SuperMarine 0.2b Instructions

G. St-Pierre Lemieux

Abstract

Creating a good meshing is a complex task which needs a lot of time. It is often an important wall for people wishing to use CFD to understand the physics inside a reactor or a mixing device. The "easy" solution often implies a tetrahedral mesh, which isn't recommended for CFD applications. SuperMarine is a geometry generator for OpenFOAM that can be used for stirred tanks commonly used in chemical and biochemical engineering. It aims to give an easy solution for academic research and students that which to use OpenFOAM on common and simple geometry. The scripts create a blockMesh file ready to be used within a OpenFOAM case.

Key words: Bioreactor, Reactor, Python, Mesh, Geometry

1 Introduction

SuperMarine was first developed to build a marine impeller, but can also be used to create a wide variety of impeller. The present version of the code is sufficient to create accurately a basic stirred tank design with its impeller, but can be tweaked in order to have more sophisticated shapes. Since it is a python script, the user can change the code has he wishes and adapt it to his case.

The method used in the script allow the creation of extruded shape with a minimum of an impact on the mesh quality. The blockMesh code created is simple and use a minimum of the exotic capability offered, making the mesh generation a highly predictable process. The script quickly leads to a good mesh passing most, if not all, checkMesh tests.

Also the mesh generated is fully hexahedral, which is the best way to cut space for CFD application.

2 Base Structure

The basic geometry of the tank is composed of a cylinder of two or more layers consisting of two or more sections split in quadrants. The center section is hexagonal, while the section adopt a toric configuration. This configuration avoids the stretching of the cells around the cylinder. This also has the consequence of forcing the outer section to be composed of a number of quadrants which is a multiple of four.

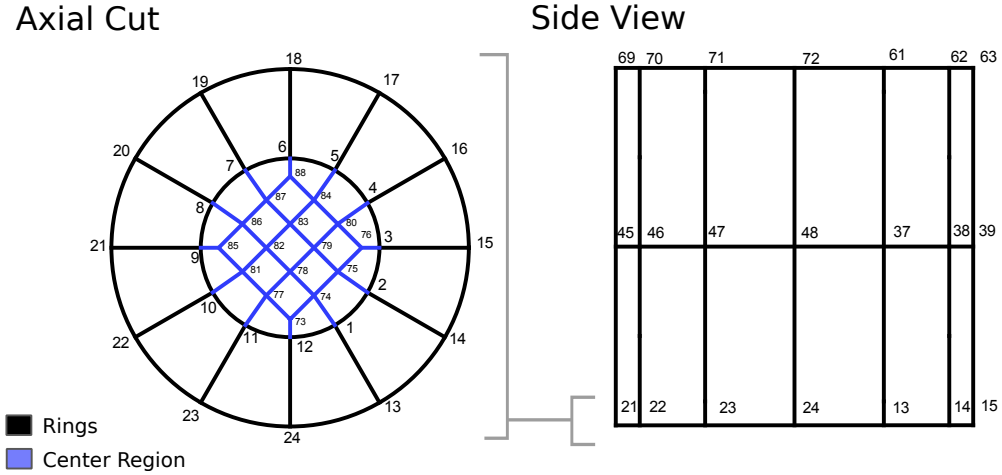

Fig 1. Vertex Ordering

The outer region is built first, counterclockwise, from the center to the perimeter. Then each layer is added, from bottom to top. The center section is built at the end to fit the cylinder configuration, therefore the index starts with the last value of the outer ring setup. The vertex order is shown in the figure 4. You can use *paraFoam* has a tool to visualize how the geometry is built. Call the application in the case directory with the argument "–block".

The hexahedrons are built in the same order. Using each vertex in the order specified in the blockMesh instructions. For sake of simplicity, all the rings hexahedrons have the same meshing instructions and the center hexahedrons meshing are fixed by the rings setup. This often results in a fine mesh in the center of the geometry, you can try to change this by using ... instructions like specified in the blockMesh instructions. But I do not recommend you to use this function.

