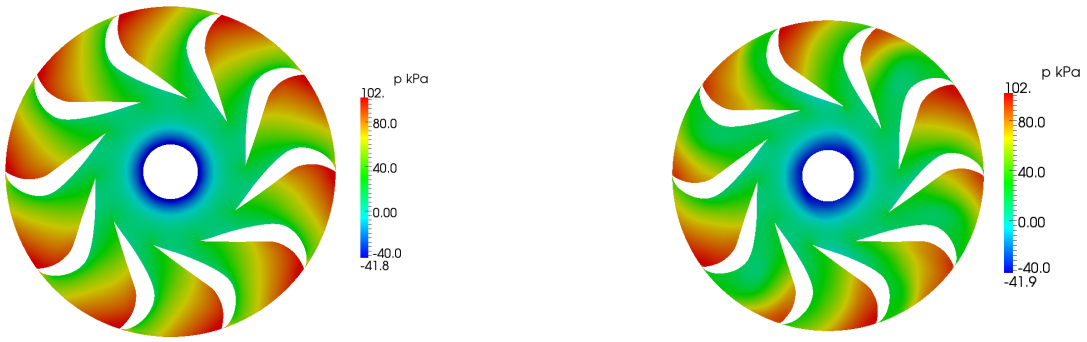


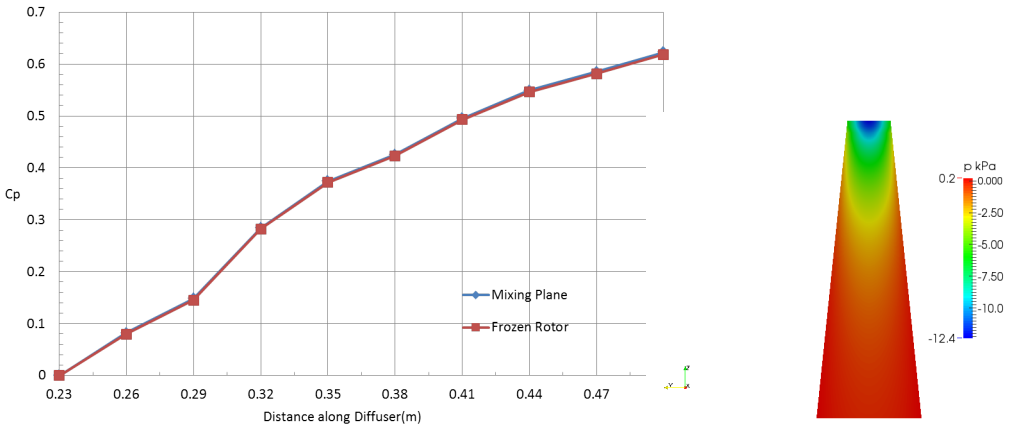
Figure 3.16: *Distribution of static pressure on pressure side of runner blade at high load, with mixing plane interface*



(a) Mixing Plane (b) Frozen Rotor

Figure 3.17: *Pressure distribution in runner (cut plane at z = 0.004) at high load*

It can be seen from Figure 3.18a and it's corresponding pressure distribution Figure 3.18b, that unlike other operating conditions the graph does not rise sharply in the initial regions rather the slope is less to a certain point and after that it rises to a value of 0.4 and after that there is a steady rise and it ends at around 0.62, which suggests that pressure recovery is quite low at high load.



(a) *Graph showing variation of Cp* (b) Pressure Distribution

Figure 3.18: *Performance of draft tube at high load*

Chapter 4

Conclusion and scope for future work

In this work a mesh of acceptable quality has been made on open source software Salome and then simulated on OpenFOAM. The results obtained can be regarded of reasonable quality. A considerable time and focus has been made on mesh generation. Since there was no previous work available that author was aware off, the approach for working was trial and error, for example a considerable amount of time was spent on Hexablock module in Salome, but author could not make any mesh using this module.

There is a lot of scope for future work in terms of parametrisation with Salome. For complete parametrisation the geometry must be made on Salome itself as the imported geometries offer only limited scope for parametrisation as only mesh distribution is parametrised. Salome can be tuned for parametrisation via only terminal, this would improve the total time for case setup even more.

The design of the draft tube could be altered by using a conical section at the outlet of runner and a elbow type design at later part. This kind of design could be parametrised for testing the effects of diffuser angle and length for maximum pressure recovery. The guide vane design can also be changed to increase the area occupied by guide vanes.

In OpenFOAM one interesting, type of simulation can be carried out using cyclic AMI which like GGI is used for coupling of two non conformal meshes. Like GGI a function object for cyclicAMI could be written to check the flux imbalance across AMI patch. This could be one of the ways to compare these two boundary conditions.

Bibliography

[1] B. Andersson, R. Andersson, L. Håkansson, M. Mortensen, R. Sudiyo, and B. Van Wachem. *Computational fluid dynamics for engineers*. Cambridge University Press, 2011.

[2] M. Beaudoin and H. Jasak. Development of a generalized grid interface for turbomachinery simulations with OpenFOAM. In *Open Source CFD International Conference*, volume 2, 2008.

[3] T. Behrens. OpenFOAM's basic solvers for linear systems of equations. 2009.

[4] O. Bergman. Numerical investigation of the flow in a swirl generator, using OpenFOAM. Master's Thesis, 2010.

[5] F. Blaim, O. Borm, T. Fröbel, and H. Kau. Implementation of rotor-stator interfaces in OpenFOAM. 2008.

[6] G. Cheng and S. Farokhi. On turbulent flows dominated by curvature effects. *Journal of fluids engineering*, 114(1):52–57, 1992.

[7] J. Denton. The calculation of three-dimensional viscous flow through multistage turbomachines. *Journal of turbomachinery*, 114(1):18–26, 1992.

[8] S. Derakhshan and A. Mostafavi. Optimization of GAMM francis turbine runner. *World Acad. Sci. Eng. Technol*, 59:717–723, 2011.

[9] P. Drtina and M. Sallaberger. Hydraulic turbines—basic principles and state-of-the-art computational fluid dynamics applications. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 213(1):85–102, 1999.

[10] E. Furbo. Evaluation of RANS turbulence models for flow problems with significant impact of boundary layers. Master's Thesis, 2010.

[11] W. Gyllenram. *Analytical and Numerical Studies of Internal Swirling Flows*. PhD thesis, Chalmers University of Technology, 2008.

[12] H. Jasak. *Error analysis and estimation for the finite volume method with applications to fluid flows*. PhD thesis, Imperial College London (University of London), 1996.

[13] H. Jasak and M. Beaudoin. OpenFOAM turbo tools: From general purpose cfd to turbomachinery simulations. In *ASME-JSME-KSME 2011 Joint Fluids Engineering Conference*, pages 1801–1812. American Society of Mechanical Engineers, 2011.

[14] H. Jasak, A. Jemcov, and Z. Tukovic. OpenFOAM: A C++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20, 2007.

[15] J. Kortelainen. Meshing tools for open source CFD-a practical point of view. *VTT, Espoo, Finland, Tech. Rep*, 2009.

[16] B. Lewis. *Improving Unsteady Hydroturbine Performance During Off-design Operation by Injecting Water from the Trailing Edge of the Wicket Gates*. PhD thesis, Penn State University, 2014.

[17] P. Maruzewski, H. Hayashi, C. Munch, K. Yamaishi, T. Hashii, H. Mombelli, Y. Sugow, and F. Avellan. Turbulence modeling for francis turbine water passages simulation. 12(1):012070, 2010.

[18] S. Mauri. Numerical simulation and flow analysis of an elbow diffuser. 2002.

[19] E. Nilsson, I. Nilsson, O. Thullin, and S. Samuelsson. Förbättring av Francisturbin. Bachelor's Thesis, 2010. Institutionen för Tillämpad mekanik, Chalmers Tekniska Högskola, Goteborg, Sweden.

[20] H. Nilsson. Turbomachinery training at OFW8. 2013.

[21] M. Page, M. Beaudoin, and A.-M. Giroux. Steady-state capabilities for hydroturbines with OpenFOAM. In *IOP Conference Series: Earth and Environmental Science*, volume 12, page 012076. IOP Publishing, 2010.

[22] M. Pathak, A. Dewan, and A. Dass. Effect of streamline curvature on flow field of a turbulent plane jet in cross-flow. *Mechanics Research Communications*, 34(3):241–248, 2007.

[23] O. Petit. *Towards Full Predictions of the Unsteady Incompressible Flow in Rotating Machines, Using OpenFOAM*. PhD thesis, Chalmers University of Technology, 2012.

[24] O. Petit, M. Page, M. Beaudoin, and H. Nilsson. The ERCOFTAC centrifugal pump OpenFOAM case-study. *3rd IAHR International Meeting of the Workgroup of Cavitation and Dynamic Problems in Hydraulic Machinery and Systems*, pages 523–532, 2009.

[25] J. Rygg. CFD Analysis of a Pelton Turbine in OpenFOAM. Master's thesis, 2013.

[26] S. Sickel, M.-C. Yeung, and M. J. Held. A comparison of some iterative methods in scientific computing. *Summer Research Apprentice Program*, 2005.

[27] R. Susan-Resiga, G. D. Ciocan, I. Anton, and F. Avellan. Analysis of the swirling flow downstream a Francis turbine runner. *Journal of Fluids Engineering*, 128(1):177–189, 2006.

[28] C. Trivedi, M. J. Cervantes, B. Gandhi, and O. G. Dahlhaug. Experimental and numerical studies for a high head Francis turbine at several operating points. *Journal of Fluids Engineering*, 135(11):111102, 2013.

[29] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

Appendix A

Procedure for Mesh Generation in Salome

Salome can be downloaded from the official website. The version used for this work version 7.4.0. The installation instructions for both Windows and Linux platforms are available on the official website.
In case of any doubts, Help module can be loaded from the toolbar. It presents the detailed description of a particular feature with it's TUI command which can be used for creating script. All the geometry files which will be used for following the later sections are available via this link at Web Link: https://www.dropbox.com/sh/tis96j8h9zietmf/AADtKbe3giBbRW55hRxzP0usa?dl=0

A.1 2D Mesh Generation in Salome

A.1.1 Working in GUI

The aim of this section is to give user introduction to geometry module of Salome and create a 2D mesh. As a first example a 2D mesh is made. The goal is to make a disk by specifying centre of disk, it's radius and it's orientation. First geometry is made and then a sample blocking strategy is presented for creating a mesh. This is useful in turbine components such as draft tube and spiral case, where circular cross sections are present.

1. Open Salome and load the geometry module.
2. Go to : New Entity, Primitives, Disk, Keep default radius and orientation.
3. New Entity, basic, Isoline, select Disk 1 in object browser, U isoline, value 0.5.
4. Click Apply, Now select V isoline, value 0.5, Click Apply & Close.
5. Go to : Operations, Partition, Objects : Disk1.
6. Tool Objects : Hold ctrl and select Isoline1 and Isoline2. Click Apply & Close.
7. Select Partition1 : with single click & click New Entity, explode.
8. In Select Sub shape types : select edge.
9. Right Click Partition 1 and select "Show only Children".
10. In Graphical window : select any circular edge (arc). It will turn white.
11. New Entity, Point, third button on top, keep 0.5 parameter. Vertex1 comes in object browser.
12. Now a middle point on circular edge is created.
13. New Entity, Basic, vector, first button on top.
14. Select Vertex1 as point 1 and O (origin : first item in tree list) as point 2.
15. A vector is now made pointing in the direction to origin.
16. Now go to Operations, Transformation, Translation, third button
17. In object, place edge1 and under vector place vector1
18. Check : Activate Distance and give value of 45 under distance.
19. Do this for rest of three edges and Figure A.1
20. Proceed by selecting 4 translated edges by holding shift key.
21. Operations, Partition, there should be 4 objects in the Objects container.
22. Click Apply and close. Select Partition 2 & explode it into edges like in Step 7.
23. In total there must be 20 edges in tree browser.
24. Now right click on graphics window and select Hide All

25. Right click partition2 and select ”Show only children”

26. Also switch on 4 edges (Edge 1,2,7,8) of the disk created initially.

27. Select 8 smaller edges which are overlapping and delete them to obtain Figure A.2,edges to be deleted are 9,10, 13-16, 19,20. Tip : In step 19, Use Operations, transformations, multirotation to get 3 other vectors. Step 27 works like trim function.

