EDF R&D

FLUID DYNAMICS, POWER GENERATION AND ENVIRONMENT DEPARTMENT
SINGLE PHASE THERMAL-HYDRAULICS GROUP

6, QUAI WATIER
F-78401 CHATOU CEDEX

TEL: 33 1 30 87 75 40
FAX: 33 1 30 87 79 16

JULY 2019

Code_Saturne documentation

SALOME and *Code_Saturne* tutorial: Using SALOME to Create *Code_Saturne* Sliding Interfaces

contact: saturne-support@edf.fr

TABLE OF CONTENTS

Chapter I

Introduction

1 Tutorial Components

This tutorial makes use of:

- The SALOME [1] platform for geometry generation, meshing, and post-processing
- *Code_Saturne* [2, 3] for CFD calculations

To work through this tutorial you will need a computer on which these two software applications are already available or on which you have permission to install them.

2 Tutorial Structure

The purpose of this tutorial is to provide a description of the methodologies developed to intervene in existing meshes and create rotating domains using SALOME and Python scripts.

Starting from tetrahedral meshes, rotating domains are created by

1. Excavating the volume to create space for the sliding interface which we seek to construct
2. In the excavated volume, creating a sliding interface meshed with rectangular elements so that it is compatible with *Code_Saturne*
3. Meshing the excavated volume with tetrahedral elements on each side of the interface
4. Joining the mesh on each side of the interface to the original mesh which remains after excavation

The methodology is illustrated and demonstrated on two test cases. An older, alternative methodology which involved nodes displacement is also described as it may be of interest in future work.

This tutorial is made of two parts:

- Part 1 (Section II) presents the first test case.
- Part 2 (Section III) presents the alternative method, as applied to the first test case.
- Part 3 (Section IV) presents the second test case.

Chapter II

Part 1 - Test Case 1 - Rotating Cylinder

Case: Rotation of a cylinder inside a cube. Using scripts, a cylindrical interface is created inside a cube, which initially contains only tetrahedral elements. To be compatible with *Code_Saturne*, the interface is made of rectangular elements only. To validate the methodology, the model containing the rotating cylinder is tested in *Code_Saturne*.

1 Geometry

The geometry is a cube 400x400x400 center at the origin, as shown in Figure II.1.

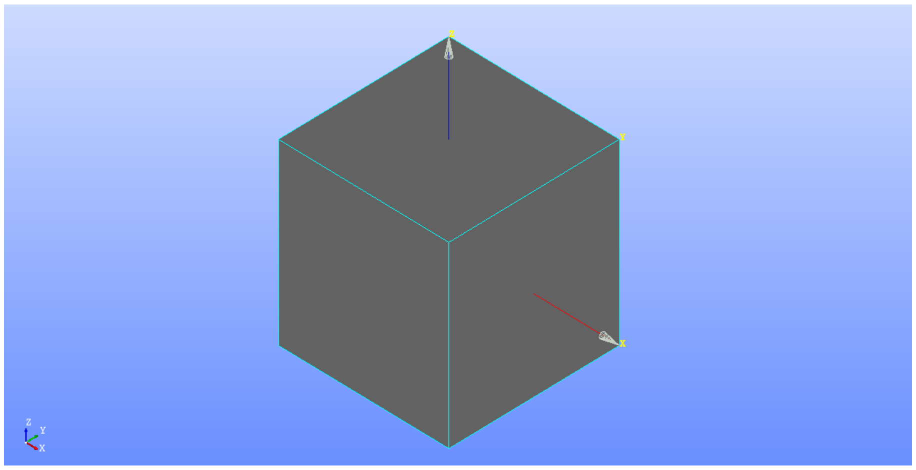

Figure II.1: Geometry

2 Mesh

The cube was meshed with the hypothesis "3D: Automatic Tetrahedralization" with a "Max size" of 30.

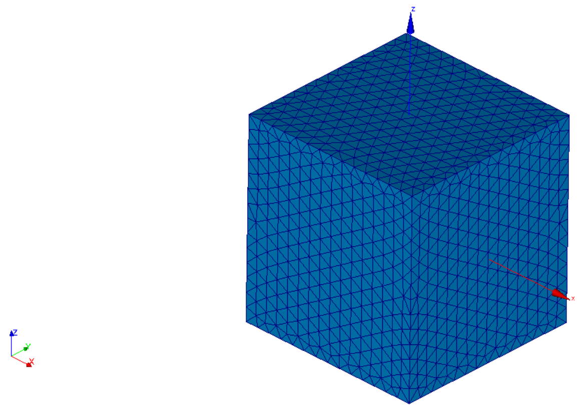

Figure II.2: Mesh

3 Objective

Starting from the existing mesh, we want to insert a cylindrical interface (R in Figure II.3) meshed with rectangular elements only. In order to do that, we will split the mesh into two different regions around the rotating interface as shown in Figure II.3.

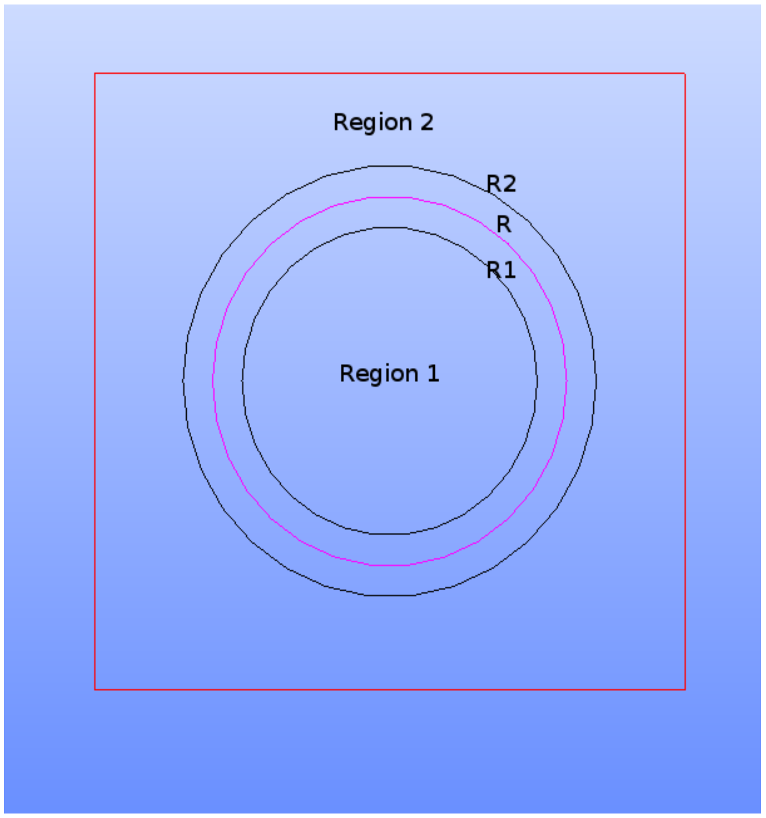

Figure II.3: Schematic of the different regions.

Regions 1 and 2 are made up of the cells from the original mesh. The region between R1 and R will be meshed with tetrahedral elements, except for the interface (R) which will have a surface meshed with rectangular faces. This region will be later merged with Region 1, using the surface at R1 as the criterion for the joining operation. The region between R and R2 will be meshed with tetrahedral elements, except for the interface (R) which will have a surface meshed with rectangular faces. This region will be later merged with Region 2, using the surface at R2 as the criterion for the joining operation.

First, two scripts are used to create the 2 meshes corresponding to regions 1 and 2. Then, the interfaces R1 and R2 are exported in stl and used to create the two volumes between R1 & R and R2 & R in the Geometry module.

4 Script 1: Creation of region 1 from the original mesh.

The script can be found in Appendix A. It contains two steps.

4.1 Step 1: Creating the groups corresponding to Region 2

The first step of the script creates a list of nodes corresponding to region 2. This list is created by selecting all the nodes which have a radius greater than R1 = 100. Then, from this list of nodes a volume group, a group of faces and a group of edges are created, corresponding to the elements in region 2. For example, the volume group created is shown in Fig. II.4.

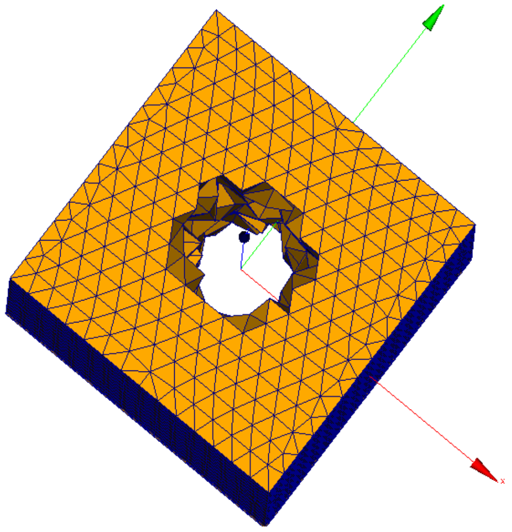

Figure II.4: Volume group corresponding to region 2.

4.2 Step 2: Deleting elements belonging to Region 2 from the original mesh

The next step consists in removing these groups from the original mesh. The volumes, faces, and edges are removed from the mesh by performing a loop on each group. The nodes are deleted using the fonction 'RemoveOrphanNodes()'.

The mesh which is obtained after this step is shown in Figure II.5.

