Fibers, Filter Generation

by

Robabeh Izadshenas

Institut für Mechanische Verfahrenstechnik (IMVT)
Universität Stuttgart

Contents

0.1 Introduction

Executing industrial projects in real life need some simulations in a different area. The separation of solids from fluid provides us productive industrial tools. For instance, in car industry, the strong performance of cars depends on their filters when they are able to separate dust from the air, engine oil, and etc. So, the particle filtration simulation is one of the most significant research topics in recent years in CFD and CFDEM fields. Therefore, the development and finding an optimized method to design a filtration system are in advantage.

This study is not dropping straight fibers inside the domain. However, generate the fibers by the straight path and deform the path which is the desired deformed shape in µm. To develop the model, the fibers are following the Euler rotation by von-Mises and uniform distribution, and transformation by uniform one. To make it more productive filter geometry, an intersection function has defined, which express that one can consider and separate different size of particles during the simulation inside the domain. Another advantage of this filter geometry is that following the natural phenomena of the rigid body of each object on top of each other to preserve the adhesive style. For this study, python 3.7, Blender 2.8, and Paraview 5.8.0 have been used.

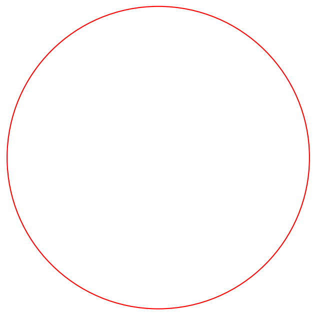


Figure 1: Bezier Circle

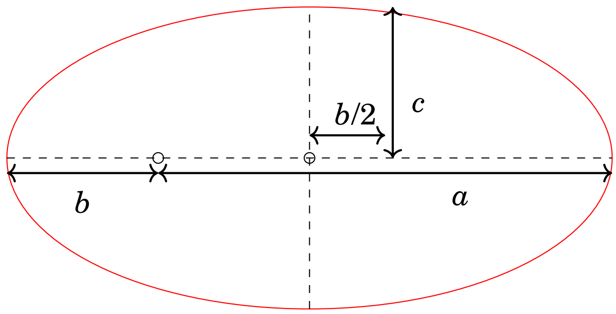


Figure 2: Bezier Ellipse

0.2 Fibers, Filter Generation

Filter geometry is based on some steps, which have been done with the combination of python scripts and software blender. First of all, the user should invoke desire python libraries and define the essential parameters like desired packing density c_p, the diameter of fibers d, computational domain size in the cross-section w, filter depth along the direction of flow l. Also, defining a physical domain by (l,w,w) as a cube with a length of 1mm is required, which is shown in figure 4a.

In this study fibers are generated with the help of bezier circle like figure 1 and a path. Since a specific ellipsoid owl was the purpose, the circle converted to the ellipse with the parameters $a = 20\,\mu\text{m}$, $b = 10\,\mu\text{m}$, and $c = 10\,\mu\text{m}$, which it is shown in figure 2. Therefore, the bezier circle converted to an ellipse by scaling the circle with ellipse's width equal to $a+b$ and the distance of hight c from the origen by $b/2$. Also, the scale version of distance $b/2$ in the unit circle is considered as $x = \frac{b/2}{(a+b)/2} \times \frac{\pi}{2}$. and $y = \sqrt{(1-x^2)}$. Therefore, by considering the eccentricity factor as c/y, resize the bezier circle by $\frac{a+b}{2}$ and the factor [1], [2].

(a) A path with 5 nodes

(b) Omitted two nodes

(c) Direction of new middle point

(d) Deformed path

Figure 3: Deformation steps

Once we have got the desired ellipse shape, a path needs to be defined along the X axis with the first vertices is -1.5 and the last one is 1.5. To deform the path, a specific deformed function has been written [3], [4]. Since it has 5 nodes, the second, and fourth nodes should be removed and then consider a path with two endpoints and a middle point. The idea is, a vector between the middle point and one endpoint should be found. To do so, imagine a plane in midpoint who's normal leis in the direction of the vector between the middle point and the endpoint [5]. Therefore, an arbitrary random point has chosen on the plane, which is perpendicular to the distance vector of the middle and endpoint. Furthermore, normalized the vector length by L^2 norm. Then moved the midpoint out by a scaled value along the vector direction which is guaranteed to be orthogonal to the path, which is depicted in figure 3.

In the next step, the deformed path will be generated successively by random Euler rotation such that the angle β between the path and the flow direction along Y- axis follows statistical von-Mises distribution, which the following probability density [6], [7]:

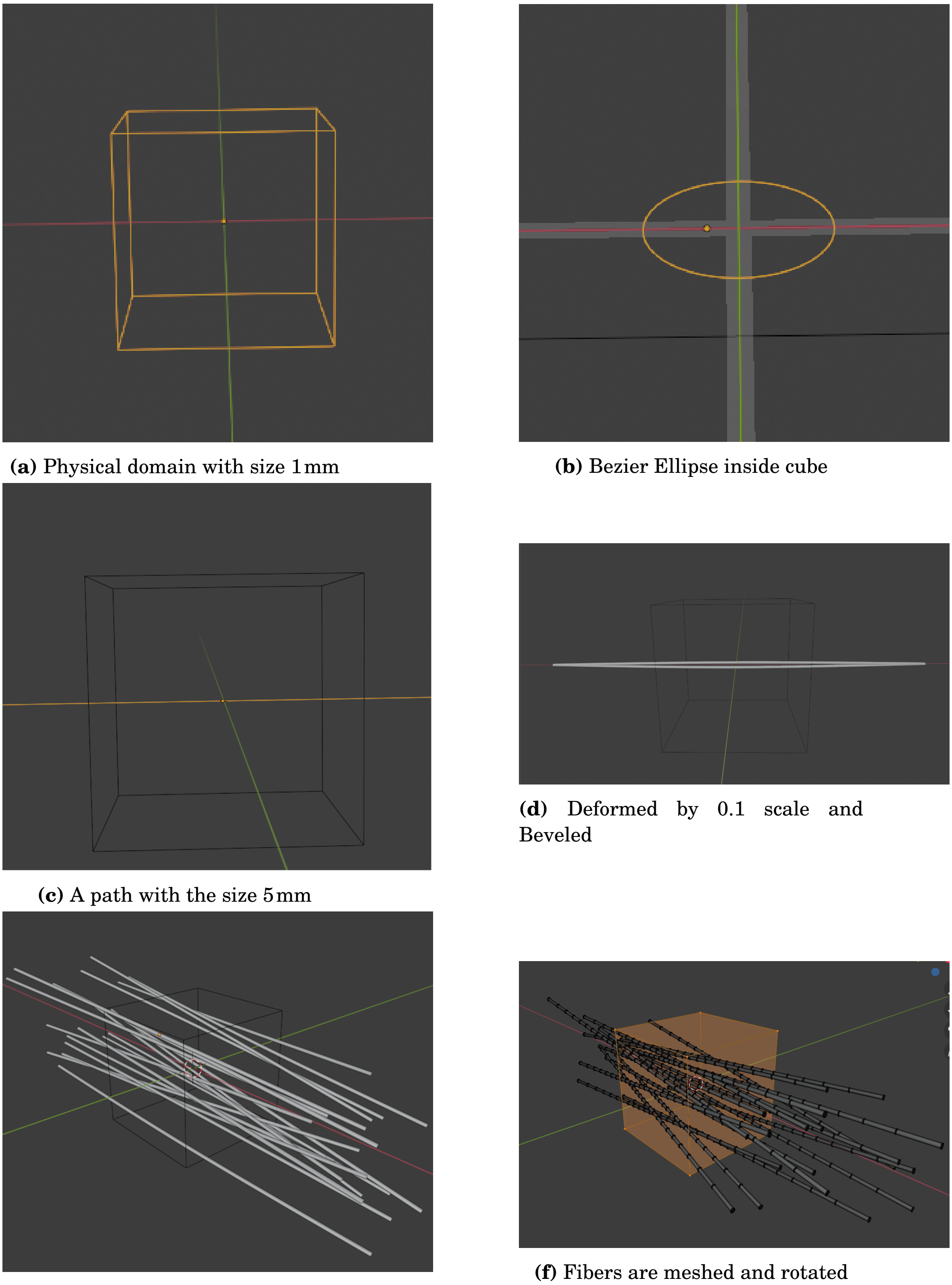
$$f(\beta, C_p) = \frac{e^{C_p \cos(\beta)}}{2\pi I_0(C_p)} \qquad for \quad -\pi < \beta < \pi \tag{1}$$