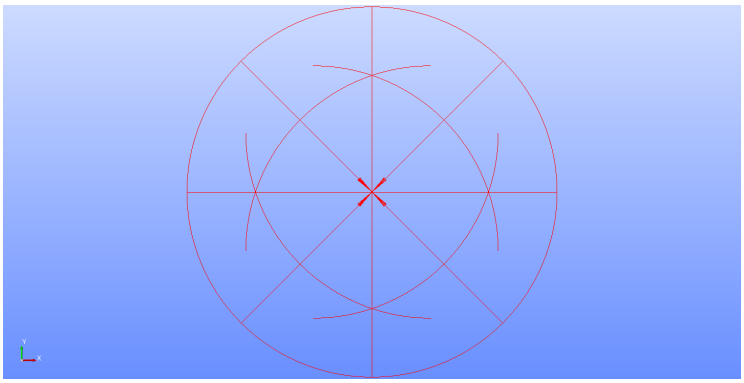


Figure A.1: Step19

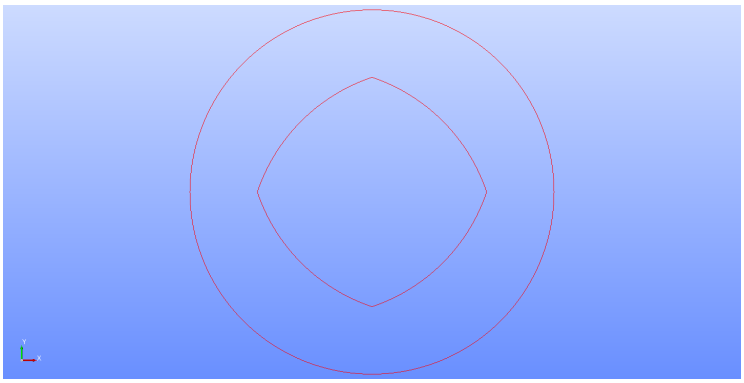


Figure A.2: Step27

28. Now blocks have to be created, New Entity, Blocks, Quadrangle Face, second button on top.

29. In Edge 1, select any outer edge and in Edge 2 select corresponding inner edge. As a preview a pink block will appear.

30. Repeat previous step for other 3 edges. See Figure A.3

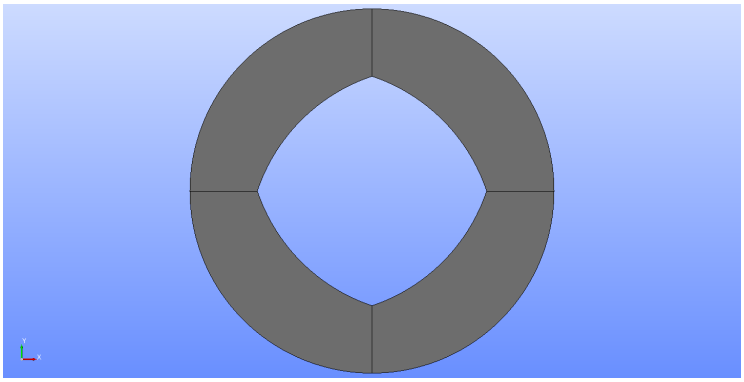


Figure A.3: Four quadrangle faces

31. Now hide the 4 blocks created. Again Go to New entity, blocks, Quadrangle face
32. Now select third button on top of box and select 4 inner edges.
33. Tree browser should now show 5 Quadrangle faces.
34. The order of edge selection must be outer edge to inner edge, every time otherwise there are problems while creating mesh.
35. Hold shift and select 5 Quadrangle faces in tree browser, right click show only.
36. Keeping them selected, go to New Entity, build Compound, select apply and close. Compound is like a bag containing objects which may be of different type.
37. Now select Repair, Glue Edges, keeping the default tolerance, go to second button on top and click Glue edges.
38. Now the result will be like as shown in Figure A.4. These edges are the common edges which are shared by the four quadrangle blocks.
39. Select the first button on top again and select Compound1 again, click Apply and Close.
40. Now click Compound1 again and go to repair, "What is" option, it can be noticed that there are 16 edges, check the same for Glue1, there will be four lesser edges.
41. As some last steps, now the edges and faces must be named for the mesh.
42. Select Glue 1, Go to Transformation Blocks, propagate and click Apply and Close.
43. It can be seen that three compounds are now created. Two Compounds are for inner and outer edge, one contains radial sections.

44. Holding Shift key select Compound 1 and Compound 3, Go to New Entity Group, Union Group

45. Only click on Apply, keeping the dialogue box On, Right Click Graphical Window, and select delete, press OK.

46. Rename Compound 2, to radial and Union 1 to Circular.

47. Now, There is a geometry with a blocks. The Next step would be create a mesh

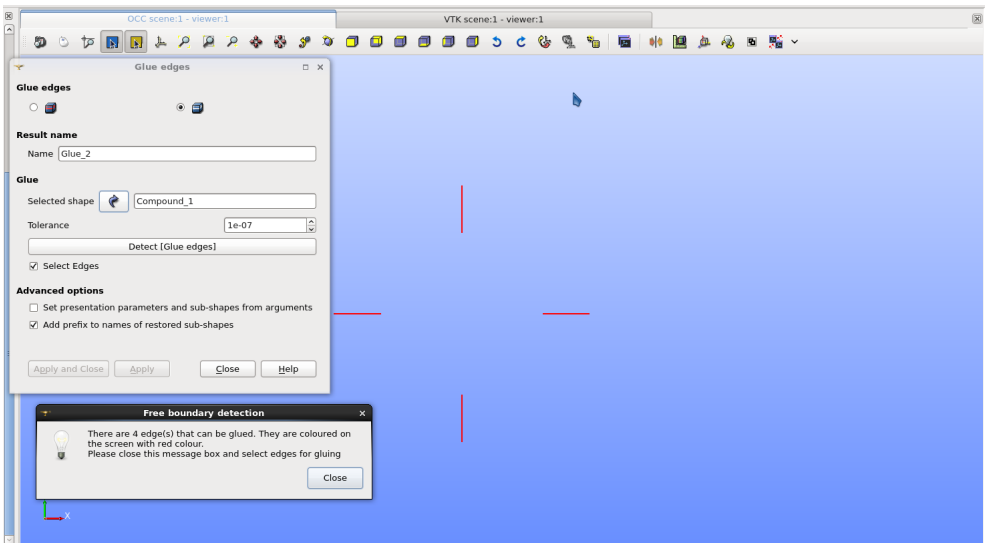


Figure A.4: Preview of Edges to be Glued

48. Load the Mesh Module.

49. Go to Mesh ,Create Mesh, Glue 1, Click Apply and Close, like in the Figure A.5.

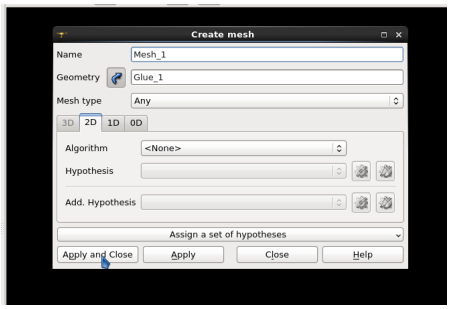


Figure A.5: Create Empty Mesh

50. There is now a empty/null mesh on the object.

51. Now, Right Click on mesh 1, select create submesh. Specify Circular as name and Geometry Object by selecting Circular in tree browser.

52. In Algorithms specify Wire Discretisation and in Hypothesis specify Nb. Segments.

53. The screen should look like as in Figure A.6.

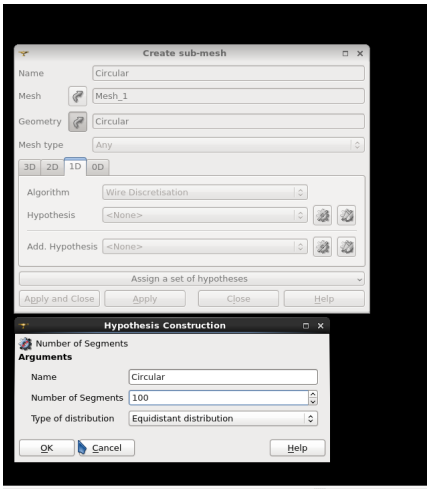


Figure A.6: Specifying Nodal Distribution

54. Give Value as 100 as argument to Nb. Segments.

55. Now it is specified that each edge in the set called circular must be broken down into 100 edges equally.

56. Next Click OK, Apply and Close. A submesh has been created on the child object.

57. Do the same step for Radial object also but instead of Specifying equidistant distribution in the hypothesis section specify scale distribution with scale factor as 5 and Number of Segments 20. One can notice the option of specifying reverse edges, this is useful, if while creating blocks there would have been a mismatch as warned earlier.

58. Right Click on Mesh1 (empty mesh) and Click Compute, and it should look like Figure A.7. By default the view is in wireframe mode, user can change this by right click graphical window and change to nodes in display mode.

59. Again Right Click Mesh1 and in 2D tab select Algorithm as quadrangle mapping. Click Apply and Close.

60. Now right click in object browser on Mesh1 and Click Compute to obtain Figure A.8

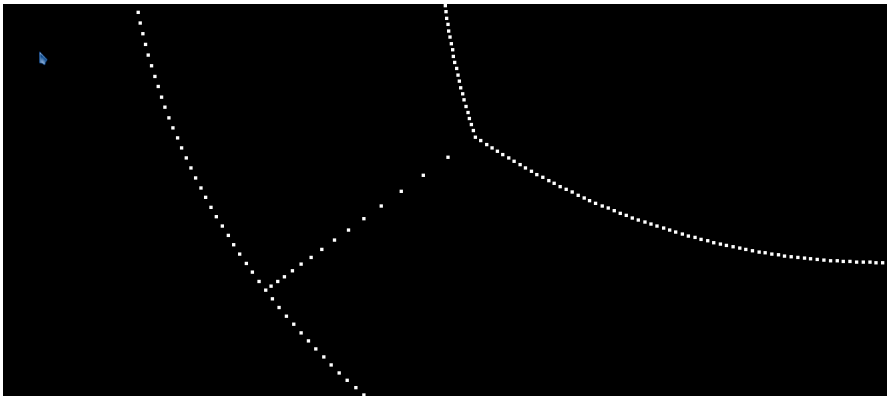


Figure A.7: Fine Mesh in Near Wall Region

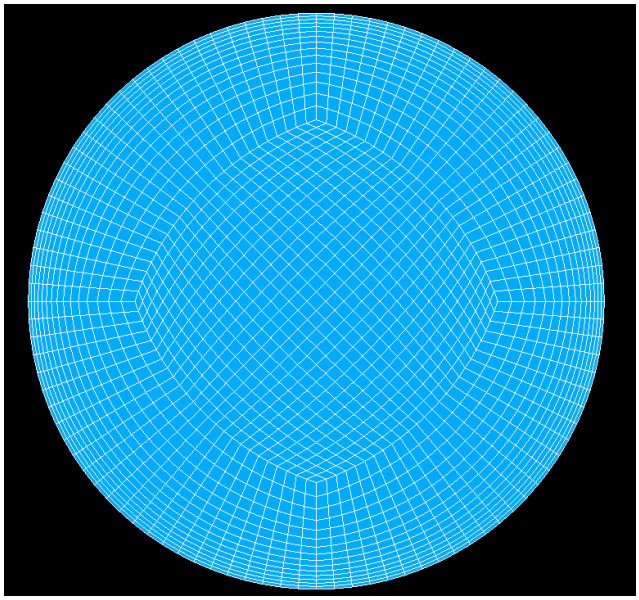


Figure A.8: Final Mesh

A.1.2 Python Script

Following is a python script doing exactly the same functions as done in GUI but it is made more generic. The aim of this script is that with any given radius, it's orientation and radial distance a 2D mesh can be made.