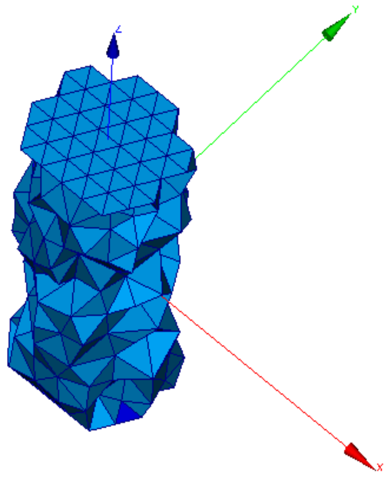

Figure II.5: Mesh corresponding to region 1.

5 Script 2: Creation of region 2 from the original mesh.

The script can be found in Appendix B. The methodology is similar to the one used to create Region 1.

5.1 Step 1: Creating groups corresponding to Region 1.

First, a list of all the nodes corresponding to region 1 is created. This list is created by selecting all the nodes with a radius lower than R2 = 140. Then, from this list of nodes a volume group, a group of faces and a group of edges are created, corresponding to the elements in region 1. For exemple, the volume group created is shown in Figure II.6.

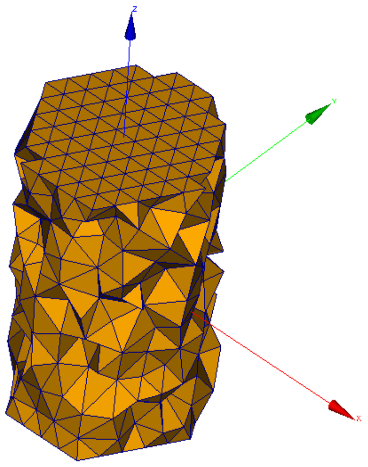

Figure II.6: Volume group corresponding to region 1.

5.2 Step 2: Deleting elements belonging to Region 1 from the original mesh

The next step consists in removing these groups from the original mesh. The volumes, faces, and edges are removed from the mesh by performing a loop on each group. The nodes are deleted using the fonction 'RemoveOrphanNodes()'.

After this step the mesh obtained is shown in Figure II.7.

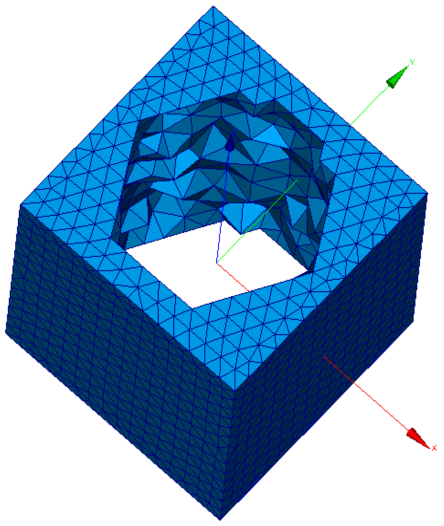

Figure II.7: Mesh corresponding to region 2.

6 Creating the interface region

After executing the 2 scripts, 2 differents mesh are obtained corresponding to regions 1 and 2 (from Figure II.3) as shown in Figure II.8.

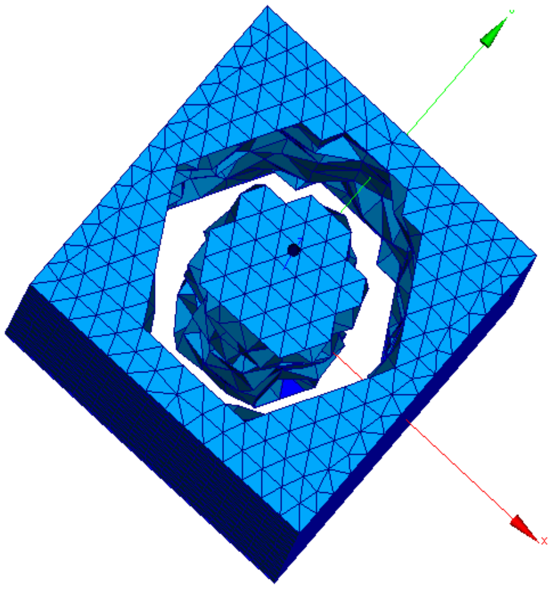

Figure II.8: Meshes for regions 1 and 2.

6.1 Building the sliding interface geometry

The gap between the two meshes show in Figure II.8 is then filled in with the geometry containing the cylindrical interface. The geometry contains two volume, one on each side of the interface. The outer volume will connect on its outside to the Region 2 mesh created above. The inner volume will connect on its inside to the Region 1 mesh created above. The two volumes connect via the cylindrical interface.

At the moment this region is built manually. The first step consists in exporting the 2 joining interfaces corresponding to R1 and R2 in 'stl' format, and to then import them back in the GEOM module of SALOME.

The inner volume is then created as illustrated below. First, the R1 interface is closed with disks in order to create a volume (Figure II.9, Figure II.10).

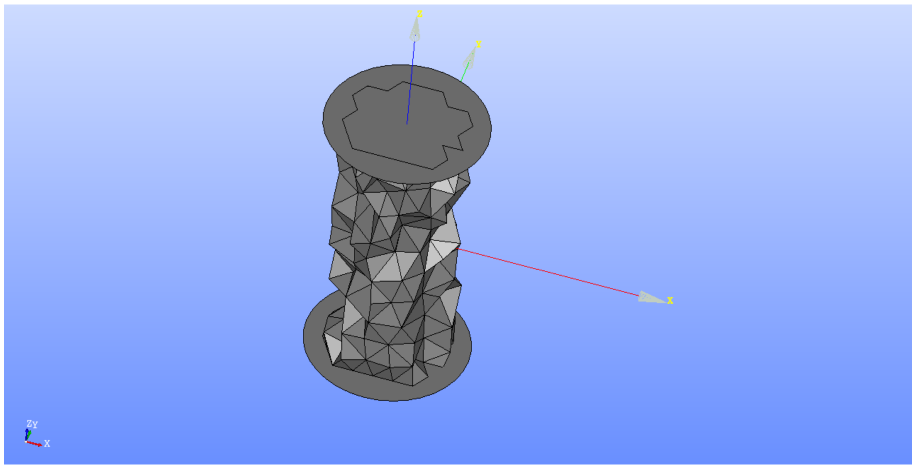

Figure II.9: Creating top and bottom faces from disks and cut operations.

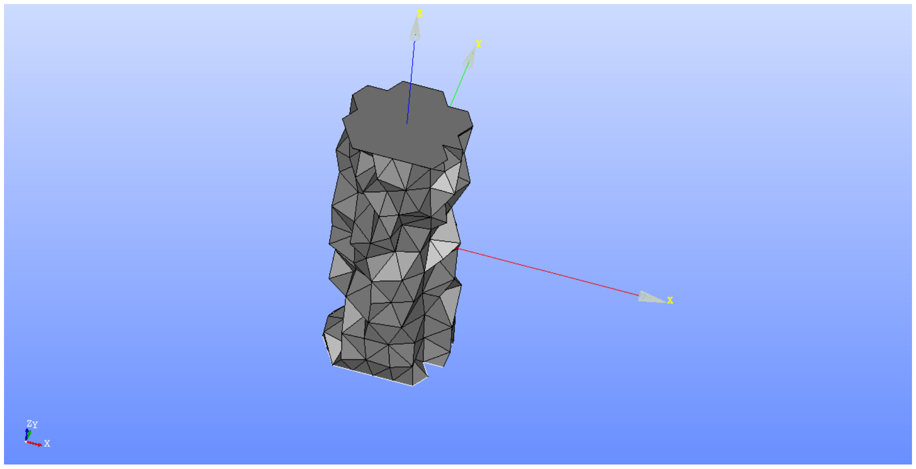

Figure II.10: Volume created from the R1 interface 'stl' surface.

A cylinder is then created in GEOM, with an external radius of R = 120, corresponding to the desired sliding interface. The inner region of the new mesh is then obtained by performing a cut operation between the cylinder and the volume based on the 'stl' built previously (Fig. II.11).

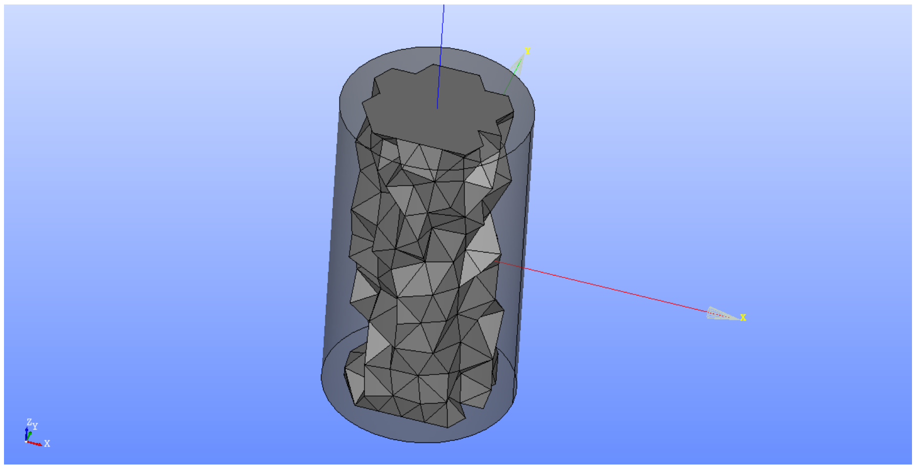

Figure II.11: Cutting operation.

The inner volume is then obtained (Figure II.12).