3 Settings

The script is controlled using 8 global variables which are used has set up parameters. Just by changing those variable a wide range of geometries can be generated. The impeller can be described having the vertical extrusion of a polar grid cut-out. Therefore, the idea is to carve out sections of the polar grid to create the impeller. After making the cut, each vertex layer can be rotated to generate curves. This process is simple and can generate good approximation of many shapes. If more precision is needed, the vertex can be slightly moved without changing the grid to much. The later step is considered has an advanced manipulation and will be described later in the document.

*3.1 The **DIVI** global variable*

The **DIVI** global variable sets the number of divisions in the meshing for each ring hexahedrons. This option also controls the center hexahedrong meshing, which has to represent the outer ring cut-out.

- For the rings:
  DIVI = [Angular , Radial, Z-Direction]
- For the center:
  DIVI = [X and Y-Directions , No-Effect, Z-Direction]

Finer control of the layering and meshing can be achieved in blockMesh. To have a simpler experience only use a uniform meshing rule. Instead, to make the grid finer or to create a gradient, simply append smaller layers or rings. See the **HLAY** and **RQUAD** sections for more information on this technique.

*3.2 The **RQUAD** global variable*

The **RQUAD** is an array that contain at least two real number. The first define the center radius and the other the radius of each section. Each ring mesh has the same number of meshing. So, if the **DIVI** setting is $[10, 10, 10]$ each ring will contain 10 cells in both directions, whatever is the size of your ring. Using this technique, a gradient can be generated with ease. A quick example: each layer can be appended with a 1.2 factor simply by entering $[0.02, 0.024, 0.03]$ for a cylinder of a 3 cm radius. Also, since **NUMPY** is loaded, you can use it to create smooth scaling or to fill the array with ease.

Few examples:

- 1 dm in 10 even meshed ring (using NUMPY):

```
RQUAD = np.arange(0.1,1.1,0.1)*0.1
```

- 1 dm in 10 even meshed ring (manually):

```
RQUAD = [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1]
```

- 5 mm 1.2 factor ring:

```
RQUAD = 1.2**np.arange(0,9.83,1)-0.1598
```

*3.3 The **HLAY** global variable*

The **HLAY** global variable is an array of at least one real number. They define the height of each layer. Like **RQUAD** each layers have to same number of cells. You can also split the space such as it creates a scaling factor. Here, again, you can use **NUMPY** to create the array.

Few examples:

- Two layers 1 cm of height with a cut at 0.2cm). This creates a finer bottom.

```
HLAY = [0.002, 0.01]
```

- Three layer (1 dm), with a finer section in the center:

```
HLAY = [0.04, 0.06, 0.1]
```

*3.4 The **SHAFT** global variable*

The **SHAFT** global variable is an array of the same size has HLAY containing boolean. If True, the central section will not be meshed, therefore producing the shaft of the impeller.

Trick: Sometime, just extending the shaft will make the meshing much smaller since the center has a lot of cells.

Table 1
Quick Instructions

DIVI	Angular, Radial and Vertical divisions per block
RQUAD	Radius of each CADran This parameter must be a list of at least two elements, The first being the center hole/square section.
NSEC	The Number of SECtors to create Must be a multiple of 4 (12,24 and 36 are useful multiple of 4!)
HLAY	Height of each LAYers This parameter must be a list of at least one element
SHAFT	Section on which the SHAFT exists This parameter must be a list which as the same number of elements of HLAY
IMPELLERCUT	Where to CUT for the IMPELLER. This parameter must be a NSEC by NCAR by NHLAY 3d matrix. A 1 in a region means to cut that region for the impeller.
SQRRATIO	The RATIO for the distance between the center SQuaRe region and the outer cylinder. Must be larger than 0 and smaller than 1.