1. Launch Salome Study by specifying working directory.

```
import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'{working directory}')
```

2. Launch Geometry module and Specify Origin and local coordinate axes.

```
import GEOM
from salome.geom import geomBuilder
import math
import SALOMEDS


geompy = geomBuilder.New(theStudy)

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

Radius = 100
Distance = Radius*0.45
```

3. Specify radius and distance, size of the inner block is specified by distance parameter which is fraction of radius.

4. Define the function and it's arguments.

```
def Ogridpattern(radius, list1, list2, distance):

center = geompy.MakeVertex(list1[0],list1[1],list1[2])
vector = geompy.MakeVectorDXDYDZ(list2[0],list2[1],list2[2])
Distance = radius*distance
Disk_1 = geompy.MakeDiskPntVecR(center, vector, radius)
```

5. Define isolines and partition disk with them. true/false stands for U/V isolines.

```
Isoline_1 = geompy.MakeIsoline( Disk_1, True, 0.5 )
Isoline_2 = geompy.MakeIsoline( Disk_1, False, 0.5 )
Partition_1 = geompy.MakePartition
([Disk_1], [Isoline_1, Isoline_2], [], [], geompy.ShapeType["FACE"], 0, [], 0)
[Face_1,Face_2,Face_3,Face_4] =
geompy.ExtractShapes(Partition_1, geompy.ShapeType["FACE"], True)

list_of_edges = geompy.ExtractShapes(Partition_1, geompy.ShapeType["EDGE"], True)
```

6. Next lines of script are used to make the code generic. This is necessary because the with change in orientation the labels of the objects change and it becomes difficult to identify which object name stands for which object.
So when it is required to carry out some operation on one object, it gets complicated. For example if two lines are to be deleted from a set of lines. The name of these two edges will be different in different orientation.

```
# Make three empty lists

    Length_List = []
Useful_List = []
Translated_List = []

m = len(list_of_edges)

# Loop over all edges to find the length
# Basic properties function finds out geometrical properties of objects
# This function results into array, first element of this array contains length

for i in xrange(0,m):
```

```
Basic_Properties = geompy.BasicProperties(list_of_edges[i])
Length = Basic_Properties[0]
Length_List.append(Length)
Length_List.sort(reverse=True)
# Length of the elements is stored in a list in ascending order

for i in xrange(0,m):

Basic_Properties2 = geompy.BasicProperties(list_of_edges[i])
Length2 = Basic_Properties2[0]
#print Length2

if(Length2 == Length_List[0] or Length2
== Length_List[1] or Length2 == Length_List[2] or
Length2 == Length_List[3]):

#print "Match Found"
Useful_List.append(list_of_edges[i])

# Useful list contains four outer edges of equal length
```

7. Creating vectors and translating circular arcs along the vectors.

```
# Loop over the list and create mid points on them
# From these mid points create vectors pointing to origin
m = len(Useful_List)

for i in xrange(0,m):

p1 = geompy.MakeVertexOnCurve(Useful_List[i],0.5)

V1 = geompy.MakeVector(p1, center)

# Translate the four outer edges to a distance specified

T1 = geompy.MakeTranslationVectorDistance(Useful_List[i], V1, Distance)

Translated_List.append(T1)
```

8. Partition the translated edges with each other

```
    Translations_Partition = geompy.MakePartition
([Translations], [], [], [], geompy.ShapeType["EDGE"], 0, [], 0)

TEL = []

LLT = []

ULT =[]

TEL = geompy.ExtractShapes(Translations_Partition, geompy.ShapeType["EDGE"], True)
# As previously smaller edges are located

m = len(TEL)

for i in xrange(0,m):

BPT = geompy.BasicProperties(TEL[i])
LT = BPT[0]
#print Length
LLT.append(LT)
LLT.sort(reverse=True)

    for i in xrange(0,m):

BPT2 = geompy.BasicProperties(TEL[i])
LT2 = BPT2[0]
#print LT2

if(LT2 == LLT[0] or LT2 == LLT[1] or LT2 == LLT[2] or LT2 == LLT[3]):

#print "Match Found"
ULT.append(TEL[i])
```

9. Make the blocks and put them in a Compound

```
face_list = []
```

```
for i in xrange(0, 4):
F1 = geompy.MakeQuad2Edges(Useful_List[i], ULT[i])
     face_list.append(F1)
O_Grid_Pattern = geompy.MakeShell([face_list[0], face_list[1], face_list[2],
face_list[3], Internal_Face])
geompy.addToStudy(O_Grid_Pattern, "O_Grid_Pattern")

Pattern_1 = Ogridpattern(100, [0,0,0], [0,0,1], 0.45)
```

A.2 Diffuser Mesh

The following section explains a script which has been used to parametrise the geometry of diffuser. Basically strategy for 2D blocking remains same as above. The difference is that now from 2D blocks 3D blocks are being made. Note that few commands in this section are different from previous explanation, in principle the aim to familiarise the reader with most of commands.

1. The first lines of the code do exactly same, as detailed in previous two sections. Additionally a circular face is broken down into four edges. Though edge is not right terminology for an arc but that's how it is addressed in software.

```
##
same as before
##



geompy = geomBuilder.New(theStudy)

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

# The radius of the disk and its orietation + it's center is specified here
# distance factor takes a value between 0 and 1

def Ogridpattern(radius, list1, list2, distance):

center = geompy.MakeVertex(list1[0],list1[1],list1[2])
vector = geompy.MakeVectorDXDYDZ(list2[0],list2[1],list2[2])
Distance = radius*distance
Disk_1 = geompy.MakeDiskPntVecR(center, vector, radius)



Isoline_1 = geompy.MakeIsoline( Disk_1, True, 0.5 )
Isoline_2 = geompy.MakeIsoline( Disk_1, False, 0.5 )
Partition_1 = geompy.MakePartition([Disk_1],
[Isoline_1, Isoline_2], [], [],
```

```
geompy.ShapeType["FACE"], 0, [], 0)


[Face_1,Face_2,Face_3,Face_4] =
geompy.ExtractShapes(Partition_1, geompy.ShapeType["FACE"], True)


[Edge_1,Edge_2,Edge_3] =
geompy.ExtractShapes(Face_1, geompy.ShapeType["EDGE"], True)

[Edge_4,Edge_5,Edge_6] =
geompy.ExtractShapes(Face_2, geompy.ShapeType["EDGE"], True)

[Edge_7,Edge_8,Edge_9] =
geompy.ExtractShapes(Face_3, geompy.ShapeType["EDGE"], True)

[Edge_10,Edge_11,Edge_12] =
geompy.ExtractShapes(Face_4, geompy.ShapeType["EDGE"], True)

list_of_edges =
geompy.SubShapeAllSortedCentres(Partition_1, geompy.ShapeType["EDGE"])

# Sub shape all sorted centres is a similar to extract shapes
# It is also used to explode a shape into sub shapes

p1 = geompy.MakeVertexOnCurve(list_of_edges[0],0.5)
p2 = geompy.MakeVertexOnCurve(list_of_edges[1],0.5)
p3 = geompy.MakeVertexOnCurve(list_of_edges[6],0.5)
p4 = geompy.MakeVertexOnCurve(list_of_edges[7],0.5)


# Make vertex on curve takes edge/curve as first argument
# Second argument is value of parameter lying between 0 and 1
# It must be notice that this function must be used carefully in scripts
# The parameters place a point on edge according to directions

V1 = geompy.MakeVector(p1, center)
V2 = geompy.MakeVector(p2, center)
V3 = geompy.MakeVector(p3, center)
```

```
V4 = geompy.MakeVector(p4, center)

T1 = geompy.MakeTranslationVectorDistance(list_of_edges[0], V1, Distance)
T2 = geompy.MakeTranslationVectorDistance(list_of_edges[1], V2, Distance)
T3 = geompy.MakeTranslationVectorDistance(list_of_edges[6], V3, Distance)
T4 = geompy.MakeTranslationVectorDistance(list_of_edges[7], V4, Distance)



Partition2 = geompy.MakePartition([T1, T2, T3, T4])

# Self partition command meaning these objects are partitioned by themselves
# Can be thought like a trim operation in common CAD Software.

list_of_edges2 =
geompy.SubShapeAllSortedCentres(Partition2, geompy.ShapeType["EDGE"])

# There are two commands being used to make Quadrangle face, one requires
# four edges as argument and other requires two edges


central_face = geompy.MakeQuad
(list_of_edges2[2], list_of_edges2[3], list_of_edges2[8],list_of_edges2[9])

quad_face1 = geompy.MakeQuad2Edges(list_of_edges[0], list_of_edges2[2])
quad_face2 = geompy.MakeQuad2Edges(list_of_edges[1], list_of_edges2[3])
quad_face3 = geompy.MakeQuad2Edges(list_of_edges[6], list_of_edges2[8])
quad_face4 = geompy.MakeQuad2Edges(list_of_edges[7], list_of_edges2[9])


# A quadrangle face is also required at the top
# Since an O grid in not being generated at the top part
# of diffuser which connects to the shaft of
# runner, it will be a circular face without any blocking

pure_circular_faces = geompy.MakeQuad
(list_of_edges[0], list_of_edges[1], list_of_edges[6], list_of_edges[7])

list_of_objects =
```

```
[central_face, quad_face1, quad_face2, quad_face3, quad_face4, pure_circular_faces]

return list_of_objects

#Ogridpattern(radius, list1[], list2[], distance)
--> Arguments taken by this function

# Next 4 lines call the function

Pattern_1 = Ogridpattern(47.7601468838, [0,0,-500.00], [0,0,1], 0.45)
Pattern_2 = Ogridpattern(19.652380356 , [0,0,-230.06], [0,0,1], 0.45)
Pattern_4 = Ogridpattern(19.6495005997, [0,0,-14.20], [0,0,1], 0.45)
Pattern_3 = Ogridpattern(4.0210511808 , [0,0,-8.17], [0,0,1], 0.45)
```

2. In the next lines of the script the conical section of the draft tube is being generated with blocks. A block must contain only six faces.

```
m = len(Pattern_1) + len(Pattern_2)
mn = len(Pattern_1)
c = Pattern_1 + Pattern_2

list_points = []
list_edges = []
list_pipes = []

for i in xrange(0, m):
s = geompy.MakeVertexOnSurface(c[i], 0.5, 0.5)
#geompy.addToStudy(s, "s")
list_points.append(s)

# Make vertex on surface works by parametrise surface into u,v
# For better understanding:
# Create a disk in Salome with default settings
# The disk is lying in x-y plane
# Go to New Entity > Points > 5th button on top
# The centre of this disk will have 0.5,0.5 as u,v parameters for locating
# it in 2D space
# Similar operating can be done for a rectangle also
```

```
for i in xrange(0, mn): #--------------> Can be omitted
edge = geompy.MakeEdge(list_points[i], list_points[mn+i])
#geompy.addToStudy(edge, "edge")
list_edges.append(edge)

for i in xrange(0, mn-1):
pipe = geompy.MakePipeWithDifferentSections([Pattern_1[i], Pattern_2[i]], [],
list_edges[0],False,False)

# Make pipe with different sections works like Skin/Loft
# command which is very commonly
# found in CAD software, it needs a list of cross sections
# with definition of path
# along which they will be joined.

faces = geompy.SubShapeAllSortedCentres(pipe, geompy.ShapeType["FACE"])
shell = geompy.MakeShell
([faces[0], faces[1], faces[2], faces[3], faces[4], faces[5]])
#geompy.addToStudy(shell, "shell")
solid = geompy.MakeSolid([shell])
list_pipes.append(solid)


Solids_Compound = geompy.MakeCompound(list_pipes)
#geompy.addToStudy(Solids_Compound, "Solids_Compound")

Diffuser_Part = geompy.MakeGlueFaces(Solids_Compound, 1e-07)
geompy.addToStudy(Diffuser_Part, "Diffuser_Part")


IsValid = geompy.CheckCompoundOfBlocks(Diffuser_Part)
if IsValid == 0:
raise RuntimeError, "Invalid compound created, or glue faces 1 by 1"

# On few occassions there may be a error
# reported "Tolerance value is too high"
# In such cases it's advisable to glue
# faces one by one.
```

```
else:
print "\n ********* Blocking is Complete, for this part ***************"
```

3. The geometrical sequence is such that from 6 faces a shell is made, then solid is created from this shell. Combination of such solids are combined by adding them into a compound. Finally block of compound is checked for errors (Check Compound of blocks command in Measures tab), see Figure A.9. To remove this error, adjacent blocks sharing a common face can then be glued like as shown in Figure A.10.