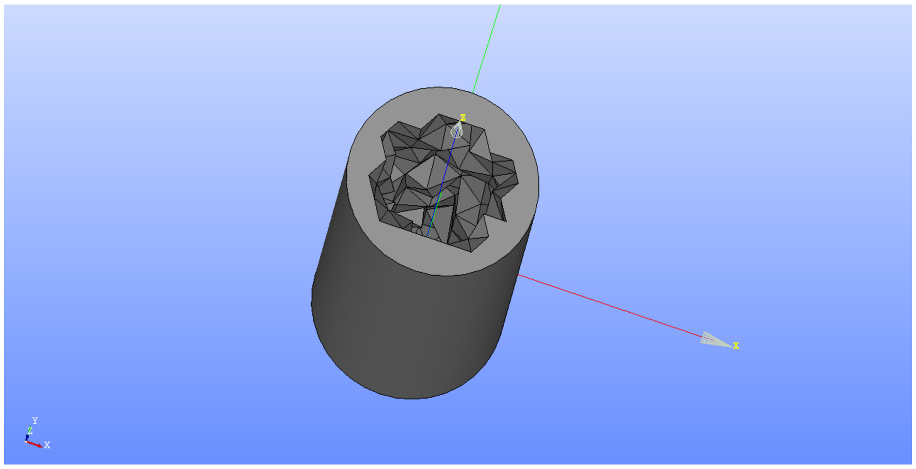

Figure II.12: Interface region 1.

The outer part is built using the same procedure (Fig. II.13).

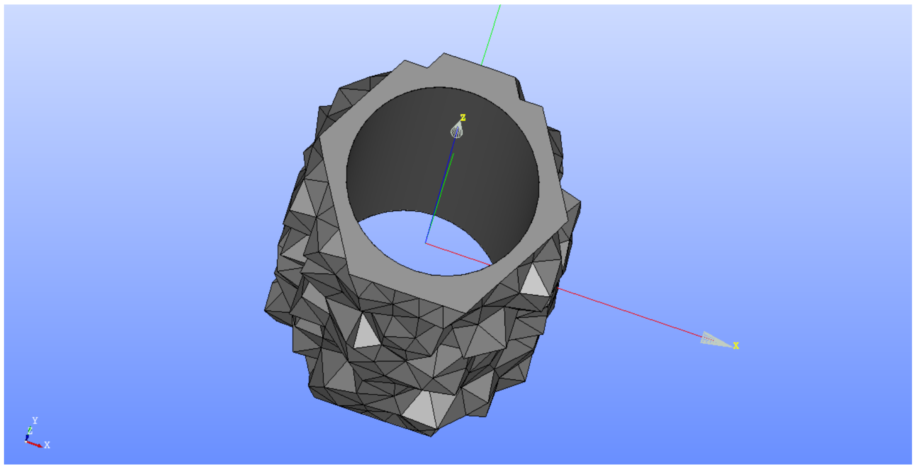

Figure II.13: Interface region 2.

Note: In case of the boolean operation (cut) is not working properly in SALOME, one can use the 'explode' tool instead to obtain the faces of the cylinder. Then, a shell can be built to obtain a solid directly.

6.2 Meshing the interface

Having build the interface regions, they are now meshed using rectangular faces for the the rotating surface (R) and tetrahedral cells everywhere else. The resulting meshes are shown in Figs. II.14 and II.15 below.

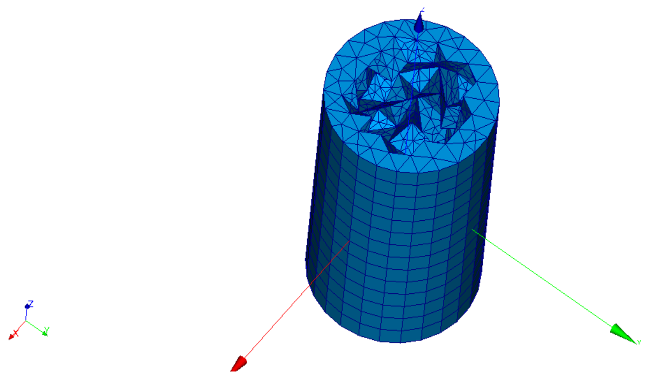

Figure II.14: Mesh of the interface region 1.

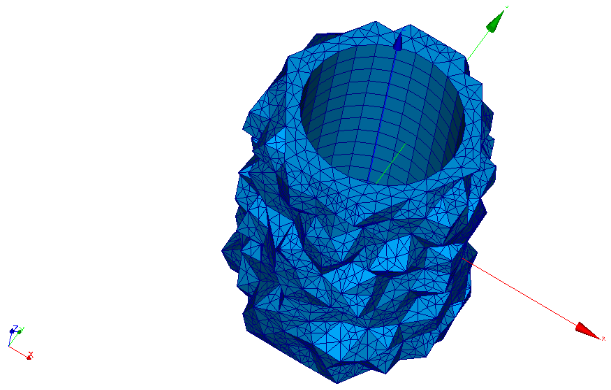

Figure II.15: Mesh of the interface region 2.

Fig. II.16 below displays the 4 four meshes, including regions 1 and 2 from the original mesh and the two meshes created for the interface region.

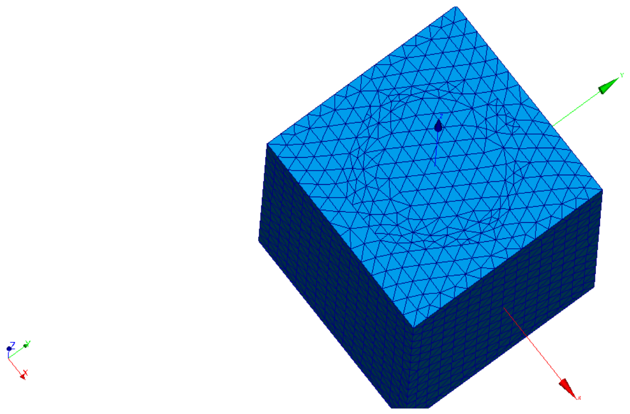

Figure II.16: The 4 meshes together.

7 Creating Groups

The creation of a volume group for the rotating part is required in order to be able to use the turbomachinery module. It is advisable to use the same name for the volume group in region 1 and for the volume group in the interface region 1, and to name the group of faces corresponding to the interface where the rotation will occur with the same name in each mesh.

8 *Code_Saturne* Test Case

A *Code_Saturne* case has been set up in order to verify that the regions and the sliding interface are working correctly.

To create the model, first import the 4 meshes and then define the joining operation as described in Table II.1.

Fraction	Plane	Selection Criteria	Description
0.1	25	Inter_cyl or Inter_INNER	Merging the two meshes which will rotate
0.1	25	Inter_OUTER or Inter_square	Merging the two meshes which will be static

Table II.1: Face joining Parameters.

Activate the turbomachinery module by selecting "Full transient simulation". Then add the volume group defined in SALOME for the rotating part. Set-up a rotation velocity of $0.25 rad.s^{-1}$ and the axis direction for the rotation. Finally, add the rotation interface in the Face joinning field with the parameter of Table II.2.

Fraction	Plane	Selection Criteria	Description
0.1	25	Inter_rotation	Interface of rotation

Table II.2: Face joining Parameters.

The case has been run succesfully for several rotations with a relative velocity of $30 m.s^{-1}$ at the interface.

Chapter III

Part 2 - Test Case 1 - Alternative Method

This method was developed first. After check mesh tests in *Code_Saturne* detected cells crossing each other, it was then abandonned and the method described above was developed instead. Nonetheless, it is summarised here as part of the documentation of the work performed on mesh manipulations.

Similar to the method described above, the alternative one relies on first excavating the volume which will contain the sliding interface mesh. However, unlike the method above, the sliding interface and the mesh immediately connected to it is then created as two pipes connected by the sliding interface (Fig. III.1).

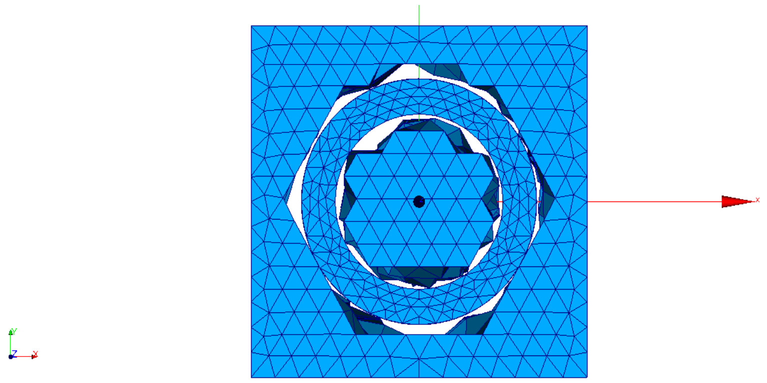

Figure III.1: Meshes for region 1, region 2, and the sliding interface pipe.

The nodes in the original meshes on each side of the the pipe are then moved to the inner and outer radii of the pipe (Figs. III.2, III.3).

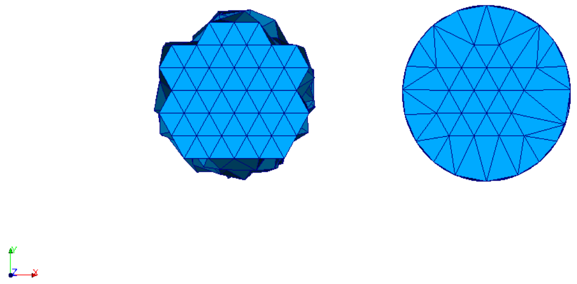

Figure III.2: Region 1 original (left), Region 1 after moving nodes (right).