where C_p is the fiber concentration factor, and $I_0(C_p)$ is the modified Bessel function of order 0. Subsequently, the fiber elements are randomly transformed and rotated along X,Y, Z axis, by uniform distribution in the physical domain in the ranges $[(-l/2, l/2), (-w/2, w/2), (-w/2, w/2)]$ and $(0, \pi/10)$, respectively [8]. In order to get fibers from paths, the objects are beveled with the help of generated bezier ellipse shape which is shown in figure 4b. Of course, the fibers also need mesh, therefore, they converted to mesh. To go further, in this way fibers need a second round of rotation in order to visualize the cross-section much more horizontally. Therefore, they are rotated with the global orient type and with values 1.2, 0.1, -0.25 for the orient axis X, Y, Z, respectively.

To follow the purpose behind this study, an intersection function is defined in order to recognize the intersections between the fibers. Of course, there are some overlaps but there are no intersections, see the figure 5b, which is clear by taking a slice of geometry in Paraview.

In order to recognize the collision between fibers, since the fibers are originally stored in a list, therefore, we can have access to their verities and polygons and invoke the BVHTree function with respect to them. Here BVH is the abbreviation of bounding volume hierarchy is a tree structure on a set of geometric objects [9]. Thus, a new list that consists of only fibers(not physical domain and not bezier ellipse) could help us in case if there is an intersect between them delete those fibers [10].In this case, when an object has removed from the list, there would be a chance to do an additional try. This means that there is a counter which counts how many times the algorithm had been applied and the desired maximum times specifies the maximum times that the algorithm is allowed to implement. Another way around when there is no intersection, keep the objects in the list, obviously, there is no additional try in this case.

To have some information of how much empty space left after generating fibers in the physical domain, make volume calculation of the fibers and domain are considered. Therefore, defining a ratio of them as $\alpha = \frac{Fibers'Volume}{Domain'sVolume}$ will be easy. In the next step, the ratio α and the defined packing density will be compared. If α is still less than the packing density, then we have to try more. Additionally, if the number of tries which is counted already by intersection is bigger than the maximum tries, the simulation will be stopped and consider α as a packing density and if α is bigger than the packing density it will end up with the satisfaction of packing density α. In order to get better intuition, check the figures 4, 5 and its flowchart.



(a) Physical domain with size 1 mm

(b) Bezier Ellipse inside cube

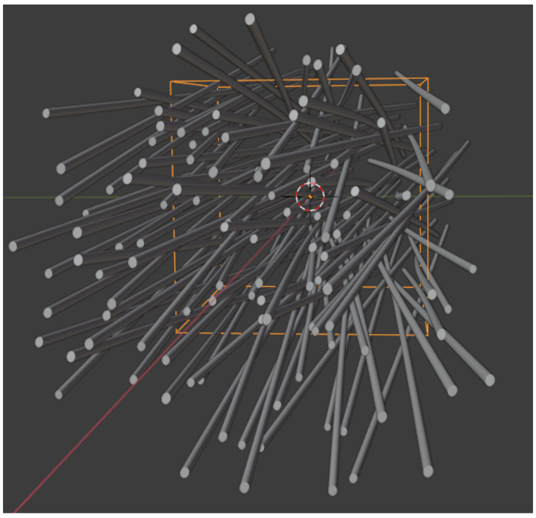
(c) A path with the size 5 mm

(d) Deformed by 0.1 scale and Beveled

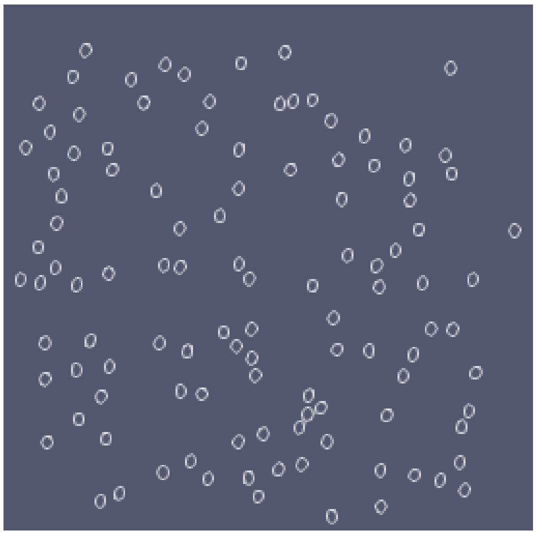
(e) Deformed, von-Mises distributed and Beveled