*3.5 The **IMPELLERCUT** global variable*

The **IMPELLERCUT** global variable is the trickier. It's a tridimensional boolean matrix which represents where the cylinder has to be cut. Each cells of the matrix represent a hexahedron in the geometry, if the cells are true, the corresponding hexahedron is carved out.

The hexahedron are indexed counterclockwise, from the center to the perimeter and from the base to the top.

Missing Image Index

*3.6 The **SQRRATIO** global variable*

The **SQRRATIO** is a real number larger than 0 and smaller than 1. This number represents the ratio of the inner tetrahedrons with the connecting section.

If you wish to have more controls, you probably will, see the "Advanced Tweaking" section.

4 Simples Exemples

4.1 Three lobbed marine

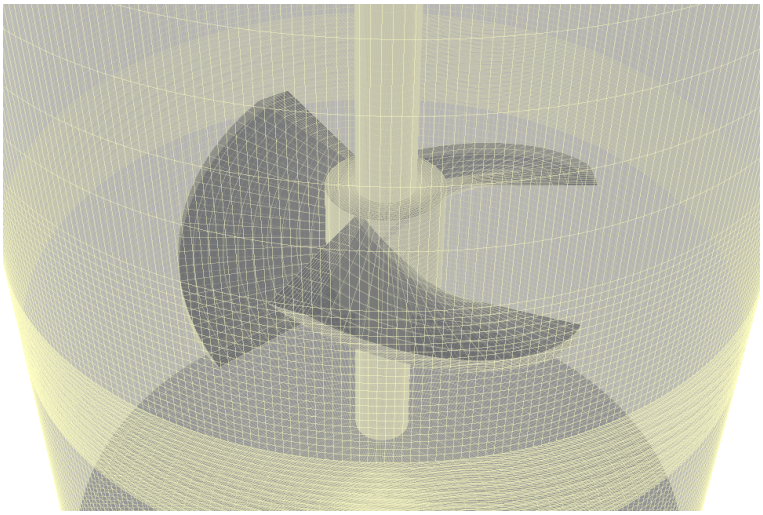

Fig 2. Three lobbed marine impeller

The three lobbed marine impeller is what the script is built for. The only little defect is that the blade is thinner close to the impeller. It's not perfect, but it contrasts greatly with the paper-thin approximation!

```
DIVI       = [  10,  10, 20]
RQUAD       = [ 0.05, 0.1, 0.350, 0.615]
NSEC       = 36
HLAY       = [ 0.25, 0.60, 0.80,1.05,2.75]
SHAFT      = [  0,  1,  1,  1,  1]
LVLROT     = [  0,  0,  90,  90,  90,  90]
IMPELLERCUT = np.zeros((NSEC,len(RQUAD),len(HLAY)))
IMPELLERCUT[::(NSEC//3),[0,1],2] = 1
IMPELLERCUT[0:NSEC,0,2] = 1
```

If you like this geometry, you just have to change the value to correspond to the size you wish!

4.2 8 blade turbine

This interesting example is not hard to create. Use the same setting has the *three lobbed marine* and change how the impeller is cut.

```
IMPELLERCUT[::3, 0, 1] = 1
```