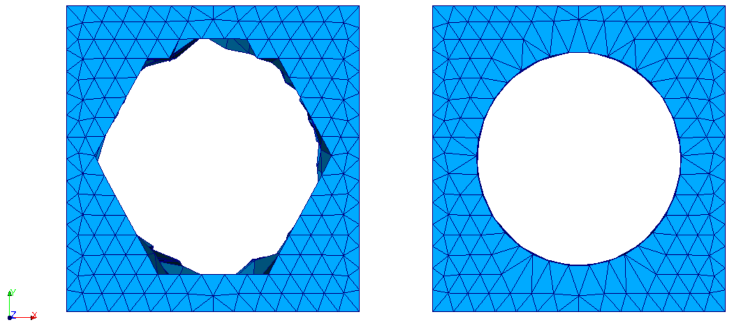
Figure III.3: Region 2 original (left), Region 2 after moving nodes (right).

resulting in the four meshes shown below (Fig. III.4).

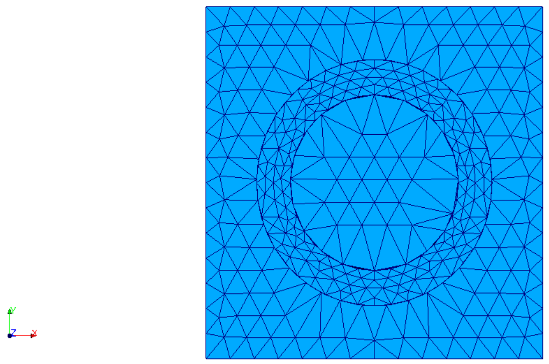
Figure III.4: The 4 meshes together.

Whilst these meshes looked valid, the check mesh in *Code_Saturne* showed that some cells were crossing each other. Zooming on the interfaces before the nodes are moved, one can see that the shape of the cells is such that crossing can result from displacing the nodes (Fig. III.5). In such cases, it is not enough to just change the coordinates of the node along a radius in order to match the pipe's surface, but a more complex, three dimensional displacement should be considered in order not to distort the cells in to shapes which are not valid for CFD.

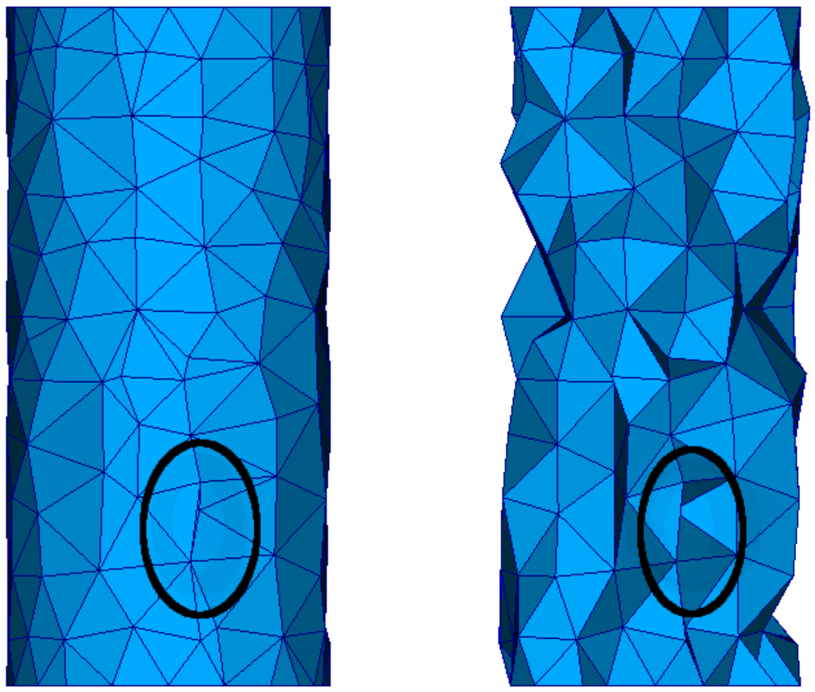

Figure III.5: Cells which can potentially cross each other.

Ensuring that the nodes' displacement alway results in valid cells would have required a much larger algorithmic effort, and one which could probably still fail on cases which are difficult to foresee and prepare for. As the objective of this work was to derive methodologies which Renuda could use in general, industrial situations, this method was, therefore, abandonned for now. It could be interesting to revisit it using ALE type methodologies coupled with constraints to enforce cell validity criteria.

Chapter IV

Part 3 - Test Case 2 - Rotating Cylinder with Disk

Case: Rotation of an impeller inside a cube. This case is more complex than the previous one as it seeks to create a rotating part which ressembles the topology of impellers which can be found in mixing vessels. Therefore, it not only contains a cylindrical shaft as before, but also a disk attached to the bottom of the shaft.

1 Geometry

The starting geometry is the same as for Test Case 1.

2 Mesh

The initial tetrahedral mesh was created in a cube with the hypothesis '3D: Automatic Tetrahedralization' with a 'Max size' of 30 (Fig. IV.1).

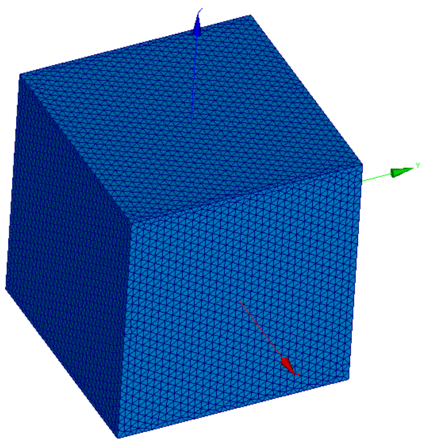

Figure IV.1: Mesh

3 Objective

Inside the cube, we want to insert the rotating interface drawn in purple in Figure IV.2). The interface is meshed with rectangular faces in order to ensure compatibility with *Code_Saturne*.

Similar to the first case, in order to do that we will split the starting mesh in two different region around the rotating interface, as drawn on Fig. IV.2.

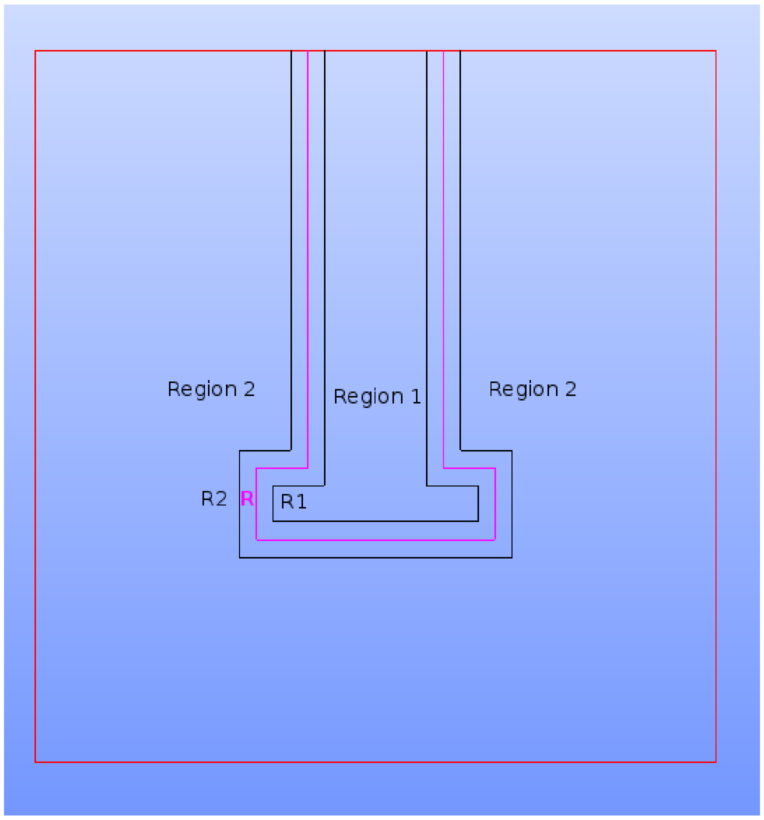

Figure IV.2: Splitting the mesh into two different regions.

Regions 1 and 2 correspond to the regions where the original mesh is kept. The region between R1 and R will be meshed with tetrahedra except at the interface at R, which will have a surface meshed with rectangular elements. This region will then later be merged with Region 1, using the surface at R1 as criterion for the joining operation. The region between R and R2 will be meshed with tetrahedra except at the interface at R, which will have a surface meshed with rectangular elements. This region will then later be merged with Region 2, using the surface at R2 as criterion for the joining operation.

Two scripts are used to create the 2 meshes corresponding to regions 1 and 2. Then, the interfaces R1 and R2 are exported in stl and used to create the two volumes between R1 & R and R2 & R in SALOME's GEOM module.

4 Script 1: Creation of region 1 from the original mesh.

The script can be found in Appendix C.

The steps are the same as test case 1. The mesh obtained from the script corresponding to region 1 is shown in Figure IV.3.

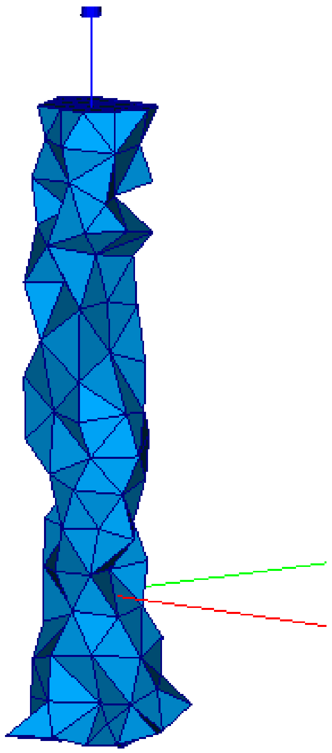

Figure IV.3: Mesh region 1

5 Script 2: Creation of region 2 from the original mesh.

The script can be found in Appendix D.

The steps are the same as test case 1. The mesh obtained from the script corresponding to region 2 is shown in Figure IV.4.