(f) Fibers are meshed and rotated

Figure 4: Steps of Fibers Generation



(a) Intersection has checked

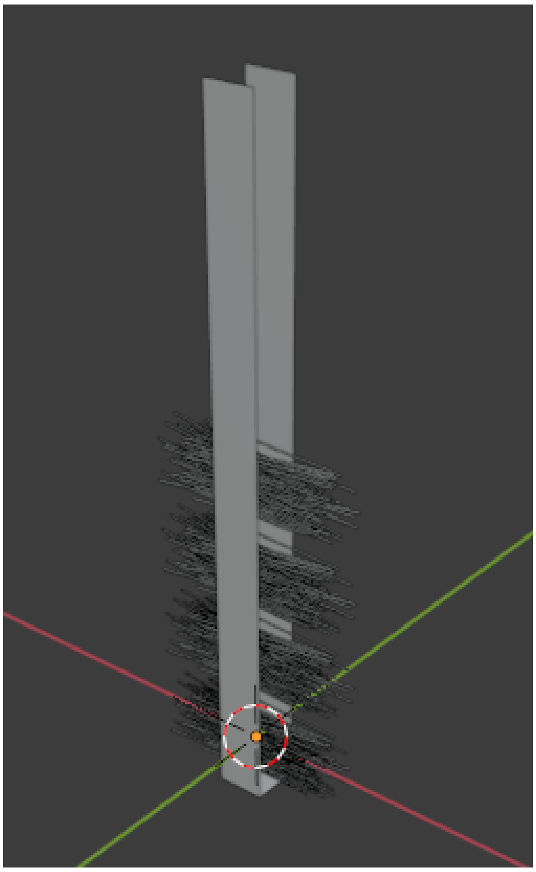


(b) Sliced geometry in Paraview

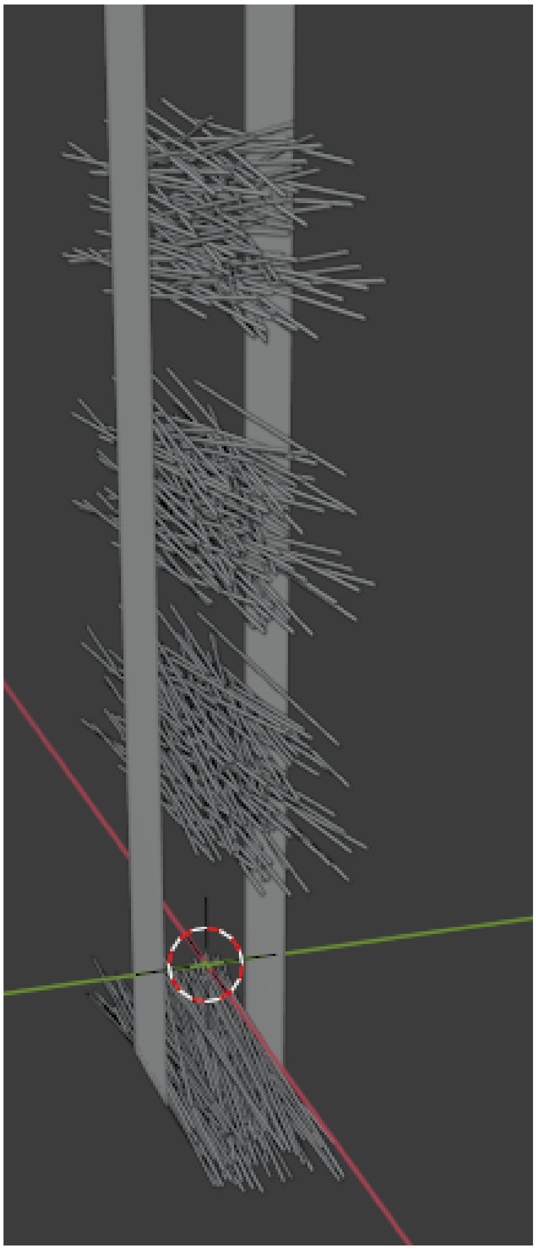
Figure 5: Steps of Fibers Generation

In this study, because of the intersection function, we might not able to add many fibers to fill the physical domain, therefore, the rigid body idea came up. To apply rigid body [11], a plane in the $-Z$ direction parallel to the ground is needed. To prevent falling the fibers outside the physical domain the other two surfaces, which are completely in front of each other on the left and right sides are made. The width of these surfaces is following the cube size however their length is four times larger than the cube height.

In order to apply a rigid body, the first group of fibers is copied four-times and pasted on top of each other with some distance. The rigid body setting for surface and fibers is set to passive and active, respectively. To have an active rigid body, the last fiber in each group is considered as an active object with the setting collisions shape convex-hull, collision source final, friction 0.5, bounciness 0, collision margin 0, damping translation and rotation 0.9, 0.1, respectively. Then, one should select each group of fibers except the surfaces, and from the object's tab choose rigid body/ copy from the active object. Of course, the active object should be the one that had got the rigid body setting. Some settings in the scene also required, considering unit scale length in millimeters. In rigid body world, the setting is considered by steps per seconds 120 and solver iteration 10, and cache with simulation starts at 1 and ends 500 which the steps are shown in figures 6,7.