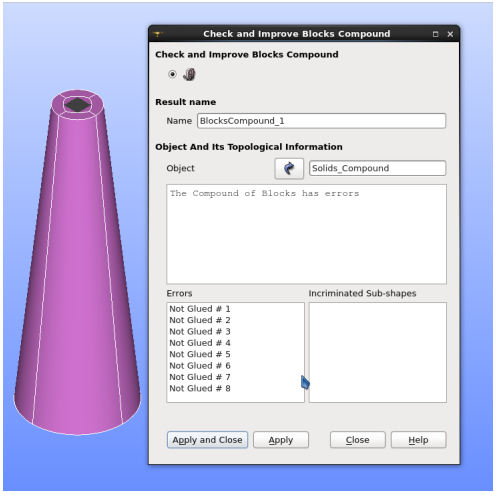


Figure A.9: Checking Compound of Blocks

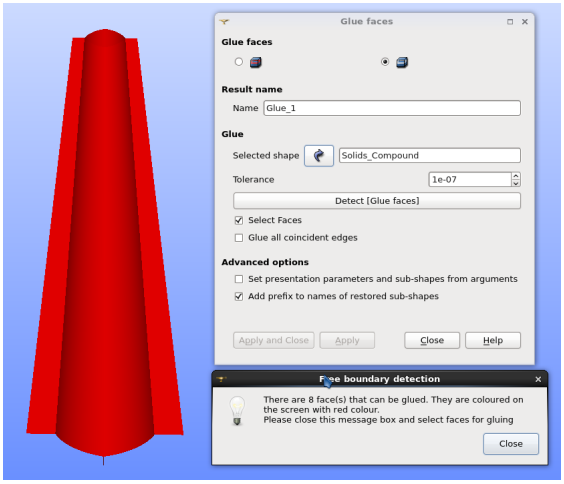


Figure A.10: Faces to be glued

```
# ------Now  blocking for axial part #
# First connect the shaft face to centre face of pattern 2
# Shaft Face means the top face of diffuser which
# would overlap with shaft of runner.


s1 = geompy.MakeVertexOnSurface(Pattern_2[0], 0.5, 0.5)
s2 = geompy.MakeVertexOnSurface(Pattern_3[5], 0.5, 0.5)
edge1 = geompy.MakeEdge(s1,s2)

#geompy.addToStudy(edge1, "edge1")


pipe_conical = geompy.MakePipeWithDifferentSections
([Pattern_2[0], Pattern_3[5]], [], edge1 ,False,False)

#geompy.addToStudy(pipe_conical, "pipe_conical") --------
---------> This is a shell

pipe_conical_faces =
geompy.SubShapeAllSortedCentres(pipe_conical, geompy.ShapeType["FACE"])

shell_conical = geompy.MakeShell
( [pipe_conical_faces[0], pipe_conical_faces[1],
pipe_conical_faces[2], pipe_conical_faces[3],
pipe_conical_faces[4], pipe_conical_faces[5] ])

solid_conical = geompy.MakeSolid([shell_conical])

#geompy.addToStudy(pipe_conical_faces[5], "pipe_conical_faces[0]")


# USE MAX AREA FOR MORE AUTOMATION
# Then we create a conical face
# geompy.addToStudy(Pattern_4[5], "face_for_cone")

edges_conical_1 = geompy.SubShapeAllSortedCentres
(Pattern_4[5], geompy.ShapeType["EDGE"])
edges_conical_2 = geompy.SubShapeAllSortedCentres
```

```
(Pattern_2[5], geompy.ShapeType["EDGE"])


list_external_faces = []
for i in xrange(0, 4):
c1 = geompy.MakeVertexOnCurve(edges_conical_1[i],0.5)
c2 = geompy.MakeVertexOnCurve(edges_conical_2[i],0.5)
ec = geompy.MakeEdge(c1, c2)
pipe_external = geompy.MakePipeWithDifferentSections
([edges_conical_1[i], edges_conical_2[i]], [], ec ,False,False)

pipe_external_faces = geompy.SubShapeAllSortedCentres
(pipe_external, geompy.ShapeType["FACE"])

list_external_faces.append(pipe_external_faces[0])


# Like Quadrangle face (2D block)
# Hexablock is a 3D entity
# It can be made from 2 faces, the two faces will be joined by 4 other faces
# along the shortest distance
# For creating blocks with non planar entities,there is option of specifying
# 6 faces. This is used later on

External_Hexa  = geompy.MakeHexa2Faces
(list_external_faces[3], pipe_conical_faces[5])

External_Hexa1 = geompy.MakeHexa2Faces
(list_external_faces[2], pipe_conical_faces[4])

External_Hexa2 = geompy.MakeHexa2Faces
(list_external_faces[1], pipe_conical_faces[1])

External_Hexa3 = geompy.MakeHexa2Faces
(list_external_faces[0], pipe_conical_faces[0])

Conical_compound = geompy.MakeCompound
([External_Hexa, External_Hexa1, External_Hexa2, External_Hexa3, solid_conical])
```

```
Conical_Completed = geompy.MakeGlueFaces(Conical_compound, 1e-07)
geompy.addToStudy(Conical_Completed, "Conical_Completed")

IsValid = geompy.CheckCompoundOfBlocks(Conical_Completed)

if IsValid == 0:
raise RuntimeError, "Invalid compound created"
else:
print "\n ***** Blocking is Complete, for Conical Part as well *****"
```

4. The blocking structure for conical part is completed at this step. Next step is to combine the conical and axial parts and glue them together for creating complete diffuser.

```
#---------Complete Diffuser

Full_Compound = geompy.MakeCompound([Conical_compound,Diffuser_Part])
Full_Diffuser = geompy.MakeGlueFaces(Full_Compound, 1e-07)

IsValid = geompy.CheckCompoundOfBlocks(Full_Diffuser)

if IsValid == 0:
raise RuntimeError, "Invalid compound created"
else:
print "\n ** Blocking is Complete, For complete geometry **"

geompy.addToStudy(Full_Diffuser, "Full_Diffuser")


print "\n ** Last step is to make edge groups
for sub-meshes and name the faces for creating patch name**"
```

5. As done for the disk it is necessary to create the compound of edges for defining the mesh properties. As a step further it is also required to specify the names of the patches for specifying boundary conditions in OpenFOAM. After edges have been named by building compounds, next step is to create faces. For this go to New Entity, Group, Create Group, select third button on top looking like a face,

name the group as walls, in graphical face select walls of diffuser holding the shift key, click Add. Refer Figure A.11

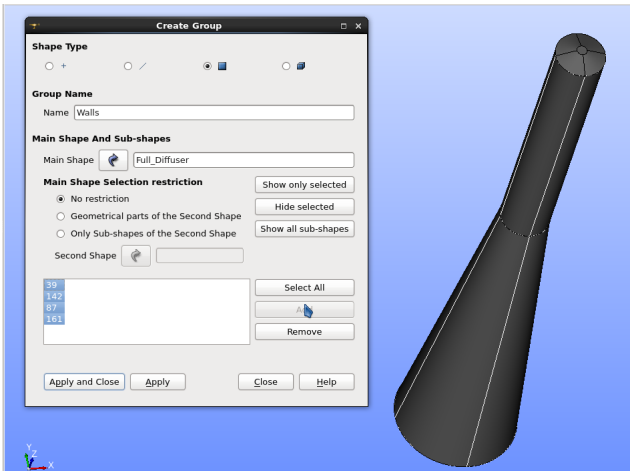


Figure A.11: Faces to be glued

```
# Edges groups for sub-meshes

[Diffuser_Long_Edges, Radial, Conical_Long_Edges, ARCS1, ARCS2] =
geompy.Propagate(Full_Diffuser)

# This way an entity is propagated into specified names.

# Create a Group with 'name' =
# On this object, types of subentity = Faces
# ID's with which they are identitified

Diffuser_Walls = geompy.CreateGroup
(Full_Diffuser, geompy.ShapeType["FACE"])
geompy.UnionIDs(Diffuser_Walls, [63, 103, 176, 5, 39, 87, 149, 166, 137])

Diffuser_Inlet = geompy.CreateGroup
(Full_Diffuser, geompy.ShapeType["FACE"])
geompy.UnionIDs(Diffuser_Inlet, [91, 22, 70, 53])

Diffuser_Outlet = geompy.CreateGroup
(Full_Diffuser, geompy.ShapeType["FACE"])
```

```
geompy.UnionIDs(Diffuser_Outlet, [173, 120, 142, 154, 161])

#geompy.addToStudyInFather ## Check syntax,
( Full_Diffuser, Diffuser_Long_Edges, 'Diffuser_Long_Edges' )

( Full_Diffuser, Radial, 'Radial' )

( Full_Diffuser, Conical_Long_Edges, 'Conical_Long_Edges' )

( Full_Diffuser, ARCS1, 'ARCS1' )

( Full_Diffuser, ARCS2, 'ARCS2' )

( Full_Diffuser, Diffuser_Walls, 'Diffuser_Walls' )

( Full_Diffuser, Diffuser_Inlet, 'Diffuser_Inlet' )
```

A.3 Runner

This section deals with introducing a concept of 3D blocking for Runner geometry.

1. First the geometry is loaded into Salome, using File Import Option
2. Next explode the main object into faces. A notification pops up, Shape contains 65 Faces, meaning that there are 65 faces in this geometry. Click OK. See Figure A.12

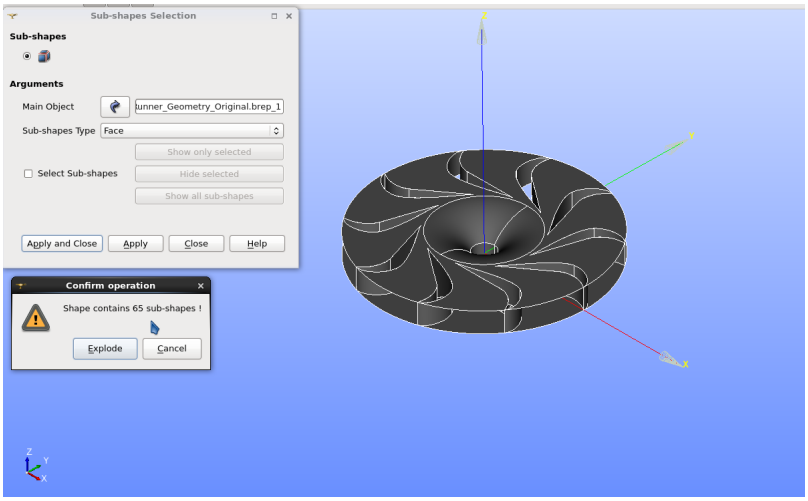


Figure A.12: Runner Geometry

3. Blocking is done only on a single blade passage and then rotated multiple times.
4. Right Click in Graphical Window and select Hide all.
5. Hold ctrl button and select following objects in the object browser with single click.
6. Object No.s Face 2, 3, 5, 6, 11, 16, 17, 21, 26, 27, 29, 34.
7. Go to New Entity, Build, Compound. Name it Control Volume.
8. Explode the compound into faces .Right Click Compound created, and select show only children
9. Right Click Original Runner File and select Conceal child items.
10. The tree browser looks more organised now. The graphical window should look like Figure A.13

11. A simple rule that must be kept in mind is that each block must have only 6 six faces always.

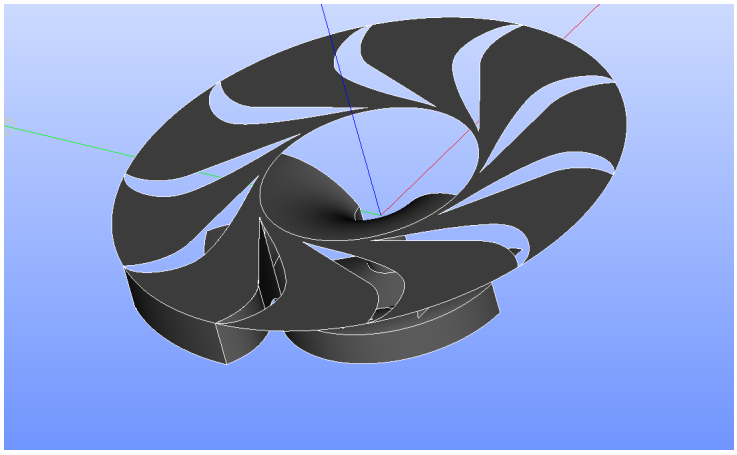


Figure A.13: Runner Control Volume

12. Again Right click on Graphical window and select hide everything

13. Now switch on the eye next to following objects to display them. Object No.s Face 66-71. The window will now have two faces for the both suction and pressure side of blade, Inlet face to runner and first part of shroud. The window should look like Figure A.14

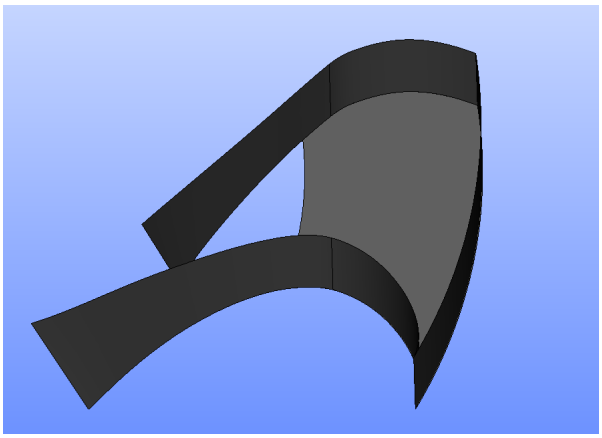


Figure A.14: Blades, Inlet and Shroud Part

14. Click New Entity, Basic, Line and make a line as shown in Figure A.15. With line dialogue box open take the cursor to the point and a small circular point will show

up as a preview, do the same for opposite vertex and preview of line will show. Click Apply and close.