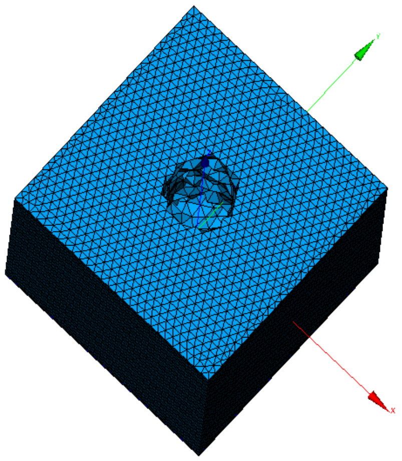

Figure IV.4: Mesh region 1

6 Creating the interface region

6.1 Building the geometry

In the test case, this region was built manually. Similar to test case 1, the first step consists in exporting the 2 joining interfaces corresponding to R1 and R2 to 'stl' surfaces. The two surfaces are then imported back in SALOME's GEOM module.

The two parts of the rotating part, the shaft and impeller are then built as cylinders. For the test, the shaft interface has a radius, R = 80, and the impeller a radius, R = 140.

The 'interface region 1' geometry is presented in Figs. IV.5 and IV.6. Unlike the method used for test case 1, the region has been obtained using the 'shell' and 'solid' tools. The faces are obtained from an 'explode' of the different cylinders. One important point to take into consideration is that the rotating interface will be meshed in hexahedral. To make it easy to build this mesh and to keep the conformity, if desired, between the different surfaces of rotation, the shell is split into quadrants using two orthogonals planes.

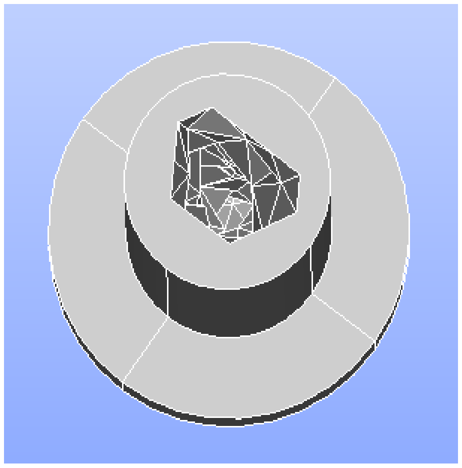

Figure IV.5: Interface region 1

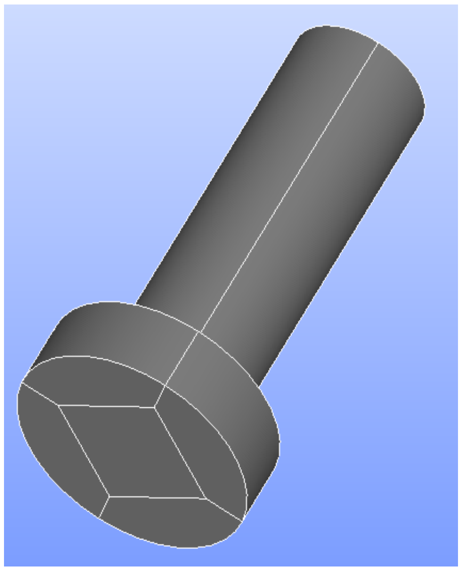

Figure IV.6: Interface region 1

The 'interface region 2' is obtained using the same steps. It is presented in Figs. IV.7 and IV.8 below.

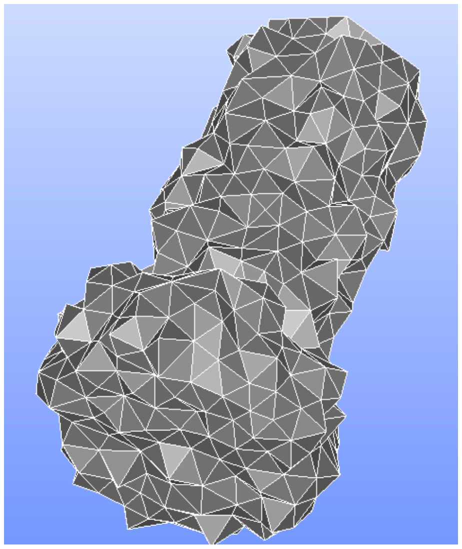

Figure IV.7: Interface region 2

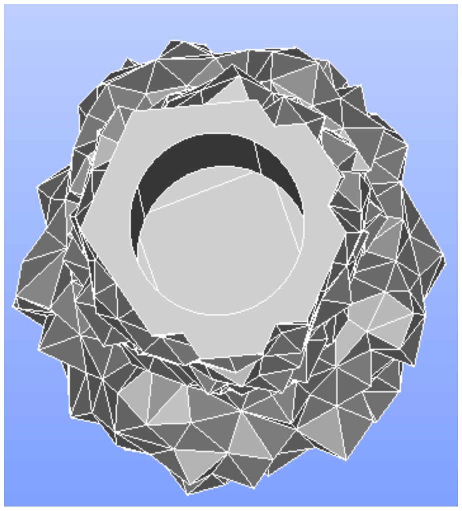

Figure IV.8: Interface region 2

6.2 Meshing the interface

Similar to test case 1, the sliding interface, R, is meshed with rectangular elements and the 'interface region 1' and 'interface region 2' volumes are filled with tetrahedral ones.

The resulting mesh is shown for 'interface region 1' (Fig. IV.9)

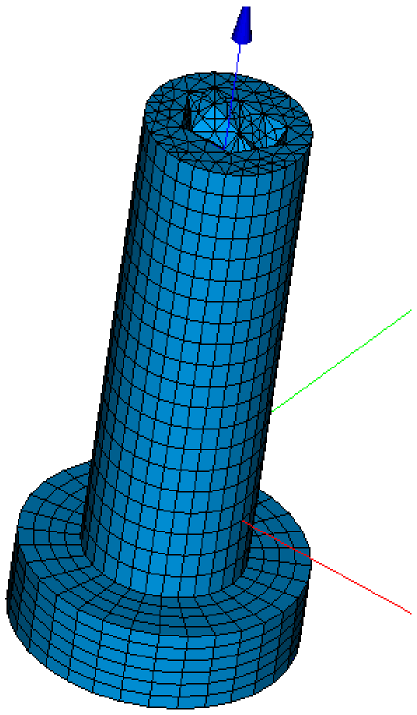

Figure IV.9: Mesh interface region 1

and 'interface region 2' (Fig. IV.10)

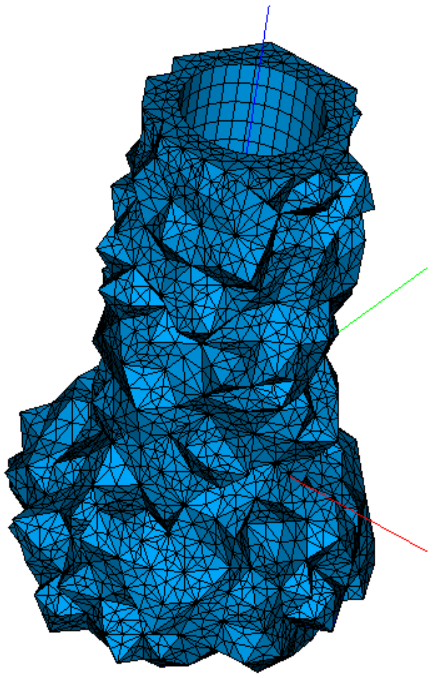

Figure IV.10: Mesh interface region 2

The four meshes of the computational domain are shown in Fig. IV.11: regions 1 and 2 from the original mesh, 'interface region 1' and 'interface region 2'.

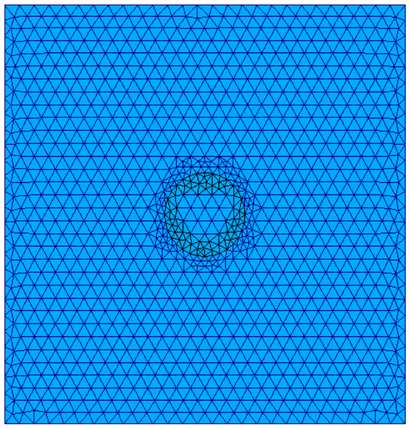

Figure IV.11: Final mesh containing the rotating part and the hexahedral, sliding interface

7 *Code_Saturne* Test Case

A *Code_Saturne* case has been set up in order to verify that the regions and the sliding interface are working correctly.

To create the model, first import the 4 meshes and then define the joining operation as described in Table IV.1.

Fraction	Plane	Selection Criteria	Description
0.1	25	Inter_INNER or Inter_join_rota	Merging the two meshes which will rotate
0.1	25	Inter_OUTER or Inter_join_stat	Merging the two meshes which will be static

Table IV.1: Face joining Parameters.

In the *Code_Saturne* GUI, activate the turbomachinery module by selecting 'Full transient simulation'. Then, add your volume group defined in SALOME for the rotating part. Set-up a rotation velocity of $0.215 rad.s^{-1}$ and the axis direction for the rotation. Finally, add the rotating interface in the Face joinning field with the parameters of Table IV.2.

Fraction	Plane	Selection Criteria	Description
0.1	25	Inter_rotation	Interface of rotation

Table IV.2: Face joining Parameters.

8 Results

The case was set up to run for a little more than one complete rotation in order to verify the relative velocities. Based on the speed of rotation and radii, the relative velocity should be equal to $30.1 m/s$ at the cylindrical interface of the impeller and $17.2 m/s$ at the shaft.

The relative velocity can be visualized in Figure IV.12 to Figure IV.14. Note: In order to visualise the mesh motion at each time step, it is necessary to have activated 'transient connectivity' in the *Code_Saturne* GUI prior to running the case.