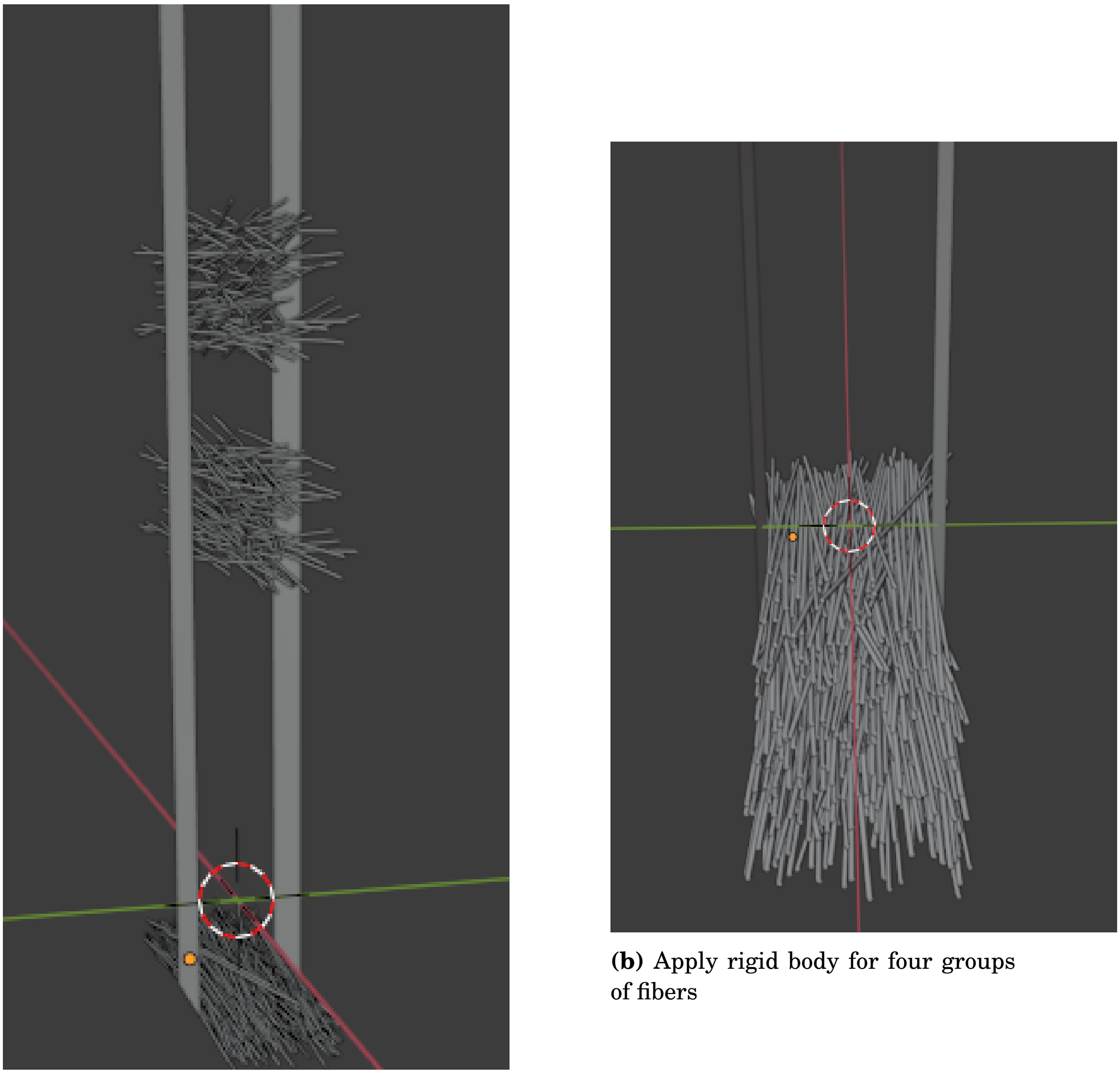


(a) Generate surfaces and four groups of fibers



(b) Apply rigid body for first group

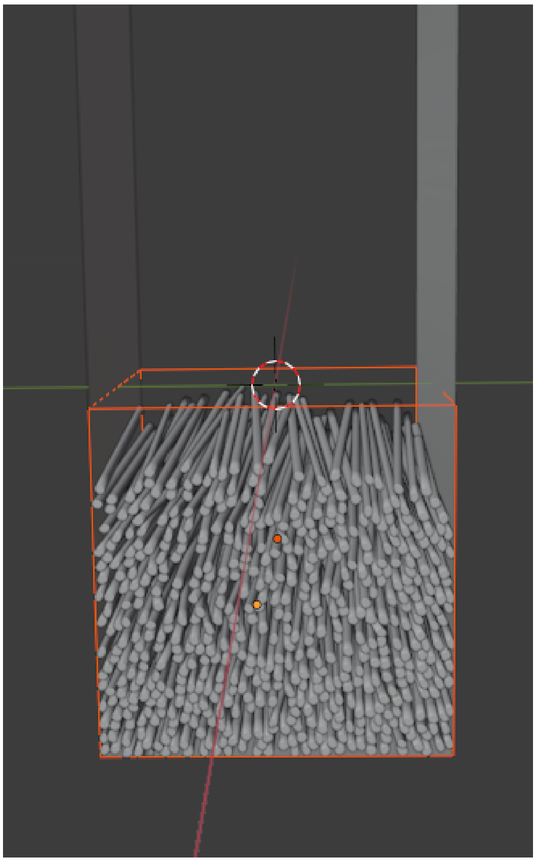
Figure 6: Steps of Rigid Body



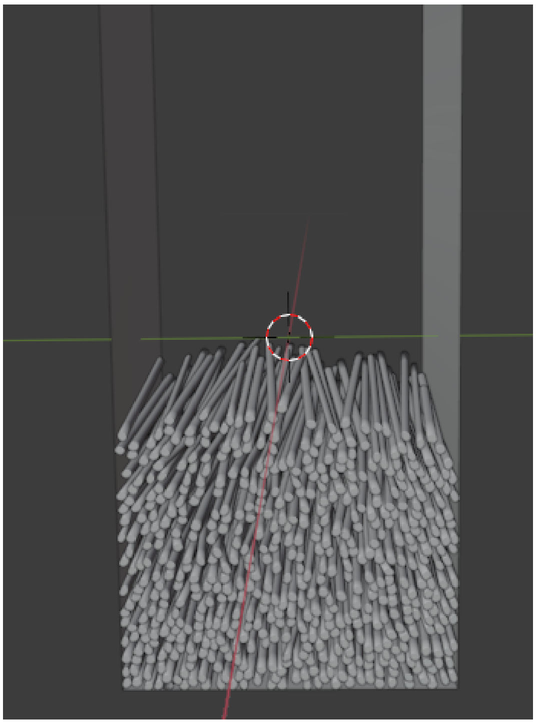
(a) Apply rigid body for first and second groups of fibers

(b) Apply rigid body for four groups of fibers

Figure 7: Steps of Rigid Body

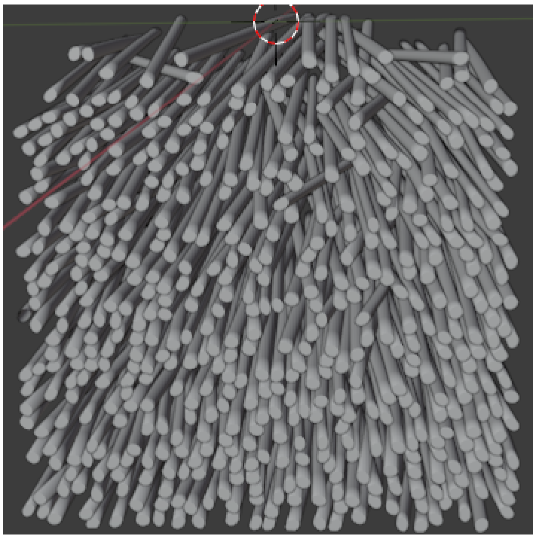


(a) Applied Boolean Modifier

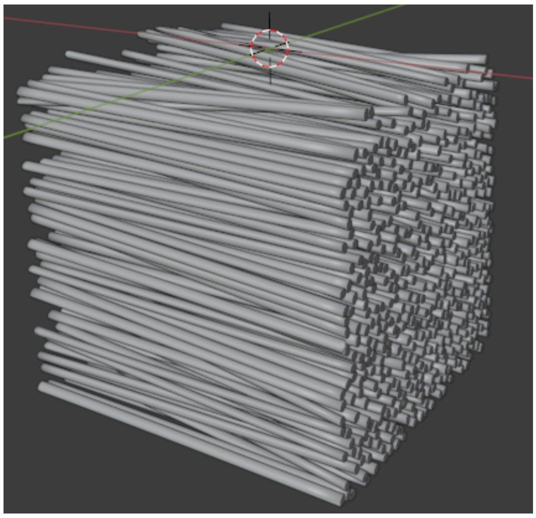


(b) Cube deleted

Figure 8: Boolean Modifier



(a) Final filter geometry front view



(b) Final filter geometry left view

Figure 9: Filter geometry

In this step, it is time to fix the size of fibers with respect to the size of the bounding box. Therefore, the original cube is added and translated with respect to the position of fibers in such a way we should not have cut fibers off after the boolean cut on the top and bottom sides. The boolean modifier[12] performs the boolean operations in 3D geometry with 3 different operations, which here the INTERSECT one has used. The picture 8 depicts the boolean modifier how to cut the length of the objects off when they had intersected with the cube sides.

Thus, after finishing all the above steps, the geometry needs to be exported as an STL file. So delete the bounding box, choose all objects except surfaces in 3D view, go to File/Export choose stl. Here the setting batch Mode off, active selection only, scene units, and apply modifiers are considered. The figures 9, 10 are shown the final shape of the STL file and sliced one, which there are some overlaps and few numbers of intersections. However, if a few fibers have considered in the first group before the rigid body, there might not be intersections. Also, this rigid body setting is the one that has made fewer intersections. It is necessary to notice that before using the STL file for simulation, one should be quite sure that the mesh of the objects are triangulated and are following the properties of manifold geometry.