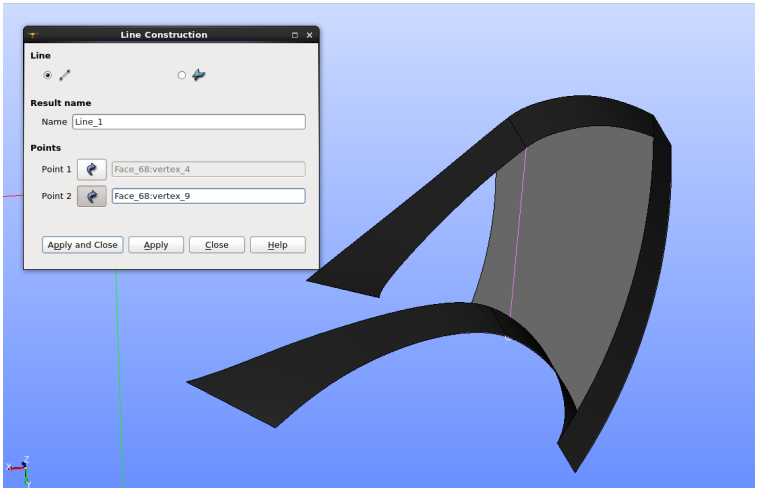


Figure A.15: Line on the Shroud

15. Again hide everything except Face 68 and line created in previous step. Explode Face 68 into edges, it will explode in 6 edges. Now graphical window should look like Figure A.16

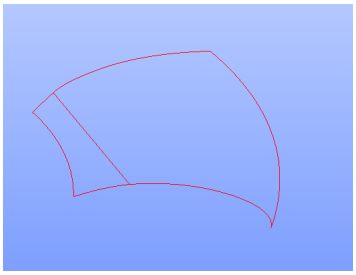


Figure A.16: Splitting shroud into blocks

16. Next build quadrangle faces (with four edges) from these edges as shown in Figure A.17

17. For simplicity, Rename face66 to Inlet, 67 ,69, 70, 71 to 77 as B1Suction, B1Pressure, B2Suction, B2Pressure, Shroud, Shroud2, centre,bottom, centre top, hub and shaft respectively

18. Switch on the display for hub as shown in the Figure, after that go to Operations, Transformations, projections, in Source vertex, edge or wire input Line1 (created in Step 14), on the target face select hub face. After clicking apply. Note the

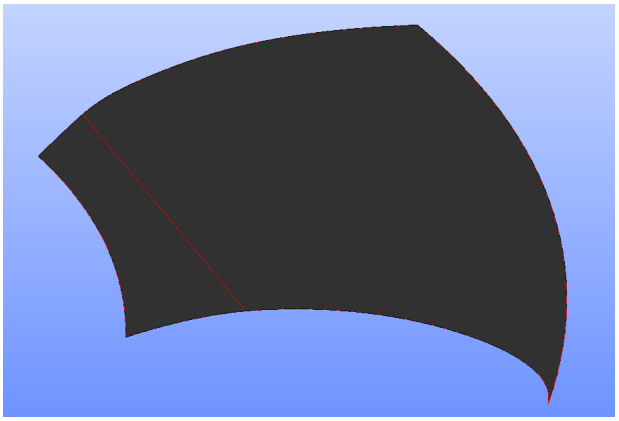


Figure A.17: Quadrangle faces

symbol of Projection in tree browser is of a compound, explode it into edge. Do the same for edge 5(created by exploding Face68). See Figure A.18

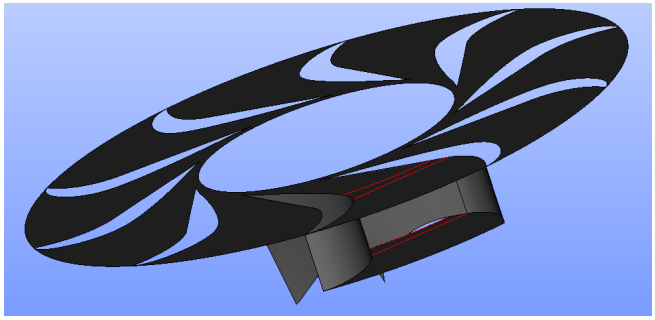


Figure A.18: Projections

19. Following last step two edges are obtained, using them as tool objects, partition the hub face. Explode the partition into Faces, Now hide partition, original hub face, and switch only children for partition just created. Name first face as hub part1 and second as hub part2. See Figure A.19

20. Hide all and switch on only the blades (both parts of pressure and suction side).

21. Create two points on the edge of the blade on the pressure side as shown in Figure A.20

22. After two points are made, go to New Entity, Generation, Revolution, select vertex1 (just made), select OZ as axis and 36 as value of rotation, since there are 10 runner blades. A preview will be shown as in Figure A.21. Do the same for other vertex as well. Two edges have been now generated.

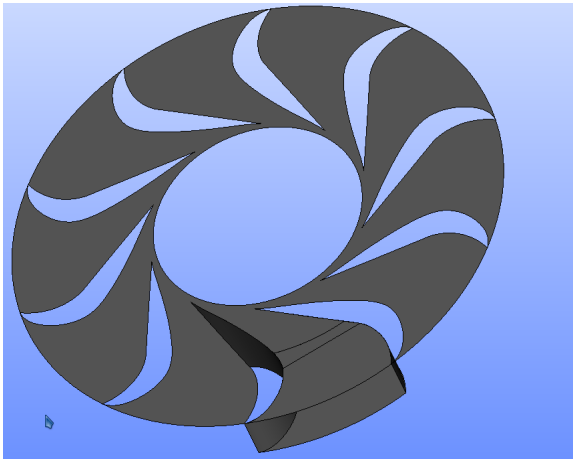


Figure A.19: Result of Partition

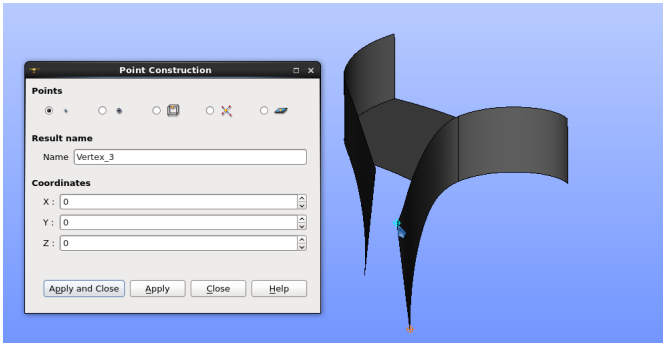


Figure A.20: Create Points

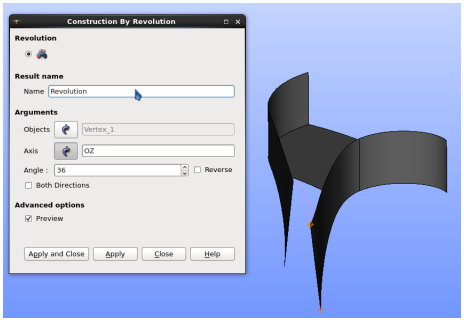


Figure A.21: Revolution

23. Next Step is to create a quadrangle face using trailing edges of blades and two edges or arcs made in last step. See Figure A.22

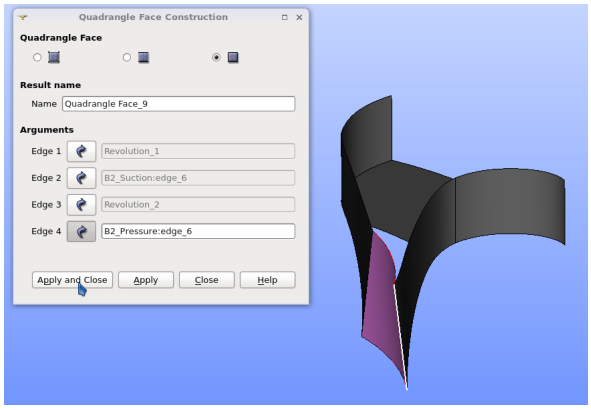


Figure A.22: Quadrangle Face

24. Next Switch on the display for Shroud and partition it with quadrangle face created previously. Delete the bigger part, the part useful is shown in yellow colour. See Figure A.23

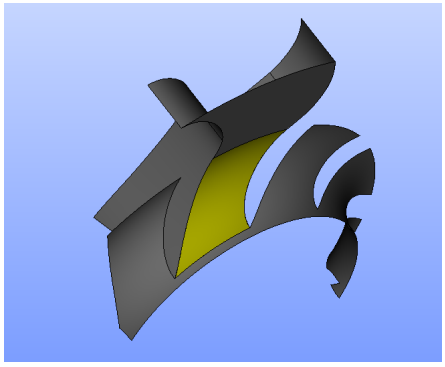


Figure A.23: New Shroud

25. Create two lines as shown in Figure A.24, using the line function and selecting the points by moving cursor over them. These lines must be made from vertex of the yellow part and hub part 2 (from step 19).

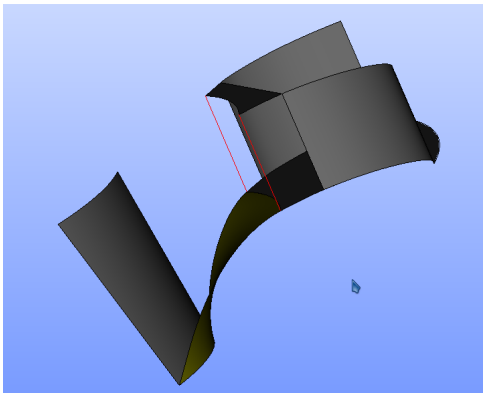


Figure A.24: working on blade

26. Now if the part of blades for both suction side and pressure side are switched on it may seem that these lines lie on them, in fact they do, but their might be a possibility that these faces are non planar and then what is being seen is not true actually. Just to be sure project these lines on the respective blades.

27. Partition the blades with these two projected edges. Now in total there are three parts of each blade (1 present originally, two created in last step). In Figure A.25, blades are coloured with default and red and green colour for easy identification.

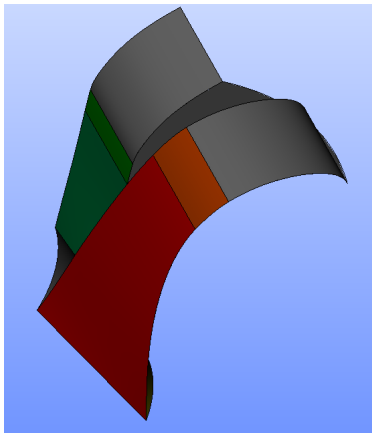


Figure A.25: working on blade

28. In the next step again create two more lines as shown in the Figure A.26, note we could also have used faces created previously.

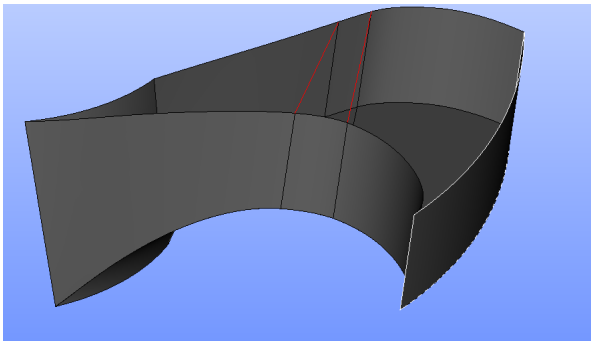


Figure A.26: Prepare blocks

29. Now create 4 Quadrangle faces as shown in the picture (two on top and two inside). Note one inside face is not seen in A.27, it is adjacent to face coloured in red.

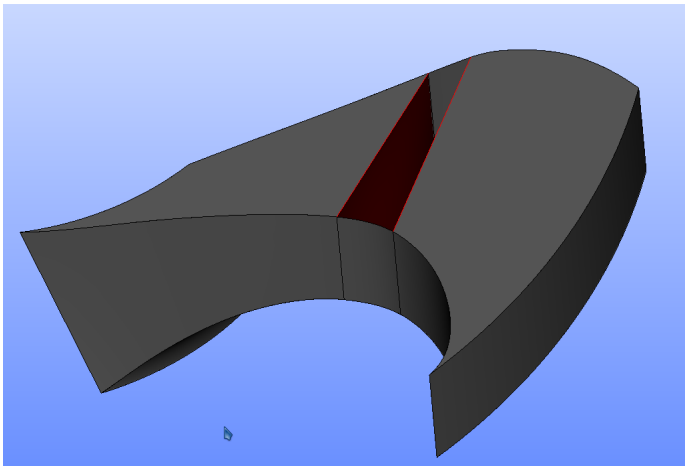


Figure A.27: Prepare blocks

30. Now using 6 faces, make shell 3 times and make solids from these shells. Note Salome may tell that "Unable to create solid from Unclosed Shape", if so, go to repair and use sewing option, with lesser than default tolerance. Now again try to create solid.