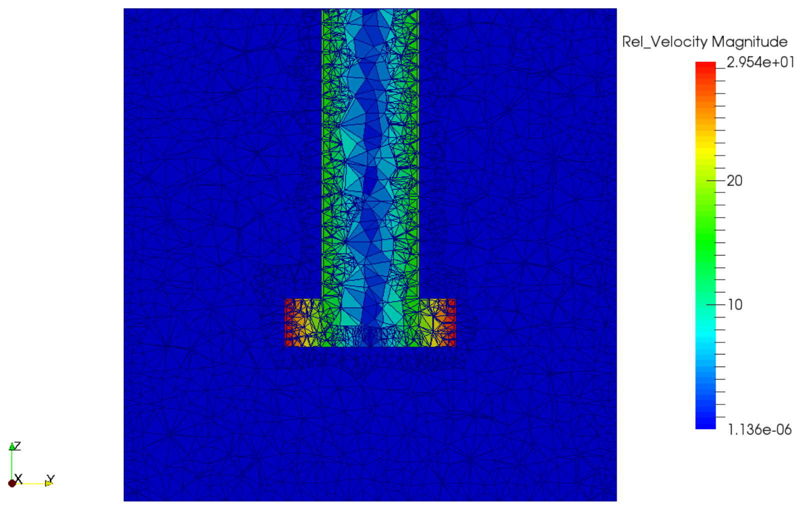

Figure IV.12: Relative velocity in the plane (yz).

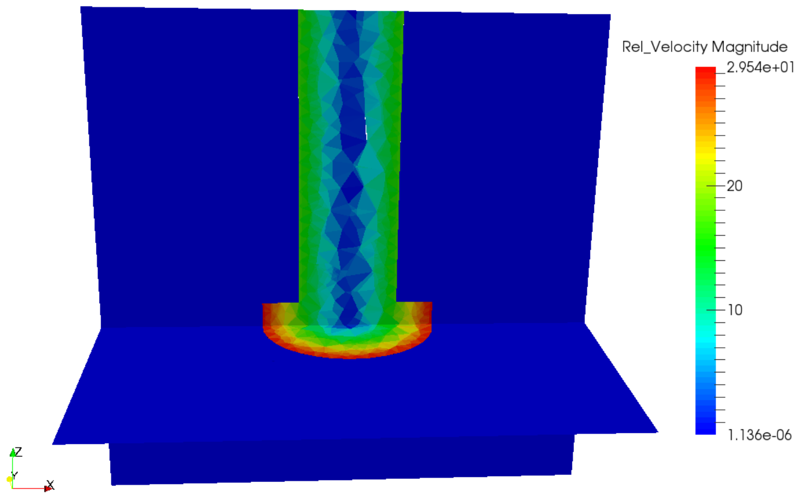

Figure IV.13: Relative velocity in the plane (yz) and (xy)

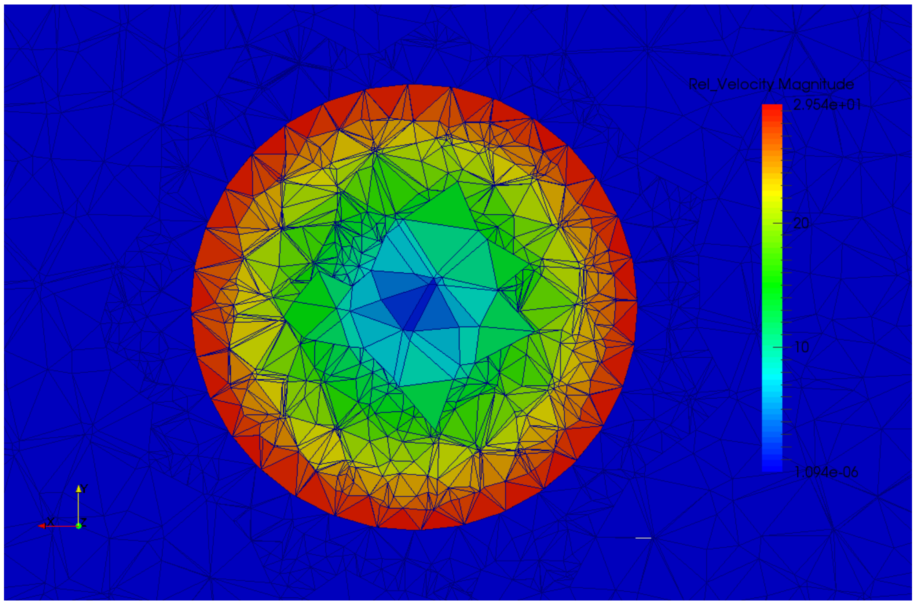

Figure IV.14: Relative velocity in the plane (xy)

The results show that the regions are working as intended.

Chapter V

Appendices

Appendix A

Appendix A – Script SALOME, building region 1 of test case 1

```
# -*- coding: utf-8 -*-

###
### This file is generated automatically by SALOME v7.6.0 with dump
    python functionality
###

import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase1/')

import time
start_time = time.time()



###
### SMESH component
###
from math import sqrt,pi,acos,cos,sin,asin,degrees
import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)
([Mesh_2], status) = smesh.CreateMeshesFromCGNS(r'/home/nicolas/
    EDF_Tutorials_Partnership/Mesh_Manipulations/Mixer_CleanedForEDF/
    TestCase1/Mesh_1.cgns')
```

```
print "Mesh Loaded"


#BC all
nbAdded, Mesh_2, Boundary_faces = Mesh_2.MakeBoundaryElements( SMESH.
   BND_2DFROM3D, 'Boundary_faces_INNER', '', 0, [])

edges = Mesh_2.GetElementsByType(SMESH.EDGE)
faces = Mesh_2.GetElementsByType(SMESH.FACE)
volumes = Mesh_2.GetElementsByType(SMESH.VOLUME)

radius=100


Group_vol_all = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'Vol_total_INNER'
   )
nbAdd=Group_vol_all.Add ( Mesh_2.GetElementsByType(SMESH.VOLUME) )

Group_vol_1 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'grp_vol_1' )
Group_vol_2 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'Vol_node_crit' )

Group_face2=Mesh_2.CreateEmptyGroup( SMESH.FACE, 'face_node_crit' )

Group_edges2=Mesh_2.CreateEmptyGroup( SMESH.EDGE, 'edges2' )


print "size volumes", len(volumes)
print "size faces", len(faces)
print "size edges", len(edges)


IDnoeud2D = []
IDnoeud2D_set = []
aGroupElemIDs = Group_vol_all.GetListOfID()
for i in range(len(aGroupElemIDs)):

        nb_nodes=Mesh_2.GetElemNbNodes(aGroupElemIDs[i])

        if(nb_nodes==4):

                nnode1=Mesh_2.GetElemNode(aGroupElemIDs[i],0)
                nnode2=Mesh_2.GetElemNode(aGroupElemIDs[i],1)
                nnode3=Mesh_2.GetElemNode(aGroupElemIDs[i],2)
                nnode4=Mesh_2.GetElemNode(aGroupElemIDs[i],3)

                x,y,z = Mesh_2.GetNodeXYZ(nnode1)
                rac_nnode1=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode2)
                rac_nnode2=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode3)
                rac_nnode3=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode4)
                rac_nnode4=sqrt(x*x+y*y)
                if(rac_nnode1>radius and rac_nnode2>radius and rac_nnode3
                   >radius and rac_nnode4>radius):

                   nbAdd=Group_vol_1.Add ( [aGroupElemIDs[i]] )
                   for k in range(nb_nodes):
                        IDnoeud2D.append(Mesh_2.GetElemNode(aGroupElemIDs
                           [i],k))

IDnoeud2D_set = list(set(IDnoeud2D))
for i in range(len(IDnoeud2D_set)):
        nodes = Mesh_2.GetNodeInverseElements(IDnoeud2D_set[i])
        for p in range(len(nodes)):
                nbAdd=Group_face2.Add ( [nodes[p]] )
                nbAdd=Group_vol_2.Add ( [nodes[p]] )
                nbAdd=Group_edges2.Add ( [nodes[p]] )

#Vol
GroupElemID_Layer = Group_vol_2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
         for j in range(len(GroupElemID_Layer)):
                 nbRemoved_vol = Mesh_2.RemoveElements([GroupElemID_Layer[
                    j]])
         print"Volume removed :",nbRemoved_vol

#Face
GroupElemID_Layer = Group_face2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
         for j in range(len(GroupElemID_Layer)):
                 nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
         print"Faces removed :",nbRemoved_faces

#Edges
GroupElemID_Layer = Group_edges2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
         for j in range(len(GroupElemID_Layer)):
                 nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
         print"Edges removed :",nbRemoved_faces

#nodes
nbRemoved_nodes = Mesh_2.RemoveOrphanNodes()
print"Nodes removed :",nbRemoved_nodes
nbAdded, Mesh_2, Boundary_faces_after_removed = Mesh_2.
   MakeBoundaryElements( SMESH.BND_2DFROM3D, '
   Boundary_faces_AFTER_removed', '', 0, [])

#Cut
New_Inter = Mesh_2.GetMesh().CutListOfGroups( [
   Boundary_faces_after_removed ], [ Boundary_faces ], 'Inter_INNER' )
smesh.SetName(New_Inter, 'Inter_INNER')

#Remove groupe construction
Mesh_2.RemoveGroup( Group_edges2 )
Mesh_2.RemoveGroup( Group_face2 )
```

```
Mesh_2.RemoveGroup( Group_vol_2 )
Mesh_2.RemoveGroup( Group_vol_1 )
Mesh_2.RemoveGroup( Boundary_faces_after_removed )

print("--- %s seconds ---" % (time.time() - start_time))

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

Appendix B

Appendix B – Script SALOME, building region 2 of test case 1