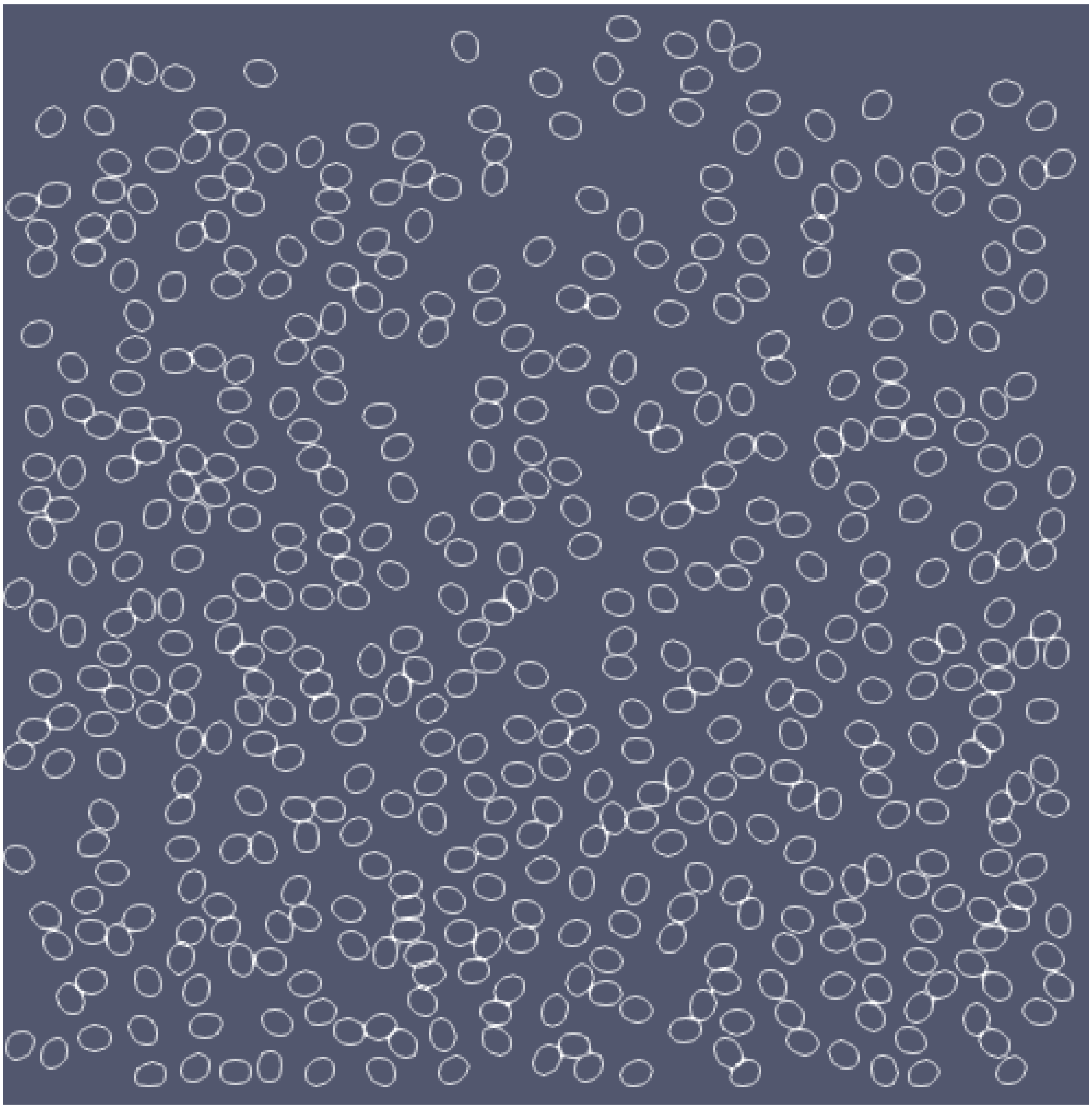
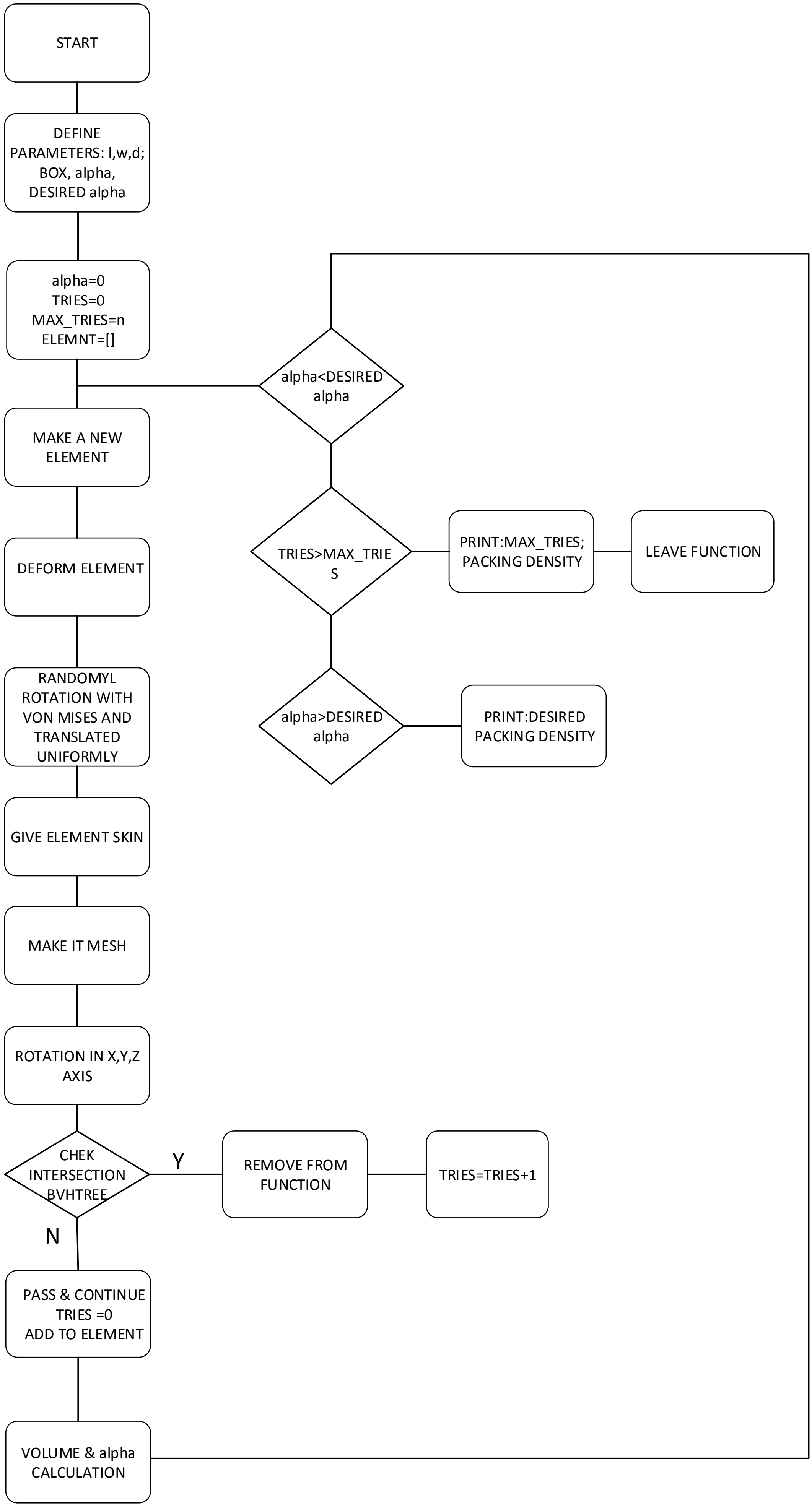