31. Three solids must be created from three shells.

32. Next put these solids into a compound. Go to view, display, and select wire frame.

33. Go to Repair, Glue faces, Click on second button on top and click Detect Glue Faces. It can be noticed that these two faces are common between blocks, if they are not glued, there will be duplicate elements. See Figure A.28

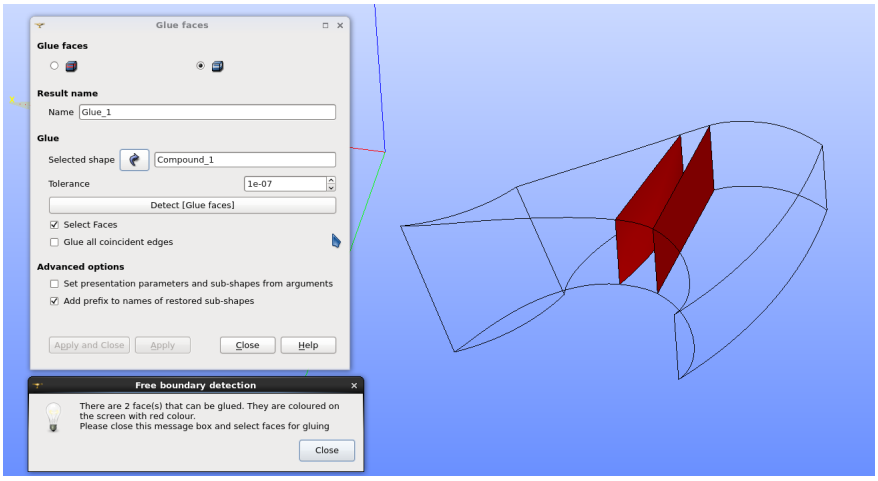


Figure A.28: Glue faces

34. Glue these faces. SAVE the work or export the Glue1 object in tree browser as a geometry using File, Export Button.

35. For creating periodic patches, create two lines from centre as shown in the Figure A.29

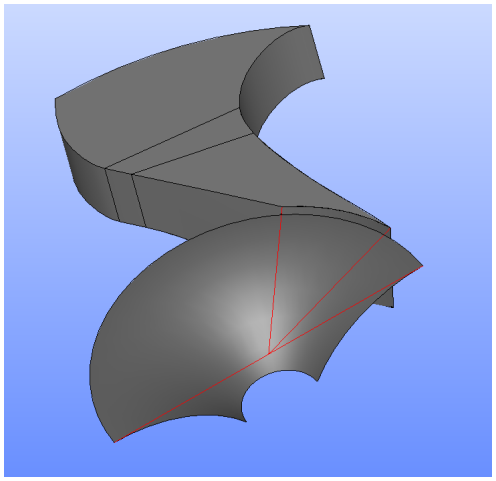


Figure A.29: Create periodic patches

36. Using extrusion along path, create a face as shown in Figure A.30

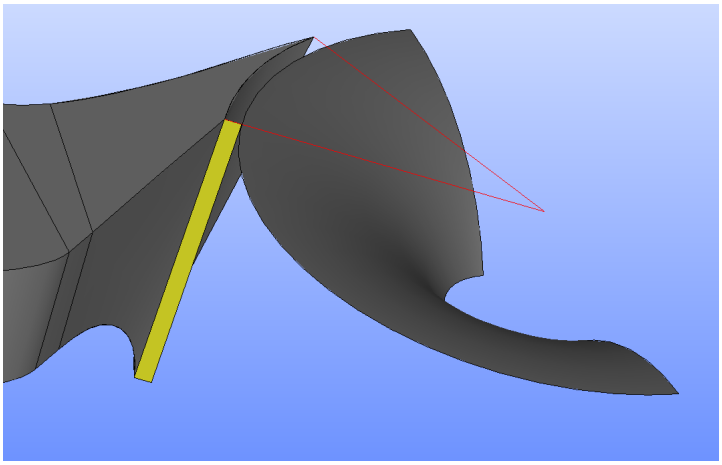


Figure A.30: Create periodic patches

37. Now this pipe(as named in object browser) can be rotated 36 degrees about central axis to generate a solid which will then be glued to the previous solid.For gluing tolerance would have to increased.

38. The next step is to create faces as shown in the next Figure this is done by using two functions, first extrusion along a path, in which the object to be extruded is side edge of hub and it should be extruded along path obtained by partition top face. The face obtained is coloured in yellow. Refer Figure A.31

39. From the object created in the previous step use the side edge, and revolve it by 36 degree to obtain the face in green colour as shown in the Figure A.32

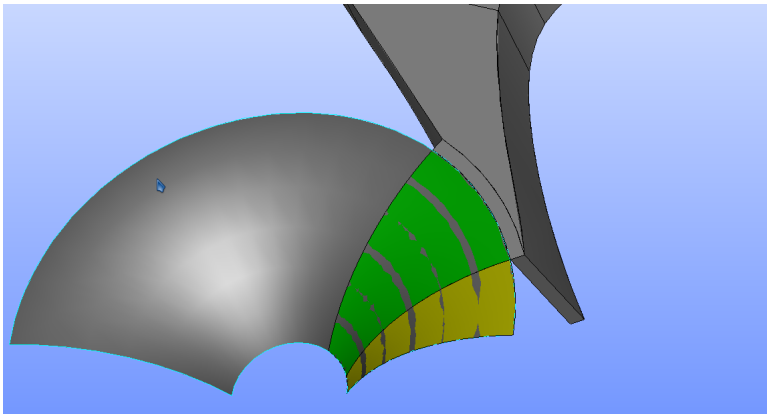


Figure A.31: Create periodic patches

40. The next step is to create face coloured in red as shown in the next Figure. This can be done by extruding the edge of the green face (of previous step).

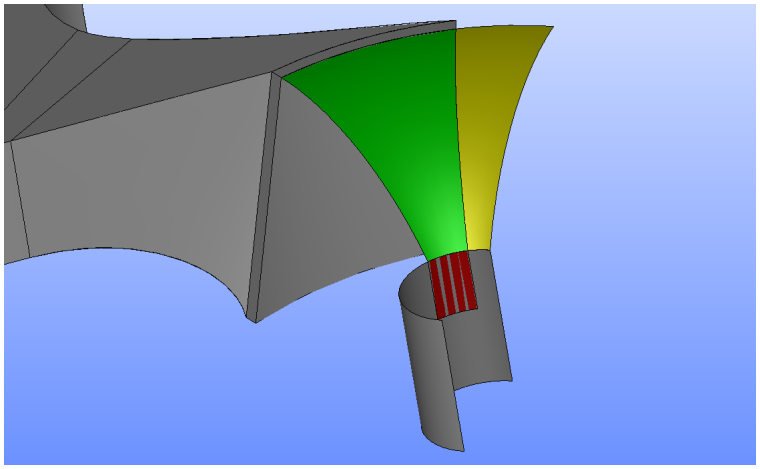


Figure A.32: Create periodic patches

41. Using the face obtained in previous step, a quadrangle face can be created shown with yellow colour in the next Figure A.33

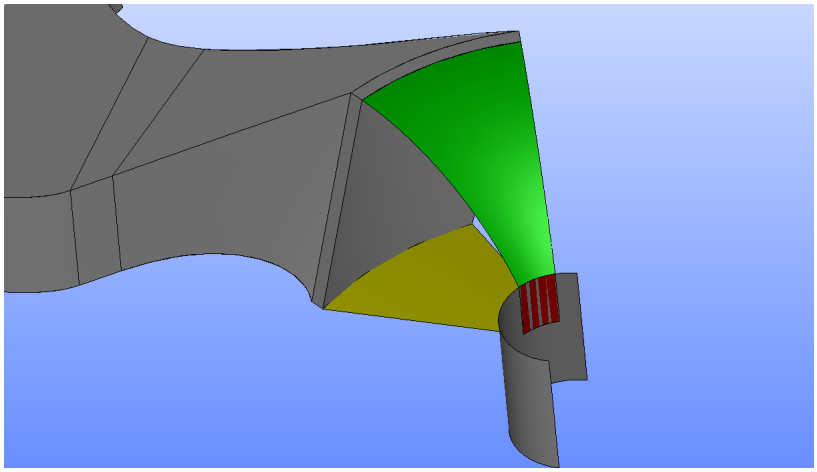


Figure A.33: Create periodic patches

42. With this periodic face can be created as shown with purple color in next Figure. Similarly periodic face must be made on other side as well.

43. Finally there are 6 faces from which a solid block can be created and glued to the previously created solid blocks, as shown in Figure A.34. In total there must be four blocks now.

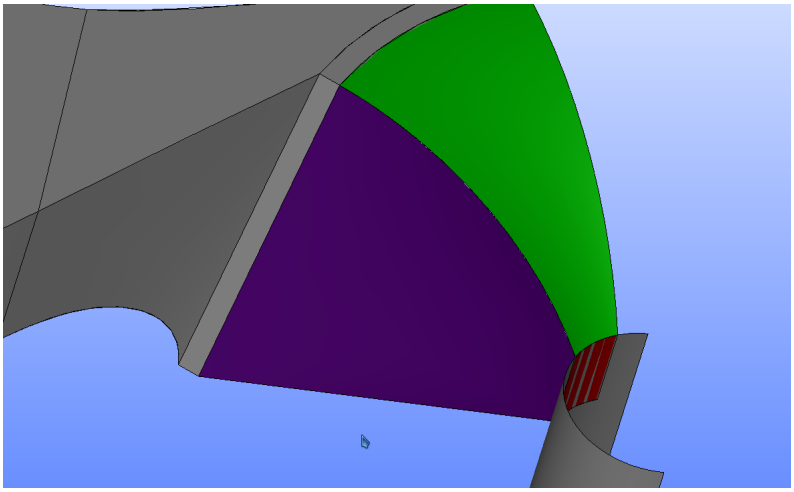


Figure A.34: Create periodic patches

44. Next Switch on the display for the outlet of runner.

45. Extrude the suitable face of the previously made block to such a distance so that it intersects the outlet face. Note that this changes the topology to a small extent but if real topology is followed it would lead to more blocks and more complexities. Refer Figure A.35

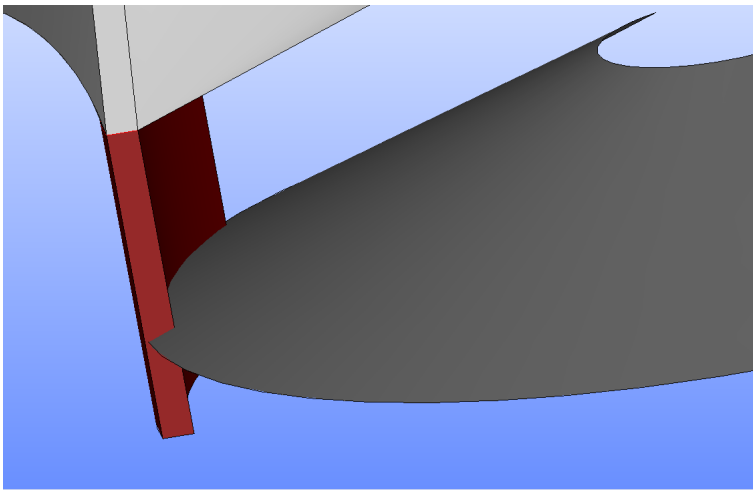


Figure A.35: Extrusion

46. Select the outlet face and extrusion just made and go to operations, Boolean, section and select them as the object 1 and object2. Result of this operation is that now a section is produced on the outlet face of the runner. The line produced

by this operation is coloured by red colour in Figure A.36.

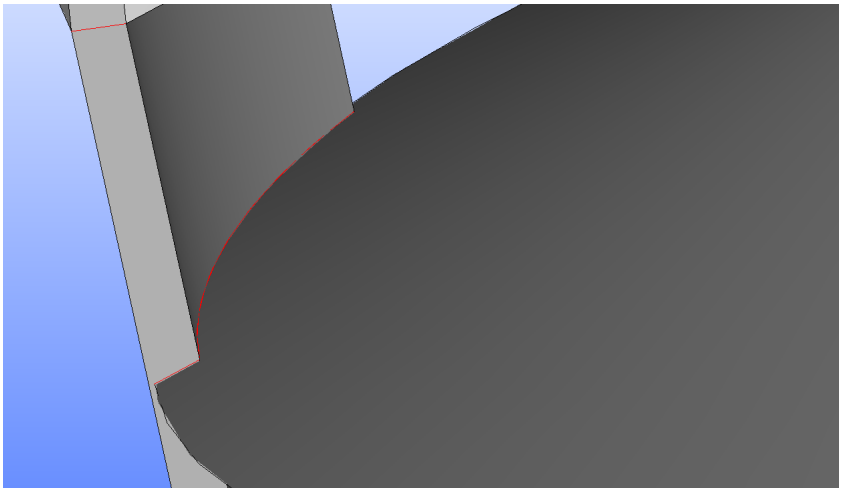


Figure A.36: Create projections

47. Partition the edges of outlet face with the section and create a quadrangle face as shown in Figure A.37.

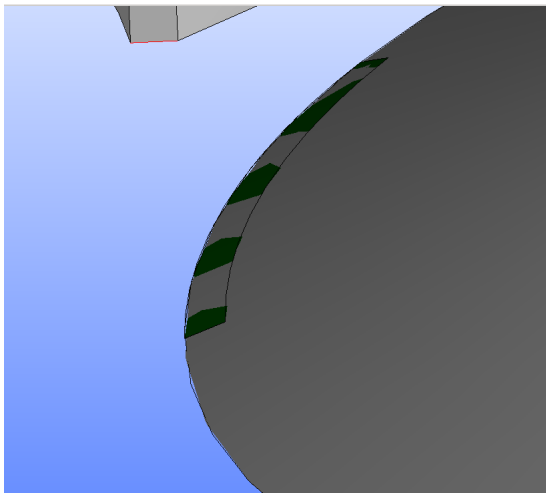


Figure A.37: partition

48. Make a hexahedral solid from new entity, blocks, using quadrangle face made in previous step and face used for extrusion. Glue it to previous made blocks.