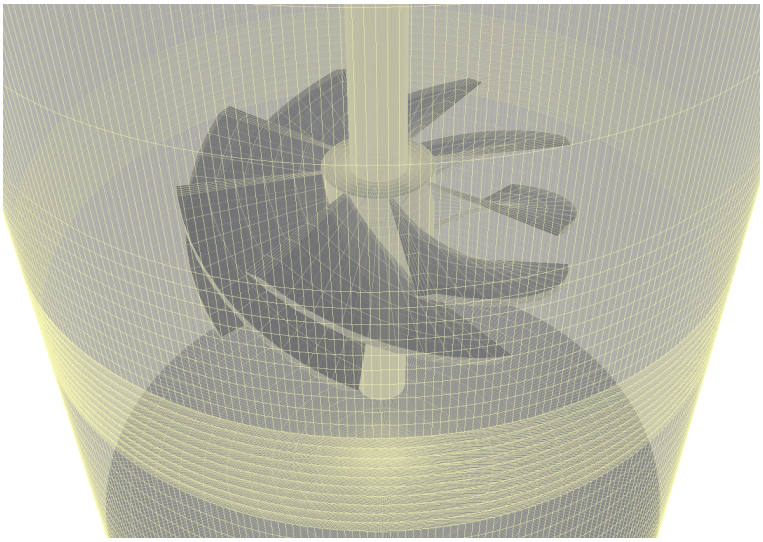

Fig 3. Three lobbed marine impeller

4.3 Rushton (Easy)

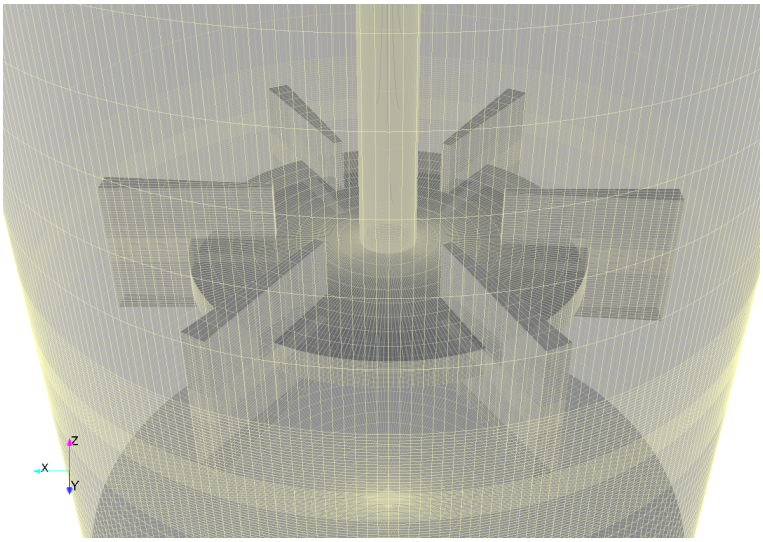

Fig 4. Three lobbed marine impeller

This interesting example is not hard to create. Use the same setting has the *three lobbed marine* and change how the impeller is cut.

```
DIVI       = [  2,  2, 20]
RQUAD       = [ 0.05, 0.2, 0.350, 0.5, 0.615]
NSEC       = 72
HLAY       = [ 0.50, 0.60, 0.65,0.75,2.75]
SHAFT      = [  0,  1,  1,  1,  1]
LVLROT     = [  0,  0,  0,  0,  0,  0]
IMPELLERCUT = np.zeros((NSEC,len(RQUAD),len(HLAY)))
IMPELLERCUT[::(NSEC//6),[1,2],1] = 1
```

```
IMPELLERCUT[::(NSEC//6),[1,2],2] = 1
IMPELLERCUT[::(NSEC//6),[1,2],3] = 1
IMPELLERCUT[0:NSEC,[0,1],2] = 1
```

5 Advanced Tweaking

As it, superMarine isn't perfect. The power of a script lie in the availability of a comprehensible code for the end user. You can, if you wish, tweak to code to fit it to your need. I you do so, please cite the original work.

This section aims to give enough information about the code to allow an easy tempering. It also has few examples of such modification.

HINT: Remember, you can use *paraFoam -block* or even *matplotlib.pyplot* to assist you with your modification!

5.1 Algorithm

(1) Setup Check
  (a) Quadrant Check
  (b) Sections
(2) Generate Geometry
  (a) Vertex rings, counterclockwise, from the center to the perimeter.
  (b) Vertex layers, from bottom to top.
  (c) Rings Hexahedrons.
  (d) Center scaffolding and grid generation.
  (e) Center Hexahedrons.
  (f) Centers to Ring Hexahedrons.
  (g) Layers Rotations.
  (h) Cylinders Arcs.
  (i) Vertical Splines.
  (j) Patch.
(3) Generate blockMesh dictionary

The setup check is quite summary for now.