Figure 10: Sliced filter geometry in Paraview


START
DEFINE PARAMETERS: l,w,d; BOX, alpha, DESIRED alpha
alpha=0
TRIES=0
MAX_TRIES=n
ELEMNT=[]
alpha<DESIRED alpha
MAKE A NEW ELEMENT
TRIES>MAX_TRIES
PRINT:MAX_TRIES; PACKING DENSITY
LEAVE FUNCTION
DEFORM ELEMENT
RANDOMYL ROTATION WITH VON MISES AND TRANSLATED UNIFORMLY
alpha>DESIRED alpha
PRINT:DESIRED PACKING DENSITY
GIVE ELEMENT SKIN
MAKE IT MESH
ROTATION IN X,Y,Z AXIS
CHEK INTERSECTION BVHTREE
Y
REMOVE FROM FUNCTION
TRIES=TRIES+1
N
PASS & CONTINUE
TRIES =0
ADD TO ELEMENT
VOLUME & alpha CALCULATION

Bibliography

[1] [Online]. Available: https://blender.stackexchange.com/questions/67069/how-to-draw-a-flat-ellipse-surface-in-blender-with-the-following-known-dimension

[2] [Online]. Available: https://mathworld.wolfram.com/Ellipse.html

[3] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[4] G. Elber, "Iii.7 - interpolation using bezier curves," in *Graphics Gems III (IBM Version)*, D. KIRK, Ed. San Francisco: Morgan Kaufmann, 1992, pp. 133 – 136. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080507552500373

[5] [Online]. Available: https://stackoverflow.com/questions/23596802/calculate-middle-point-of-bezier-curve

[6] [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.vonmises.html

[7] M. Habeck, "Generation of three-dimensional random rotations in fitting and matching problems," *Computational Statistics*, vol. 24, no. 4, p. 719, 2009.

[8] S. Abishek, A. King, R. Mead-Hunter, V. Golkarfard, W. Heikamp, and B. Mullins, "Generation and validation of virtual nonwoven, foam and knitted filter (separator/coalescer) geometries for cfd simulations," *Separation and Purification Technology*, vol. 188, pp. 493–507, 2017.

[9] [Online]. Available: https://en.wikipedia.org/wiki/Bounding_volume_hierarchy

[10] [Online]. Available: https://blender.stackexchange.com/questions/71289/using-overlap-to-check-if-two-meshes-are-intersecting

[11] [Online]. Available: https://docs.blender.org/api/current/bpy.ops.rigidbody.html

[12] [Online]. Available: https://docs.blender.org/api/current/bpy.types.BooleanModifier.html

Appendix A

Fibers and Filter Generation Python Codes

A.1 Python Codes

```
#Fiber, Filter Generation
#import python libraries
import bpy
import mathutils
from mathutils import Vector
from mathutils import geometry
from mathutils.bvhtree import BVHTree
import math
from scipy.stats import vonmises
import random
import numpy as np
import bmesh
#from math import radians
#from mathutils import Matrix

#bpy.context.scene.view_layers[0].cycles.use_denoising = True
#bpy.context.scene.unit_setting.scale_length= 0.001

#bpy.data - all data in your blend file.
#bpy.context - data in the current active view.
#bpy.ops - tools which typically operate on bpy.context
#bpy.context.object                # Active object (last object
    selected)
#bpy.context.selected_objects      # All selected objects
#context.scene.objects             # All objects in current
   scene
#bpy.data.objects                  # All objects
#bpy.data.meshes                   # All meshes