49. Using suitable edges from previous blocks create projections on the outlet face, shown by red colour in Figure A.38.

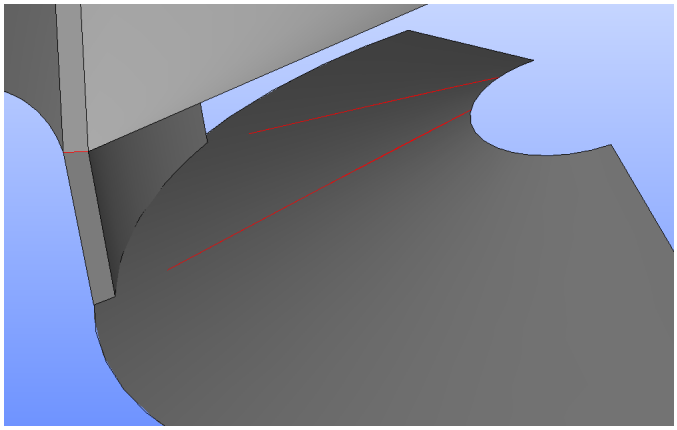


Figure A.38: Extrusion

50. Now partition the inner diameter/circumference of the outlet face with the projected lines. With the partitioned edge create a quadrangle face as shown in Figure A.39.

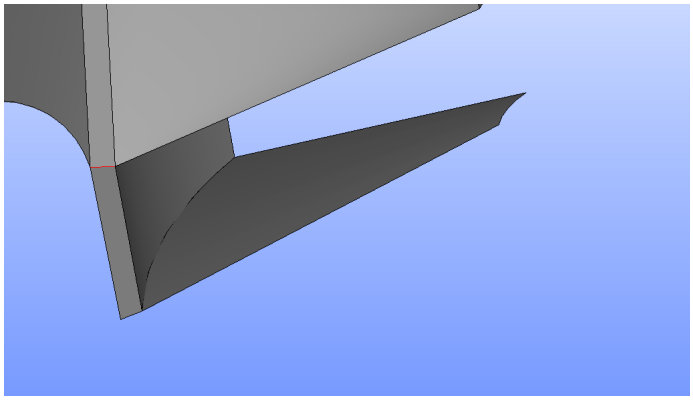


Figure A.39: Extrusion

51. Next make a hexahedral solid between the face just created and respective face above it in +Z direction. Refer Figure A.40

52. Glue this block to previously made blocks to obtain a single block. Using operations, transformations, Multi rotation rotate this block 10 times to obtain complete

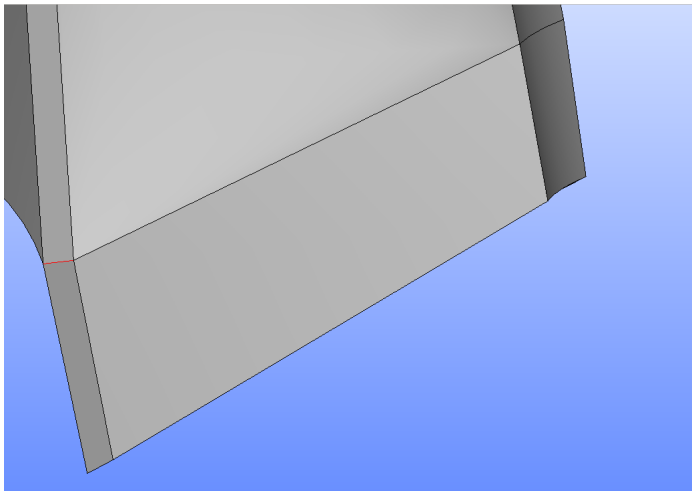


Figure A.40: Create block

geometry. Now glue all the faces together there should be 40 such faces which can be glued. Tolerance can be increased in this step if required.

53. Next select the glued blocks and go to Operations, block and click propagate. This will create 44 compound of edges.

54. Right click glued object and select show only children.

55. In tree browser holding control key select Compound Number 2, 4, 10, 15, 20, 27, 28, 31, 32, 33. Keeping them selected (marked in blue) go to New Entity, group, union group and name it as horizontal. Make sure there are 10 objects in the objects section. Click Apply and Close. If a less cluttered tree browser is required then just click apply and right click on graphical window and select delete/hide, this will delete/hide all the compounds which have been used to create union.

56. Select Compound Numbers 3, 5, 7, 9, 23, 24, 36, 37, 43, 44. and union them with name part1.

57. Select Compound Numbers 6, 8, 11, 14, 25, 26, 34, 35, 41, 42 and union them with name part2.

58. Select Compound Numbers 12, 13, 16, 19, 22, 29, 30, 38, 39,40 and union them by giving a name part3.

59. Lastly rename compound 1 to vertical, 17 to vertical 2, 21 to part5 and 18 to part4.

60. Save the geometry and in the module tool bar select Mesh.

61. In mesh module select Mesh, Create Mesh, and select the glued object as geometry. Click Apply and Close.

62. Right click Mesh 1 and select Create Submesh.

63. In the new dialogue box select vertical (by clicking in the tree browser) as Geometry, put the name of this sub mesh as vertical, in Algorithm select Wire Discretisation.

64. In Hypothesis click the gear icon and select Nb. Segments, again put name of this hypothesis as vertical and put Number of Segments as 10.

65. Do the same procedure for all the other compounds by naming the sub mesh and hypothesis and put values as listed in table A.1. Its extremely useful to put name of hypothesis because then nodal distribution can be varied later on.

Geometry group	Nb. Segments
part1	12
part2	4
part3	17
part4	2
part5	16
vertical	10
vertical2	6
horizontal	14

Table A.1: Values for Nb. Segments

66. After creating the submeshes, click on Mesh1, Select Compute. A 1D mesh has been computed.

67. Right Click Mesh1 ,select edit mesh/sub mesh and under 2D tab, select Quadrangle Mapping as algorithm. Leave other settings as it is. Right Click on Mesh1 and select compute.

68. Now a 2D mesh has been made with quadrangles.

69. Again Right Click Mesh1 and select edit mesh/sub mesh, under 3D tab, select Hexahedron (i,j,k) as algorithm and compute the mesh. See Figure A.41

70. Note that for running simulations in open foam, runner has to declared as cellZone. While there are procedures to this in Open FOAM but it can be directly done in

Salome itself. For this right click the final mesh and select create group, then click the last radio button, and click on button select all, Now group of volumes will be made under the mesh section. When converted to OpenFOAM format this group will show up as cellZone.

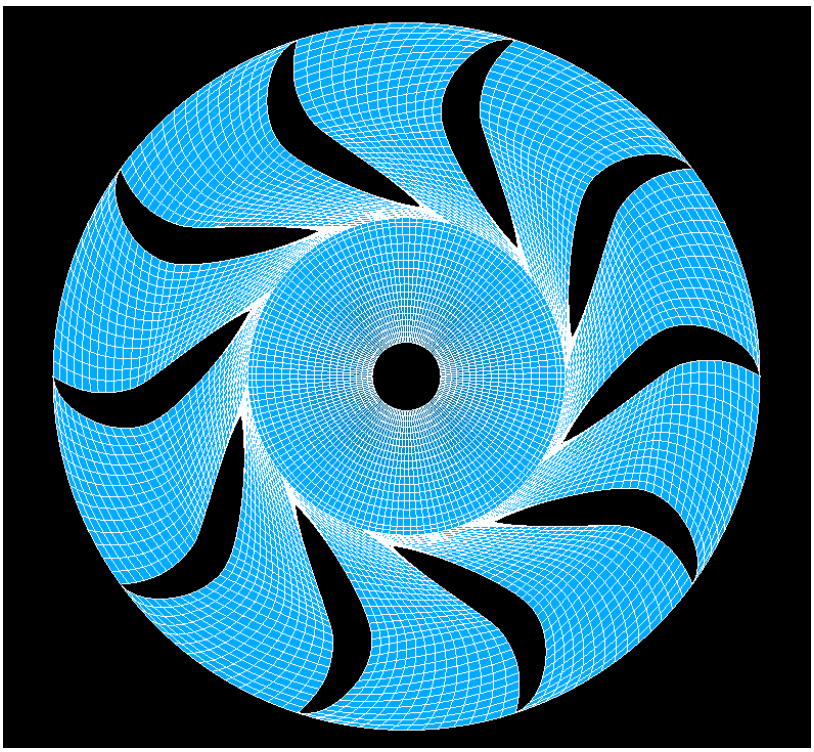


Figure A.41: Final Mesh in 3D

A.4 Guide Vanes

Procedure Summary: Guide Vanes are in principal a combination of aerofoils which are stacked in form of a annular cascade. A 2D aerofoil when extruded to a certain length will form a 3D aerofoil. This feature can be taken advantage of, while making the mesh of guide vane. The aim is to use pressure side of one guide vane and suction side of another guide vane so as to obtain a control surface. Next, as the guide vanes can be operated at various opening angles, its customary to create periodic interfaces at the leading edge and the trailing edge. However when the guide vanes are fully opened periodic interfaces near the guide vane trailing edge will vanish. In authors experience, for parametrisation, this situation would require a special treatment and cannot be scripted via the strategy explained above.
The following script can be divided in two parts. First part would create a domain for various opening angles. Domain here means the sorted geometry on which blocks can be made.
The second part will introduce the method for blocking, although the same blocking technique can be used for creating domains of different guide vane angles, in user's opinion, it would also cause compromise in the mesh quality.
Note that in given geometry the guide vane angles are not correct, this needs to be repaired, but this is not a problem as will be noticed in later sections
Also the guide vane opening angle in the given geometry is 14.4 degree and full opening angle is 15.5 degrees.

- By now reader should be familiarised with most of commands in text user interface(TUI) of Salome. So now there will be minimal focus on explaining commands. Pictures are presented for reference.
- Following lines initialise the geometry module.

```
import sys
import salome
salome.salome_init()
theStudy = salome.myStudy
import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'Working Directory')

import GEOM
from salome.geom import geomBuilder
```

```
import math
import SALOMEDS

geompy = geomBuilder.New(theStudy)

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

Guide_Vane_file = geompy.ImportSTEP("FileName with path.stp")
```

- As can be seen in the Figure the various edges of guide vanes can be picked up. The names are given for easy identification.

```
[Suction2,Leading,GV_Center1,Pressure_Side1,
Inner_radius,Suction3,Suction1,
Path_for_extrusion,Outer_radius,
Pressure_Suction,Pressure_Side2,GV_Center2] =
geompy.SubShapes
(Guide_Vane_STEPFILE_stp_1,
[596, 566, 555, 541, 39, 557, 568, 603, 64, 624, 564, 552])
```

- These edges are added into a wire (combination of consecutive edges) so that they form a single object. This single object can be rotated about a axis passing through its own centre of mass to obtain the domain for different guide vane opening angles.

```
GV_Profil = geompy.MakeWire([Suction2, Leading,
Pressure_Side1, Suction3,
Suction1, Pressure_Suction
, Pressure_Side2], 1e-07)

Central_Circle = geompy.MakeWire([GV_Center1, GV_Center2], 1e-07)

geomObj_1 = geompy.MakeCDG(Central_Circle)

Center_of_mass_for_rotation = geompy.MakeCDG(Central_Circle)
```

```
To_rotate_we_need_another_vertex =
geompy.MakeTranslation(Center_of_mass_for_rotation, 0, 0, 10)

# A vector is created between the two vertex, about which Guide Vane is rotated

Axis_for_rotating_guide_vane =
geompy.MakeVector
(Center_of_mass_for_rotation, To_rotate_we_need_another_vertex)

#outer_circle2 = geompy.MakeMirrorByPoint(Outer_Radius, O)

#geompy.addToStudy(outer_circle2, "outer_circle2")
```

- At this point in script rotation can be made to change the guide vane opening angle. Here the angle to be specified is difference between the required guide vane angle and present angle of geometry which is 14.4 degrees.

```
GV_Profil_Rotated = geompy.MakeRotation
(GV_Profil, Axis_for_rotating_guide_vane, 2*math.pi/180.0)

geompy.addToStudy(GV_Profil_Rotated, "GV_Profil_Rotated")
```

- In author's experience, Whenever a object is exploded into sub components, the method (ID's) or even the names, to address the sub components changes if the main object is at a different position/orientation than original. To make this script more robust following method may be useful.

```
Edges_Main_GV_Profil =
geompy.ExtractShapes(GV_Profil_Rotated, geompy.ShapeType["EDGE"], True)

# The length of the various edges is
# fortunately different and in a specific order.

m = len(Edges_Main_GV_Profil)
LE = []
UL = []                                  # List of lenghts of the edges
```

```
for i in xrange(0, m):
BP = geompy.BasicProperties(Edges_Main_GV_Profil[i])
l = BP[0]
LE.append(l)
LE.sort(reverse=True)

for i in xrange(0, m):

for j in xrange(0, m):

BP2 = geompy.BasicProperties(Edges_Main_GV_Profil[j])

l2  = BP2[0]

if(l2 == LE[i]):
flag = 1
if(flag ==1):
UL.append(Edges_Main_GV_Profil[j])


Pressure_long   = UL[0]
Suction_long    = UL[1]
Pressure_short  = UL[2]
Suction_short   = UL[3]
Pressure_tiny   = UL[4]
Leading_edge    = UL[5]
Trailing_edge   = UL[6]

#geompy.addToStudy(Pressure_long, "Pressure_long")
geompy.addToStudy(Suction_long , "Suction_long")
#geompy.addToStudy(Pressure_short, "Pressure_short")
geompy.addToStudy(Suction_short, "Suction_short")
geompy.addToStudy(Pressure_tiny, "Pressure_tiny")
#geompy.addToStudy(Leading_edge, "Leading_edge")
geompy.addToStudy(Trailing_edge, "Trailing_edge")
```