```
# -*- coding: utf-8 -*-

###
### This file is generated automatically by SALOME v7.6.0 with dump
    python functionality
###

import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase1/')

import time
start_time = time.time()

###
### SMESH component
###
from math import sqrt,pi,acos,cos,sin,asin,degrees
import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)
([Mesh_2], status) = smesh.CreateMeshesFromCGNS(r'/home/nicolas/
    EDF_Tutorials_Partnership/Mesh_Manipulations/Mixer_CleanedForEDF/
    TestCase1/Mesh_1.cgns')


print "Mesh Loaded"
```

```
#BC all
nbAdded, Mesh_2, Boundary_faces = Mesh_2.MakeBoundaryElements( SMESH.
    BND_2DFROM3D, 'Boundary_faces_OUTER', '', 0, [])


edges = Mesh_2.GetElementsByType(SMESH.EDGE)
faces = Mesh_2.GetElementsByType(SMESH.FACE)
volumes = Mesh_2.GetElementsByType(SMESH.VOLUME)

radius=140


Group_vol_all = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'Vol_total_OUTER'
    )
nbAdd=Group_vol_all.Add ( Mesh_2.GetElementsByType(SMESH.VOLUME) )



Group_vol_1 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'grp_vol_1' )
Group_vol_2 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'Vol_node_crit' )

Group_face2=Mesh_2.CreateEmptyGroup( SMESH.FACE, 'face_node_crit' )

Group_edges2=Mesh_2.CreateEmptyGroup( SMESH.EDGE, 'edges2' )


print "size volumes", len(volumes)
print "size faces", len(faces)
print "size edges", len(edges)


IDnoeud2D = []
IDnoeud2D_set = []
aGroupElemIDs = Group_vol_all.GetListOfID()
for i in range(len(aGroupElemIDs)):

        nb_nodes=Mesh_2.GetElemNbNodes(aGroupElemIDs[i])

        if(nb_nodes==4):

                nnode1=Mesh_2.GetElemNode(aGroupElemIDs[i],0)
                nnode2=Mesh_2.GetElemNode(aGroupElemIDs[i],1)
                nnode3=Mesh_2.GetElemNode(aGroupElemIDs[i],2)
                nnode4=Mesh_2.GetElemNode(aGroupElemIDs[i],3)

                x,y,z = Mesh_2.GetNodeXYZ(nnode1)
                rac_nnode1=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode2)
                rac_nnode2=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode3)
                rac_nnode3=sqrt(x*x+y*y)
                x,y,z = Mesh_2.GetNodeXYZ(nnode4)
                rac_nnode4=sqrt(x*x+y*y)
                if(rac_nnode1<radius and rac_nnode2<radius and rac_nnode3
                   <radius and rac_nnode4<radius):

                   nbAdd=Group_vol_1.Add ( [aGroupElemIDs[i]] )
                   for k in range(nb_nodes):
                        IDnoeud2D.append(Mesh_2.GetElemNode(aGroupElemIDs
                           [i],k))

IDnoeud2D_set = list(set(IDnoeud2D))
for i in range(len(IDnoeud2D_set)):
       nodes = Mesh_2.GetNodeInverseElements(IDnoeud2D_set[i])
       for p in range(len(nodes)):
                        nbAdd=Group_face2.Add ( [nodes[p]] )
                        nbAdd=Group_vol_2.Add ( [nodes[p]] )
                        nbAdd=Group_edges2.Add ( [nodes[p]] )
#Vol
GroupElemID_Layer = Group_vol_2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_vol = Mesh_2.RemoveElements([GroupElemID_Layer[
                   j]])
        print"Volume removed :",nbRemoved_vol

#Face
GroupElemID_Layer = Group_face2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                   GroupElemID_Layer[j]])
        print"Faces removed :",nbRemoved_faces

#Edges
GroupElemID_Layer = Group_edges2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                   GroupElemID_Layer[j]])
        print"Edges removed :",nbRemoved_faces

#nodes
nbRemoved_nodes = Mesh_2.RemoveOrphanNodes()
print"Nodes removed :",nbRemoved_nodes

nbAdded, Mesh_2, Boundary_faces_after_removed = Mesh_2.
    MakeBoundaryElements( SMESH.BND_2DFROM3D, '
    Boundary_faces_AFTER_removed', '', 0, [])

#Cut
New_Inter = Mesh_2.GetMesh().CutListOfGroups( [
    Boundary_faces_after_removed ], [ Boundary_faces ], 'Inter_OUTER' )
smesh.SetName(New_Inter, 'Inter_OUTER')


#Remove groupe construction
```

```
Mesh_2.RemoveGroup( Group_edges2 )
Mesh_2.RemoveGroup( Group_face2 )
Mesh_2.RemoveGroup( Group_vol_2 )
Mesh_2.RemoveGroup( Group_vol_1 )
Mesh_2.RemoveGroup( Boundary_faces_after_removed )

print("--- %s seconds ---" % (time.time() - start_time))

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

Appendix C

Appendix C – Script SALOME, building region 1 of test case 2

```
# -*- coding: utf-8 -*-

###
### This file is generated automatically by SALOME v7.6.0 with dump
    python functionality
###

import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/')

import time
start_time = time.time()



###
### SMESH component
###
from math import sqrt,pi,acos,cos,sin,asin,degrees
import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)
([Mesh_2], status) = smesh.CreateMeshesFromCGNS(r'/home/nicolas/
    EDF_Tutorials_Partnership/Mesh_Manipulations/Mixer_CleanedForEDF/
    Mesh_800.cgns')
```

```
print "Mesh Loaded"


#BC all
nbAdded, Mesh_2, Boundary_faces = Mesh_2.MakeBoundaryElements( SMESH.
    BND_2DFROM3D, 'Boundary_faces_INNER', '', 0, [])


edges = Mesh_2.GetElementsByType(SMESH.EDGE)
faces = Mesh_2.GetElementsByType(SMESH.FACE)
volumes = Mesh_2.GetElementsByType(SMESH.VOLUME)

rayon1=60
rayon2=120
tolerance=1e-4
z_limit1=-90
z_limit2=-130

creation_groupe_vol_all = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, '
    Vol_total_INNER' )
nbAdd=creation_groupe_vol_all.Add ( Mesh_2.GetElementsByType(SMESH.VOLUME
    ) )



creation_groupe_vol_1 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'grp_vol_1
    ' )
creation_groupe_vol_2 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, '
    Vol_node_crit' )

creation_groupe_face2=Mesh_2.CreateEmptyGroup( SMESH.FACE, '
    face_node_crit' )

creation_groupe_edges2=Mesh_2.CreateEmptyGroup( SMESH.EDGE, 'edges2' )


print "size volumes", len(volumes)
print "size faces", len(faces)
print "size edges", len(edges)


IDnoeud2D = []
IDnoeud2D_set = []
aGroupElemIDs = creation_groupe_vol_all.GetListOfID()
for i in range(len(aGroupElemIDs)):

        nb_nodes=Mesh_2.GetElemNbNodes(aGroupElemIDs[i])

        if(nb_nodes==4):

                nnode1=Mesh_2.GetElemNode(aGroupElemIDs[i],0)
                nnode2=Mesh_2.GetElemNode(aGroupElemIDs[i],1)
                nnode3=Mesh_2.GetElemNode(aGroupElemIDs[i],2)
                nnode4=Mesh_2.GetElemNode(aGroupElemIDs[i],3)
```

```
                        x,y,z = Mesh_2.GetNodeXYZ(nnode1)
                        rac_nnode1=sqrt(x*x+y*y)
                        z1=z
                        x,y,z = Mesh_2.GetNodeXYZ(nnode2)
                        rac_nnode2=sqrt(x*x+y*y)
                        z2=z
                        x,y,z = Mesh_2.GetNodeXYZ(nnode3)
                        rac_nnode3=sqrt(x*x+y*y)
                        z3=z
                        x,y,z = Mesh_2.GetNodeXYZ(nnode4)
                        rac_nnode4=sqrt(x*x+y*y)
                        z4=z


                        if(rac_nnode1>rayon2 and rac_nnode2>rayon2 and rac_nnode3
                           >rayon2 and rac_nnode4>rayon2):

                                nbAdd=creation_groupe_vol_1.Add ( [aGroupElemIDs[
                                   i]] )
                                for k in range(nb_nodes):
                                        IDnoeud2D.append(Mesh_2.GetElemNode(
                                           aGroupElemIDs[i],k))

                        if(z1<z_limit2 and z2<z_limit2 and z3<z_limit2 and z4<
                           z_limit2):

                                nbAdd=creation_groupe_vol_1.Add ( [aGroupElemIDs[
                                   i]] )
                                for k in range(nb_nodes):
                                        IDnoeud2D.append(Mesh_2.GetElemNode(
                                           aGroupElemIDs[i],k))

                        if(z1>z_limit1 and z2>z_limit1 and z3>z_limit1 and z4>
                           z_limit1 and rac_nnode1>rayon1 and rac_nnode2>rayon1
                           and rac_nnode3>rayon1 and rac_nnode4>rayon1):
                                nbAdd=creation_groupe_vol_1.Add ( [aGroupElemIDs[
                                   i]] )
                                for k in range(nb_nodes):
                                        IDnoeud2D.append(Mesh_2.GetElemNode(
                                           aGroupElemIDs[i],k))


IDnoeud2D_set = list(set(IDnoeud2D))
for i in range(len(IDnoeud2D_set)):
        nodes = Mesh_2.GetNodeInverseElements(IDnoeud2D_set[i])
        for p in range(len(nodes)):
                        nbAdd=creation_groupe_face2.Add ( [nodes[p]] )
                        nbAdd=creation_groupe_vol_2.Add ( [nodes[p]] )
                        nbAdd=creation_groupe_edges2.Add ( [nodes[p]] )

#Vol
GroupElemID_Layer = creation_groupe_vol_2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
         for j in range(len(GroupElemID_Layer)):
                  nbRemoved_vol = Mesh_2.RemoveElements([GroupElemID_Layer[
                j]])
        print"Volume removed :",nbRemoved_vol

#Face
GroupElemID_Layer = creation_groupe_face2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
        print"Faces removed :",nbRemoved_faces

#Edges
GroupElemID_Layer = creation_groupe_edges2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
        print"Edges removed :",nbRemoved_faces

#nodes
nbRemoved_nodes = Mesh_2.RemoveOrphanNodes()
print"Nodes removed :",nbRemoved_nodes

nbAdded, Mesh_2, Boundary_faces_after_removed = Mesh_2.
    MakeBoundaryElements( SMESH.BND_2DFROM3D, '
    Boundary_faces_AFTER_removed', '', 0, [])

#Cut
New_Inter = Mesh_2.GetMesh().CutListOfGroups( [
    Boundary_faces_after_removed ], [ Boundary_faces ], 'Inter_INNER' )
smesh.SetName(New_Inter, 'Inter_INNER')


#Remove groupe construction

Mesh_2.RemoveGroup( creation_groupe_edges2 )
Mesh_2.RemoveGroup( creation_groupe_face2 )
Mesh_2.RemoveGroup( creation_groupe_vol_2 )
Mesh_2.RemoveGroup( creation_groupe_vol_1 )
Mesh_2.RemoveGroup( Boundary_faces_after_removed )

Mesh_2.ExportMED( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/in/in.med', 0, SMESH.
    MED_V2_2, 1, None ,1)
Mesh_2.ExportSTL( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/in/
    Boundary_faces_INNER.stl', 1, Boundary_faces)
Mesh_2.ExportSTL( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/in/Inter_INNER.stl',
    1, New_Inter)

print("--- %s seconds ---" % (time.time() - start_time))

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

Appendix D

Appendix D – Script SALOME, building region 2 of test case 2