```

```
context = bpy.context

scene = context.scene
#scene.unit_settings.length_unit = 'MILLIMETERS'

scene.unit_settings.scale_length = 0.001
#empties = [o for o in scene.objects if o.data is None]

#### defining global variables

#a = 0.5 # packing_density
d = 0.01 # diameter of fibre
w = 1 # filter cross section
l = 1 # filter length in direction of flow
Cp = 0.5 #packing density


alpha = 0.0
alphaGoal = 0.1
max_tries=50
tries= 0.0
volCyl=0.0
volbCyl=0.0
volbcCyl=0.0
volcCyl=0.0

###### physical domain cube & planes
cube = bpy.ops.mesh.primitive_cube_add(size=1)
cube = bpy.context.object
cube.name = "Domain"
cube.display_type = 'BOUNDS'



#mesh the cube
cubeMesh = bmesh.new()
cubeMesh.from_mesh(bpy.context.object.data)
volDomain = cubeMesh.calc_volume(signed = False)
cubeMesh.free()
element = []
# Add bezier circle

bpy.ops.curve.primitive_bezier_circle_add(align='WORLD',
    enter_editmode=False, location=(0, 0, 0))
obj = bpy.context.object
obj.name = "BezierCircle"
obj.data.name = "BezierCircle"

# Change bezier circle in to elliptical shape
#define  its variables
a = 0.02
b = 0.01
```

```
c = 0.01
#d = 0.01


  #Calculate width
width = a + b   #or width = a + d + b?

  #Calculate horizontal distance from world origin of c
distance = b / 2

  #Calculate height c for the circle
  #x^2 + y^2 = 1, see Unit Circle
x = (distance / (width / 2)) * math.pi * 0.5

cCircle = math.sqrt(1 - x**2)

  #now we have to scale the height of the circle, so cCircle becomes c. And give its width.
factor = c / cCircle

#resize the bezier circle w.r.t width/2 and factor
bpy.ops.transform.resize(value=((width)/2, factor, 1), constraint_axis=(True, True, False), orient_matrix_type='GLOBAL', mirror=False, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, release_confirm=True) #proportional_edit_falloff='SMOOTH', proportional_size=1)


  #Calculate location origin
originX = -((width) / 2 ) + b
originY = 0
originZ = 0

  #Set 3d Cursor at Origin location
bpy.context.scene.cursor.location = (originX, originY, originZ)

  #Set origin to 3d cursor
bpy.ops.object.origin_set(type='ORIGIN_CURSOR')

#add path
def vec2vec_cylinder():

  cyl = bpy.ops.curve.primitive_nurbs_path_add()
    # go to edit mode
  bpy.ops.object.editmode_toggle()
  # select the path
  bpy.ops.curve.select_all(action ='DESELECT')
```

```
# current active object
cyl = bpy.context.object
  #set the first and the last vertices position in x-
 direction
cyl.data.splines[0].points[0].co.x= -1.5
cyl.data.splines[0].points[4].co.x= 1.5

#select the second and forth vertices
cyl.data.splines[0].points[1].select = True
cyl.data.splines[0].points[3].select = True
  #delete those vertices
bpy.ops.curve.dissolve_verts()
# change edit mode to object mode
bpy.ops.object.editmode_toggle()

#give name to an active object and to list
sceneObject =bpy.context.scene.objects[cyl.name]
element.append(cyl.name)

# deformation object
def deform(element):
    global volCyl
    global volbCyl
    global tries
    global alpha

    cyl = sceneObject
    bpy.context.view_layer.objects.active = cyl
          # consider  the middle point and end point
    midpoint = cyl.data.splines[0].points[1]
    endpoint = cyl.data.splines[0].points[2]

    #get the difference between middle point and end point
    def getvector(v):
      vx = v[0]
      vy = v[1]
      vz = v[2]

      #set the perpendicular condition in each plane
      # vx*x +vy*y+ vz*z = 0
      if vz != 0:
         x = random.random()-0.5 # random position of x
         y = random.random()-0.5 # random position of y
         z = (-vx*x-vy*y)/vz      # apply the perpendicular
 condition to get z
         Dis = ((vx-x)**2 +(vy-y)**2+(vz-z)**2)*0.5  # find
 Euclidean distance
      #return a vector of  normalization of the new position
 by its distance
         return [x/Dis,y/Dis,z/Dis]

      if vy != 0:
```

```
            x = random.random()-0.5
            z = random.random()-0.5
            y = (-vx*x-vz*z)/vy
            Dis =((vx-x)**2 +(vy-y)**2+(vz-z)**2)**0.5
            return [x/Dis,y/Dis,z/Dis]

         if vx != 0:
           y = random.random()-0.5
           z = random.random()-0.5
           x = (-vy*y-vz*z)/ vx
           Dis =((vx-x)**2 +(vy-y)**2+(vz-z)**2)**0.5

           return [x/Dis,y/Dis,z/Dis]


     # difference between middle point and end point
     lengthVector = Vector(midpoint.co[:3])- Vector(endpoint.
  co[:3])
     # scaling factor
     m = 0.1
     # m* direction
     mdirecVector = Vector([co *m for co in getvector(
  lengthVector)] + [1.0])
     # new middle point + m *  direcion
     midpoint.co = midpoint.co + mdirecVector


 # Apply the euler rotation by von Mises distribution along Y-
   direction
 # Apply the transformation, euler rotation along X, Y, Z by
  the uniform distribution
 def vonMises(element):
   cyl.rotation_euler[1] = vonmises.ppf(random.random(),Cp)
   cyl.location = (np.random.uniform(-l/2, l/2) , np.random.
  uniform(-w/2, w/2), np.random.uniform(-w/2, w/2)) # r
   cyl.rotation_euler[0] = np.random.uniform(0, math.pi/10) #
  phi, rotates about x-axis
   cyl.rotation_euler[1] = np.random.uniform(0, math.pi/10)
   cyl.rotation_euler[2] = np.random.uniform(0, math.pi/10)

   bpy.context.view_layer.update() # required to update
  transformations made to the object

    # Add skin to the path with the help of bezier circle
 def addBevel(element):
   obj = sceneObject
   bpy.context.view_layer.objects.active = obj

   cyl.data.bevel_object = bpy.data.objects["BezierCircle"]
   cyl.data.use_fill_caps = True # caps the ends of the tubes


```

```
# for BVHTree to work, the fibre elements must have mesh data , therefore objects need to be meshed
def convertToMesh(element):
  #obj = bpy.context.scene
  obj = sceneObject
  bpy.context.view_layer.objects.active = obj
  bpy.ops.object.convert(target='MESH')
  # second round of rotation of fibers a long  X, Y, Z  axis
def rot(element):

  obj = sceneObject
  bpy.context.view_layer.objects.active = obj
  #for ob in bpy.context.scene.objects:
  obj.select_set(True)
  bpy.ops.transform.rotate(value= 1.2, orient_axis='X', orient_type='GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orient_matrix_type='GLOBAL', constraint_axis=(True, False, False), mirror=True, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, release_confirm=True)
  bpy.ops.transform.rotate(value= 0.1, orient_axis='Y', orient_type='GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orient_matrix_type='GLOBAL', constraint_axis=(True, False, False), mirror=True, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, release_confirm=True)
  bpy.ops.transform.rotate(value= -0.25, orient_axis='Z', orient_type='GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orient_matrix_type='GLOBAL', constraint_axis=(True, False, False), mirror=True, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, release_confirm=True)



global volCyl

global tries

global alpha

# intersection function using bvhtree library, read vertices and polygons
def get_bvh(element):
  mat = element.matrix_world
  vert =[mat @ v.co for v in element.data.vertices]
  poly = [p.vertices for p in element.data.polygons]
  return BVHTree.FromPolygons( vert, poly )
# invoke bvhtree inside the intersection function
```

```
def intersected(element):
  allelements= [obj for obj in bpy.data.objects if obj.name
 != "Domain"  and  obj.name != "BezierCircle"]
  #print("element".format(element))
  #print(" allelements: {}".format(allelements))
  allelements.remove(bpy.data.objects[element.name])# if
 there is an intersection remove them
  #print(" allelements: {}".format(allelements))

  # check the  objects are overlapping or not
  # element is the list of all fibers, and allemlemnts is a
 new list which consist of all fibers
  for obj in allelements:
     if get_bvh(obj).overlap(get_bvh(element)):
       return True

     else:
       pass
  return False
  #
  #This function calculates the volume of all objects and
 domain and their ratio
def volume_ratio(element):
  global volCyl

  global tries
  global alpha

  element = bpy.context.object


  cylMesh = bmesh.new()
  #Initialize this bmesh from existing mesh datablock.
  cylMesh.from_mesh(bpy.context.object.data)
  #mesh (Mesh)     The mesh data to load(bpy.context.object.
 data)
  volCyl= volCyl + cylMesh.calc_volume(signed = False)
  #Explicitly free the BMesh data from memory, causing
 exceptions on further access.
  cylMesh.free()
  alpha = volCyl/volDomain
  print("Packing Density : {}".format(alpha))
  #logging.debug("Packing Density : {}".format(alpha))
  volSpace = volDomain-(alpha *volDomain)
  print("Volume Space : {}".format(volSpace))
  #logging.debug("Volume Space : {}".format(volSpace)) #
 logging is to print in  log file

  # Call  all  functions which are inside the
 vec2vec_cylinder()
  # vec2vec_cylinder() will be called inside while loop
deform(cyl)
```

```
  vonMises(cyl)
  addBevel(cyl)
  convertToMesh(cyl)
  rot(cyl)


  # if there is an intersection delete it
  if intersected(cyl):
    bpy.ops.object.delete()
    # since an object is deleted you try once more
    tries = tries +1
  else:
    # if there is not an intersection add the object to the list
    element.append(cyl)
    # calculate volume
    volume_ratio(cyl)
    # sinc there is no intersection you cannot try more
    tries = 0
    #print("element: {}".format(element))

  return
# since alpha is the ratio of (vlume of objects / volume of domain) if this is less than desired alpha,
# then you can add another objects
while alpha < alphaGoal:
   vec2vec_cylinder()
   # if condition to follow our maximum tries
   if tries > max_tries:

     print('attempts at insertion reached:{}'.format(max_tries))
     print('Packing Density:{}'.format(alpha))

     break
     # stop when alpha is bigger than desired packing density

   if  alpha > alphaGoal:
      print("Packing Density Satisfied")
      print('Packing Density:{}'.format(alpha))

# delte the cube
#bpy.ops.object.select_all(action='DESELECT') # Deselect all
#bpy.data.objects["Domain"].select_set(True) # Blender 2.8x
#bpy.ops.object.delete()
```