- Next lines are used for creating periodic interfaces.

```
# Pick up a point on trailing edge
```

```
pt_trailing = geompy.MakeVertexOnLinesIntersection(Pressure_long, Trailing_edge)

# Connect this point to center & rotate it by 60 degrees !!
# O means origin.

line1_trailing_to_center =
geompy.MakeEdge(O, pt_trailing)

line2_trailing_to_center =
geompy.MakeRotation(line1_trailing_to_center, OZ,
-60math.pi/180.0)

############################### Trailing connections to center
#geompy.addToStudy(line1_trailing_to_center, "line1_trailing_to_center")
#geompy.addToStudy(line2_trailing_to_center, "line2_trailing_to_center")


# Rotate the edges of Guide Vane to form another Guide vane with same angle

Pressure_long_Rotated  = geompy.MakeRotation(Pressure_long, OZ,  -60*math.pi/180.0)
Suction_long_Rotated   = geompy.MakeRotation(Suction_long, OZ,   -60*math.pi/180.0)
Pressure_short_Rotated = geompy.MakeRotation(Pressure_short, OZ, -60*math.pi/180.0)
Suction_short_Rotated  = geompy.MakeRotation(Suction_short, OZ,  -60*math.pi/180.0)
Pressure_tiny_Rotated  = geompy.MakeRotation(Pressure_tiny, OZ,  -60*math.pi/180.0)
Leading_edge_Rotated   = geompy.MakeRotation(Leading_edge, OZ,   -60*math.pi/180.0)
Trailing_edge_Rotated  = geompy.MakeRotation(Trailing_edge, OZ,  -60*math.pi/180.0)

geompy.addToStudy(Pressure_long_Rotated,  "Pressure_long_Rotated")
#geompy.addToStudy(Suction_long_Rotated,   "Suction_long_Rotated")
geompy.addToStudy(Pressure_short_Rotated, "Pressure_short_Rotated")
#geompy.addToStudy(Suction_short_Rotated,  "Suction_short_Rotated")
#geompy.addToStudy(Pressure_tiny_Rotated,  "Pressure_tiny_Rotated")
geompy.addToStudy(Leading_edge_Rotated,   "Leading_edge_Rotated")
#geompy.addToStudy(Trailing_edge_Rotated,  "Trailing_edge_Rotated")

# Now the periodic interfaces on trailing edge has been created.
# Next step is to find out outlet of guide vanes, note that outlet of guide vane
# is addressed by name "inner radius".
```

```
pt_for_blocking =
geompy.MakeVertexOnLinesIntersection(line1_trailing_to_center, Inner_radius)

pt_for_blocking2 =
geompy.MakeVertexOnLinesIntersection(line2_trailing_to_center, Inner_radius)

partition_of_inner_radius =
geompy.MakePartition([Inner_radius],
[pt_for_blocking, pt_for_blocking2], [], [], geompy.ShapeType["EDGE"], 0, [], 0)

# As mentioned before inner radius means outlet of guide vane.
# So now outlet of guide vane is obtained.

#geompy.addToStudy(partition_of_inner_radius, "partition_of_inner_radius")

# The same is done for leading edges.

pt_leading =
geompy.MakeVertexOnLinesIntersection(Suction_short, Leading_edge)
# Point on guide vane
pt_leading2 =
geompy.MakeVertexOnLinesIntersection(Suction_short_Rotated, Leading_edge_Rotated)

# edge from point on guide vane to center

line1_leading_to_center = geompy.MakeEdge(O, pt_leading)

# Centre of mass of this line. This has to be calculated because,
# The line which connects leading edge to centre will not lie on
# Outer radius of guide vanes (which is inlet of guide vane)
# The idea is to scale the two lines about their mid points
# And partition the outer radius (inlet GV) with these two lines.

CDG_line_leading = geompy.MakeCDG(line1_leading_to_center)

Scaled_Leading =
geompy.MakeScaleTransform(line1_leading_to_center, CDG_line_leading, 2)
# Scale tranform this line about its center
```

```
pt_leading_for_blocking =
geompy.MakeVertexOnLinesIntersection(Scaled_Leading, Outer_radius)
# Point on outer radius

Scaled_Leading2 =
geompy.MakeRotation(Scaled_Leading, OZ, -60*math.pi/180.0)

pt_leading_for_blocking2 =
geompy.MakeVertexOnLinesIntersection(Scaled_Leading2, Outer_radius)
# Point on outer radius

line_leading_for_blocking = geompy.MakeEdge(pt_leading_for_blocking, pt_leading)
# Edge from outer radius to guide vane

line_leading_for_blocking2 = geompy.MakeEdge(pt_leading_for_blocking2, pt_leading2)
# Edge from outer radius to guide vane

geompy.addToStudy(line_leading_for_blocking, "line_leading_for_blocking")
geompy.addToStudy(line_leading_for_blocking2, "line_leading_for_blocking2")

partition_of_outer_radius =
geompy.MakePartition([Outer_radius],
[line_leading_for_blocking, line_leading_for_blocking2],
[], [], geompy.ShapeType["EDGE"], 0, [], 0)

#geompy.addToStudy(partition_of_outer_radius, "partition_of_outer_radius")

# Here only required edges are extracted.

[Edge_Outer,aabb_i,ccdd_i] =
geompy.ExtractShapes(partition_of_outer_radius, geompy.ShapeType["EDGE"], True)

[Edge_Inner,xxzz_i,xxyy_i] =
geompy.ExtractShapes(partition_of_inner_radius, geompy.ShapeType["EDGE"], True)

geompy.addToStudy(Edge_Inner, "Edge_Inner")

geompy.addToStudy(Edge_Outer, "Edge_Outer")
```

```
partition_of_trailing_edges =
geompy.MakePartition
([line1_trailing_to_center, line2_trailing_to_center],
[Edge_Inner], [], [], geompy.ShapeType["EDGE"], 0, [], 0)

#geompy.addToStudy(partition_of_trailing_edges, "partition_of_trailing_edges")

[Line_trailing_for_blocking,
Blah_again, Line_trailing_for_blocking2, BowBow] =
geompy.ExtractShapes
(partition_of_trailing_edges, geompy.ShapeType["EDGE"], True)

geompy.addToStudy(Line_trailing_for_blocking, "Line_trailing_for_blocking")

geompy.addToStudy(Line_trailing_for_blocking2, "Line_trailing_for_blocking2")

# Make a compound of all these entities

Domain = geompy.MakeCompound
([Line_trailing_for_blocking,Line_trailing_for_blocking2,
Edge_Inner,Edge_Outer,line_leading_for_blocking ,line_leading_for_blocking2,

Pressure_long_Rotated,Suction_long,Pressure_short_Rotated,
Suction_short, Pressure_tiny ,Leading_edge_Rotated, Trailing_edge])

geompy.addToStudy(Domain, "Domain")
```

- Following above script this image (Figure A.42)should be obtained.

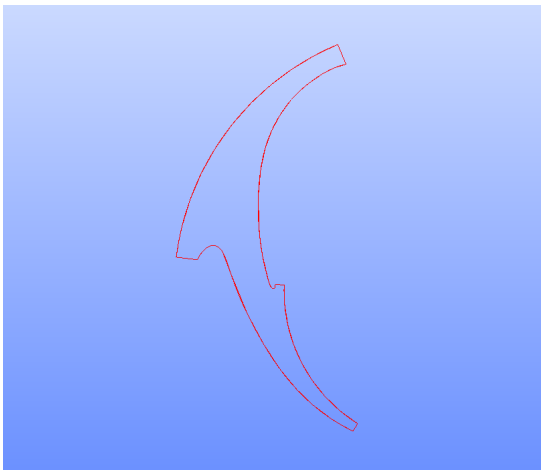


Figure A.42: Geometrical Domain

- The further procedure of creating a boundary layer mesh in near wall region is deliberately skipped as it is very well explained in tutorial supplied with open source library Cfdmsh 3.0 and it can be downloaded from this link Web Link :http://www.vzlu.cz/en/results-transfer/software-r/free-software-download#navigace_bottom.

- The above mentioned script/tutorial uses boundary layer mesh in the near wall region and keeps a unstructured mesh using Netgen algorithm in farfield. Since in this case there is no farfield, a unstructured mesh can be kept inside the domain. For a structured mesh inside the domain, Figure A.43 can be taken as reference.

- Further as till now the blocks are being made in 2D, its required to extrude the domain in 3D by exactly 8mm. The other way would be to create a 2D mesh. Then, the next step would be to extrude the 2D mesh in the mesh module. Following instructions introduces this concept with images and for sample a simple rectangle is used.

- Create a rectangle from New Entity, Generation with 10*10 as dimensions. Create a point with (0, 0, 10) as coordinates. Now a line can be made between this point and origin.

- In the mesh module create a mesh on rectangle with quadrangle mapping as algorithm under 2D tab and wire discretisation under 1D tab, the hypothesis will be Number of segments as 10.

- Create a mesh on line by using wire discretisation as algorithm. In Hypothesis,

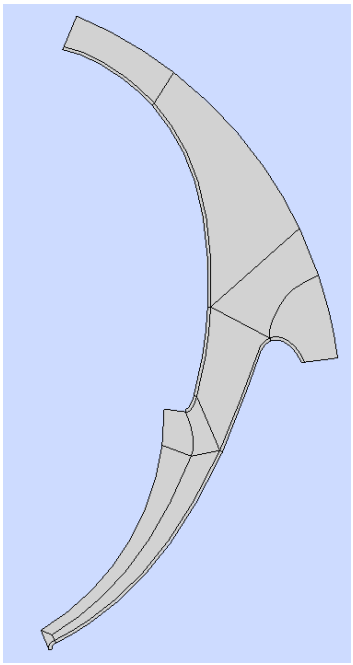


Figure A.43: Blocking strategy

instead of using Number of Segments, in the drop down list, select the option "Distribution with analytic density" and add the formula as shown in A.44

- Next in the main menu on top select Transformation and extrusion along path. In the new dialogue box select the second radio button (extrusion of 2D elements), click the next check box(select whole mesh, submesh or group) and input the mesh of rectangle. In Path select the 1D mesh created on line and then select start node (turquoise colour) as shown in the A.45

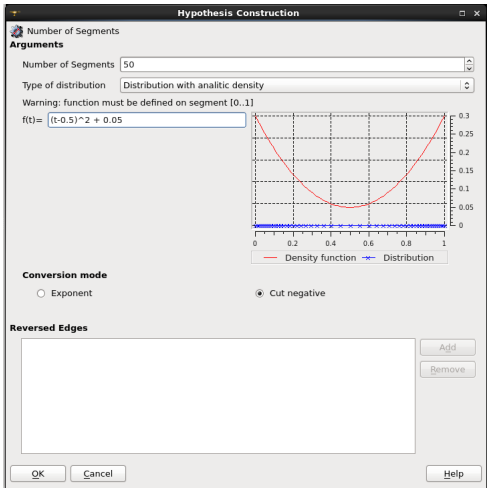


Figure A.44: Hypothesis

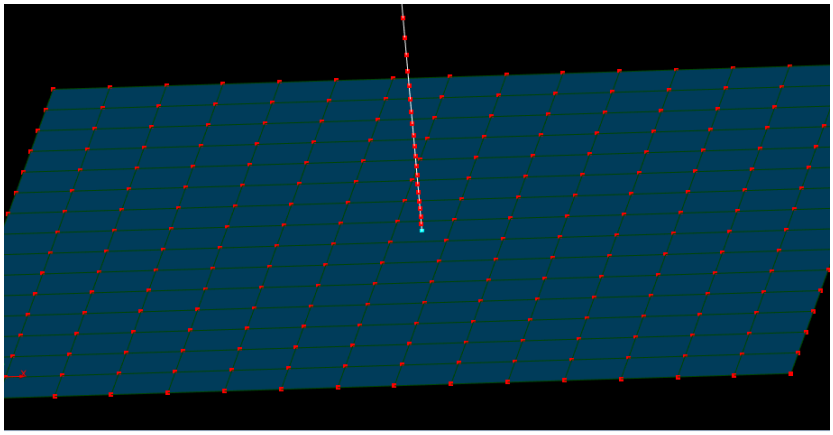


Figure A.45: Extrude Mesh