6 Advanced Example

6.1 Cone impeller

An example in which the spline functionality has to be deactivated (or changed) is when the impeller is bent inward. An example would be a cone shaped impeller. Wonder why this could be useful? It's a secret for now!

Settings

```
DIVI       = [  10,  10, 10]
RQUAD       = [ 0.009, 0.074]
NSEC       = 12
HLAY       = [ 0.03, 0.200]
SHAFT      = [   1,    0]
LVLROT     = [   0,    0,    0,    0,    0,   0]
IMPELLERCUT = np.zeros((NSEC,len(RQUAD),len(HLAY)))
```

Modification 1: Closing bottom ring

```
# CONE 15 VERTEX TWEAKS MOD1
oCylId=lambda omega,r,z:omega+r*NSEC+z*NSEC*len(RQUAD)
bottomVertex = [ oCylId( o, 2, 2) for o in range(0, NSEC)]
bottomVertex += [ oCylId( o, 3, 2) for o in range(0, NSEC)]
adjustLen = 0.05*np.tan(toRad(CONEANGLE))
for v in bottomVertex:
   unitAngle = [1,1,0]*vertex[v]/np.linalg.norm(vertex[v])
   vertex[v] -= unitAngle*adjustLen
# END MOD1
```

Modification 2: Comment out the spline algorithm

```
#-- VERTICAL SPLINE --#
# This force the vertical lines to pass
# on the outer cylinder instead of behing a straight line

# CONE 15 NO SPLINE TWEAK MOD2
spedge = []
#for H in range(0,len(HLAY)):
#  for cadran in range(0,nCad):
#     for sector in range(0,NSEC):
#        first  = sector+cadran*NSEC+H*nCad*NSEC
#        second = sector+cadran*NSEC+(H+1)*nCad*NSEC
#        unitH =
    np.array([0,0,1])*(vertex[second]-vertex[first])/DIVI[2]
```

```
#       fv    = vertex[first]*[1,1,0]
#       sv    = vertex[second]*[1,1,0]
#       if fv[0]!=sv[0] or fv[1]!=sv[1]:
#          unitR =
    (np.arccos(np.dot(sv,fv)/(np.linalg.norm(sv)*np.linalg.norm(fv))))/DIVI[2]
#          iterp = [rotz(vertex[first],p*unitR)+p*unitH for p in
    range(1,DIVI[2])]
#          spedge.append([first,second,np.array(iterp)])
# END MOD2
```

Modification 3: No Shaft patch

```
#MOD3
#blockMesh+=["shaft","{","type wall;","faces","("]
#for w in shaftPatch:
#  blockMesh.append(ftem.format(*w))
#blockMesh+=[");","}"]
#END MOD3
```

6.2 Rushton Perfect (Advanced)

Start by using the same configuration as the *rushton simple* example.

```
DIVI       = [  2,  2, 20]
RQUAD       = [ 0.05, 0.2, 0.350, 0.5, 0.615]
NSEC       = 72
HLAY       = [ 0.50, 0.60, 0.65,0.75,2.75]
SHAFT      = [  0,  1,  1,  1,  1]
LVLROT     = [  0,  0,  0,  0,  0,  0]
IMPELLERCUT = np.zeros((NSEC,len(RQUAD),len(HLAY)))
IMPELLERCUT[::(NSEC//6),[1,2],1] = 1
IMPELLERCUT[::(NSEC//6),[1,2],2] = 1
IMPELLERCUT[::(NSEC//6),[1,2],3] = 1
IMPELLERCUT[0:NSEC,[0,1],2] = 1
```

Then you have to adjust the vertex corresponding to the edge of the blades.