```
# -*- coding: utf-8 -*-

###
### This file is generated automatically by SALOME v7.6.0 with dump
    python functionality
###

import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert( 0, r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2')

import time
start_time = time.time()



###
### SMESH component
###
from math import sqrt,pi,acos,cos,sin,asin,degrees
import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)
([Mesh_2], status) = smesh.CreateMeshesFromCGNS(r'/home/nicolas/
    EDF_Tutorials_Partnership/Mesh_Manipulations/Mixer_CleanedForEDF/
    Mesh_800.cgns')
```

```
print "Mesh Loaded"


#BC all
nbAdded, Mesh_2, Boundary_faces = Mesh_2.MakeBoundaryElements( SMESH.
    BND_2DFROM3D, 'Boundary_faces_OUTER', '', 0, [])


edges = Mesh_2.GetElementsByType(SMESH.EDGE)
faces = Mesh_2.GetElementsByType(SMESH.FACE)
volumes = Mesh_2.GetElementsByType(SMESH.VOLUME)

rayon1=100
rayon2=160
tolerance=1e-4
z_limit1=-50
z_limit2=-170

creation_groupe_vol_all = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, '
    Vol_total_OUTER' )
nbAdd=creation_groupe_vol_all.Add ( Mesh_2.GetElementsByType(SMESH.VOLUME
    ) )



creation_groupe_vol_1 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, 'grp_vol_1
    ' )
creation_groupe_vol_2 = Mesh_2.CreateEmptyGroup( SMESH.VOLUME, '
    Vol_node_crit' )

creation_groupe_face2=Mesh_2.CreateEmptyGroup( SMESH.FACE, '
    face_node_crit' )

creation_groupe_edges2=Mesh_2.CreateEmptyGroup( SMESH.EDGE, 'edges2' )


print "size volumes", len(volumes)
print "size faces", len(faces)
print "size edges", len(edges)


IDnoeud2D = []
IDnoeud2D_set = []
aGroupElemIDs = creation_groupe_vol_all.GetListOfID()
for i in range(len(aGroupElemIDs)):

        nb_nodes=Mesh_2.GetElemNbNodes(aGroupElemIDs[i])

        if(nb_nodes==4):

                nnode1=Mesh_2.GetElemNode(aGroupElemIDs[i],0)
                nnode2=Mesh_2.GetElemNode(aGroupElemIDs[i],1)
                nnode3=Mesh_2.GetElemNode(aGroupElemIDs[i],2)
                nnode4=Mesh_2.GetElemNode(aGroupElemIDs[i],3)
```

```
                    x,y,z = Mesh_2.GetNodeXYZ(nnode1)
                    rac_nnode1=sqrt(x*x+y*y)
                    z1=z
                    x,y,z = Mesh_2.GetNodeXYZ(nnode2)
                    rac_nnode2=sqrt(x*x+y*y)
                    z2=z
                    x,y,z = Mesh_2.GetNodeXYZ(nnode3)
                    rac_nnode3=sqrt(x*x+y*y)
                    z3=z
                    x,y,z = Mesh_2.GetNodeXYZ(nnode4)
                    rac_nnode4=sqrt(x*x+y*y)
                    z4=z


                    if(rac_nnode1<rayon1 and rac_nnode2<rayon1 and rac_nnode3
                        <rayon1 and rac_nnode4<rayon1 and z1>z_limit2 and z2>
                        z_limit2 and z3>z_limit2 and z4>z_limit2):

                            nbAdd=creation_groupe_vol_1.Add ( [aGroupElemIDs[
                                i]] )
                            for k in range(nb_nodes):
                                    IDnoeud2D.append(Mesh_2.GetElemNode(
                                        aGroupElemIDs[i],k))


                    if(z1<z_limit1 and z2<z_limit1 and z3<z_limit1 and z4<
                        z_limit1 and rac_nnode1<rayon2 and rac_nnode2<rayon2
                        and rac_nnode3<rayon2 and rac_nnode4<rayon2 and z1>
                        z_limit2 and z2>z_limit2 and z3>z_limit2 and z4>
                        z_limit2):
                            nbAdd=creation_groupe_vol_1.Add ( [aGroupElemIDs[
                                i]] )
                            for k in range(nb_nodes):
                                    IDnoeud2D.append(Mesh_2.GetElemNode(
                                        aGroupElemIDs[i],k))




IDnoeud2D_set = list(set(IDnoeud2D))
for i in range(len(IDnoeud2D_set)):
        nodes = Mesh_2.GetNodeInverseElements(IDnoeud2D_set[i])
        for p in range(len(nodes)):
                        nbAdd=creation_groupe_face2.Add ( [nodes[p]] )
                        nbAdd=creation_groupe_vol_2.Add ( [nodes[p]] )
                        nbAdd=creation_groupe_edges2.Add ( [nodes[p]] )

#Vol
GroupElemID_Layer = creation_groupe_vol_2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
         for j in range(len(GroupElemID_Layer)):
                 nbRemoved_vol = Mesh_2.RemoveElements([GroupElemID_Layer[
                     j]])
         print"Volume removed :",nbRemoved_vol
```

```
#Face
GroupElemID_Layer = creation_groupe_face2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
        print"Faces removed :",nbRemoved_faces

#Edges
GroupElemID_Layer = creation_groupe_edges2.GetListOfID()
if(len(GroupElemID_Layer)!=0):
        for j in range(len(GroupElemID_Layer)):
                nbRemoved_faces = Mesh_2.RemoveElements([
                    GroupElemID_Layer[j]])
        print"Edges removed :",nbRemoved_faces

#nodes
nbRemoved_nodes = Mesh_2.RemoveOrphanNodes()
print"Nodes removed :",nbRemoved_nodes

nbAdded, Mesh_2, Boundary_faces_after_removed = Mesh_2.
    MakeBoundaryElements( SMESH.BND_2DFROM3D, '
    Boundary_faces_AFTER_removed', '', 0, [])

#Cut
New_Inter = Mesh_2.GetMesh().CutListOfGroups( [
    Boundary_faces_after_removed ], [ Boundary_faces ], 'Inter_OUTER' )
smesh.SetName(New_Inter, 'Inter_OUTER')

#Remove groupe construction

Mesh_2.RemoveGroup( creation_groupe_edges2 )
Mesh_2.RemoveGroup( creation_groupe_face2 )
Mesh_2.RemoveGroup( creation_groupe_vol_2 )
Mesh_2.RemoveGroup( creation_groupe_vol_1 )
Mesh_2.RemoveGroup( Boundary_faces_after_removed )

Mesh_2.ExportMED( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/out/out.med', 0,
    SMESH.MED_V2_2, 1, None ,1)
Mesh_2.ExportSTL( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/out/
    Boundary_faces_OUTER.stl', 1, Boundary_faces)
Mesh_2.ExportSTL( r'/home/nicolas/EDF_Tutorials_Partnership/
    Mesh_Manipulations/Mixer_CleanedForEDF/TestCase2/out/Inter_OUTER.stl',
     1, New_Inter)


print("--- %s seconds ---" % (time.time() - start_time))

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

Chapter VI

References

1 References

[1] www.salome-platform.org

[2] F. Archambeau, N. Méchitoua, M. Sakiz,
Code_Saturne: a Finite Volume Code for the Computation of Turbulent Incompressible Flows - Industrial Applications,
International Journal on Finite Volumes, Vol. 1, 2004.

[3] www.code-saturne.org