Listing A.1: Main Code for Fibers and Filter Generation

```
# Creat surfaces to apply Rigid Body
# import Python libraries
import bpy
import mathutils
from mathutils import Vector
from mathutils import geometry
from mathutils.bvhtree import BVHTree
import math
from scipy.stats import vonmises
import random
import numpy as np
import bmesh
# Generate physical domain cube & planes

cube = bpy.ops.mesh.primitive_cube_add(size=1)
cube = bpy.context.object
cube.name = "Domain"
cube.display_type = 'BOUNDS'
#Take the cube size  and design the surfaces and mesh each surface with its verieses
# Apply passive rigid body to each surface
def createPlane(cube):

    obj = bpy.context.object
    coordinates = [obj.matrix_world @ v.co for v in obj.data.vertices]

    x_list = [co.x for co in coordinates]
    y_list = [co.y for co in coordinates]
    z_list = [co.z for co in coordinates]


    x_min, x_max = min(x_list), max(x_list)
    y_min, y_max = min(y_list), max(y_list)
    z_min, z_max = min(z_list)-0.5 , max(z_list)+12

    verts1 = ((x_min, y_max, z_min),(x_min, y_min, z_min),(x_max, y_min, z_min),(x_max, y_max, z_min))

    verts2 = ((x_min, y_min, z_min),(x_max, y_min, z_min),(x_max, y_min, z_max),(x_min, y_min, z_max))

    verts3 =((x_min, y_max, z_min),(x_max, y_max, z_min),(x_max, y_max, z_max),(x_min, y_max, z_max))



    bm1 = bmesh.new()
    [bm1.verts.new((v[0], v[1], v[2])) for v in verts1]
    bm1.faces.new(bm1.verts)

    bm1.normal_update()
```

```python

    me1 = bpy.data.meshes.new("")
    bm1.to_mesh(me1)

    plane1 = bpy.data.objects.new("", me1)
    bpy.context.scene.collection.objects.link(plane1)
    plane1.select_set(state=True)
    bpy.context.view_layer.objects.active = plane1

    bpy.ops.rigidbody.objects_add(type='PASSIVE')



    bm2 = bmesh.new()
    [bm2.verts.new((v[0], v[1], v[2])) for v in verts2]
    bm2.faces.new(bm2.verts)

    bm2.normal_update()

    me2 = bpy.data.meshes.new("")
    bm2.to_mesh(me2)

    plane2 = bpy.data.objects.new("", me2)

    bpy.context.scene.collection.objects.link(plane2)
    plane2.select_set(state=True)
    bpy.context.view_layer.objects.active = plane2

    bpy.ops.rigidbody.objects_add(type='PASSIVE')

    bm3 = bmesh.new()
    [bm3.verts.new((v[0], v[1], v[2])) for v in verts3]
    bm3.faces.new(bm3.verts)

    bm3.normal_update()

    me3 = bpy.data.meshes.new("")
    bm3.to_mesh(me3)

    plane3 = bpy.data.objects.new("", me3)

    bpy.context.scene.collection.objects.link(plane3)
    plane3.select_set(state=True)
    bpy.context.view_layer.objects.active = plane3

    bpy.ops.rigidbody.objects_add(type='PASSIVE')

# Cll the function
createPlane(cube)
```

Listing A.2: Generate surfaces and their rigid body

```
# Apply Boolean Cut w.r.t position of Fibers
# Import Python Libraries

import bpy
import mathutils
from mathutils import Vector
from mathutils import geometry
from mathutils.bvhtree import BVHTree
import math
from scipy.stats import vonmises
import random
import numpy as np
import bmesh

# Take the original cube
cube = bpy.ops.mesh.primitive_cube_add(size=1)
# Transform it
bpy.ops.transform.translate(value=(-0, -0, -0.5), orient_type='
    GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)),
    orient_matrix_type='GLOBAL', constraint_axis=(False, False,
    True), mirror=True, use_proportional_edit=False,
    proportional_edit_falloff='SMOOTH', proportional_size=1,
    use_proportional_connected=False, use_proportional_projected
    =False, release_confirm=True)
cube=bpy.context.object
cube.name = "Domain"
cube.display_type = 'BOUNDS'
# Mesh the objects
cubeMesh = bmesh.new()
cubeMesh.from_mesh(bpy.context.object.data)
volDomain = cubeMesh.calc_volume(signed = False)
cubeMesh.free()

# Make a list of objects which are fibers
fibers = [obj for obj in bpy.data.objects if obj.name.
    startswith('NurbsPath')]
# Apply Boolean Cut
for fiber in fibers:

  bpy.context.view_layer.objects.active = fiber
  fiber = bpy.context.object

  bpy.ops.object.modifier_add(type = 'BOOLEAN')
  bpy.context.object.modifiers["Boolean"].object = bpy.data.
    objects["Domain"]
  bpy.context.object.modifiers["Boolean"].operation = '
    INTERSECT'
  bpy.ops.object.modifier_apply(apply_as='DATA', modifier="
    Boolean")

#  To calculate the volume ratio at the end
# global volFiber
```

```

# global tries
# global alpha
# volFiber = 0.0
  #alpha =0.0
  #fiberMesh = bmesh.new()
  #fiberMesh.from_mesh(bpy.context.object.data)#Initialize this
     bmesh from existing mesh datablock.
    #mesh (Mesh)     The mesh data to load(bpy.context.object.
   data)
  #volFiber= volFiber + fiberMesh.calc_volume(signed = False) #
   if it is true it willl retrn negative value
  #fiberMesh.free()#Explicitly free the BMesh data from memory,
     causing exceptions on further access.
  #alpha = volFiber/volDomain
  #print("Packing Density : {}".format(alpha))
    #logging.debug("Packing Density : {}".format(alpha))
  #volSpace = volDomain-(alpha *volDomain)
  #print("Volume Space : {}".format(volSpace))
```

Listing A.3: Transfer the domain and apply Boolean Cut