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

###
### This file is generated automatically by SALOME v8.3.0 with dump python functionality
###

import sys
sys.path.append("C:\Software\Python27\Lib\site-packages")
import salome
import numpy as np

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.NoteBook(theStudy)
sys.path.insert(0, r'C:/Users/nmaftoon/Desktop')

###
### GEOM component
###

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

geompy = geomBuilder.New(theStudy)

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

geompy.addToStudy( O, 'O' )
geompy.addToStudy( OX, 'OX' )
geompy.addToStudy( OY, 'OY' )
geompy.addToStudy( OZ, 'OZ' )
#-----------------------------------------------------------------------------------------------------
#Sphere development

Mesh_1_stl_1 = geompy.ImportSTL("C:/Users/nbdaw/Desktop/School/Research Assistant/Salome 8 code/Mesh_1.stl" )
Translation_1 = geompy.MakeTranslation(Mesh_1_stl_1, 0, 0, 0)

#geompy.addToStudy( Mesh_1_stl_1, 'Mesh_1.stl_1' )
geompy.addToStudy( Mesh_1_stl_1, 'Sphere' )

#Extract all faces of a sphere
Face_array =
[Face_1,Face_2,Face_3,Face_4,Face_5,Face_6,Face_7,Face_8,Face_9,Face_10,Face_11,Face_12,Face_13,Face_14,Face_15,Face_16,Face_17,Face_18,Face_19
 = geompy.ExtractShapes(Translation_1, geompy.ShapeType["FACE"], True)

#Extract all vertexes of a sphere
Vertex_array =
[Vertex_1,Vertex_2,Vertex_3,Vertex_4,Vertex_5,Vertex_6,Vertex_7,Vertex_8,Vertex_9,Vertex_10,Vertex_11,Vertex_12,Vertex_13,Vertex_14,Vertex_15,Ve
 = geompy.ExtractShapes(Translation_1, geompy.ShapeType["VERTEX"], True)

#Create empty lists of area, normal vecotr of faces and center of mass
Area = []
Normal_vector = []
cm = []

#Extract the values from each face for lists of area, normal vector and center of mass.
for i in xrange(len(Face_array)):
  Point_1 = geompy.MakeCDG(Face_array[i])
  Vector_Normal_1 = geompy.GetNormal(Face_array[i],Point_1)
  length, area, volume = geompy.BasicProperties(Face_array[i])
  Area.append(area)

  Normal_vector.append(tuple(np.multiply(geompy.VectorCoordinates(Vector_Normal_1),(-1,-1,-1))))
  cm.append(geompy.PointCoordinates(Point_1))

  #geompy.addToStudyInFather( Translation_1, Face_array[i], 'Face_' + str(i+1))
  #geompy.addToStudy( Point_1, 'Point_' + str(i+1))
  #geompy.addToStudy( Vector_Normal_1, 'Vector_Normal_' + str(i+1))

#for i in xrange(len(Vertex_array)):
  #geompy.addToStudyInFather( Translation_1, Vertex_array[i], 'Vertex_' + str(i))

#-----------------------------------------------------------------------------------------------------
#Sketch development

#Funcation is recursive.The number of branches in each level will be 2 times more than the preivous level.
#For example, 2,4,8....
```

```
def f(d0,x,y,z,count,collection,temp_x,temp_y,temp_z,Xi,Xc,times,Sp_step,Xc_step,Xc_i):

    #This is the criteria to stop the alogorithm and to generate the results
    if d0 < 2:
        return d0

    # Normal-like distribution with mu = 2.8 & std = 1.0 for the ratio of length to diameter
    mu1 = 350
    sigma1 = 20
    ini_ratio = np.random.normal(mu1, sigma1, count) # The ratio of diameter to length

    # Please see the literature paper and notes for the defined equations
    d1 = (r)**(1/n)*d0
    d2 = d0*(1-r)**(1/n)
    theta1 = np.arccos((1+r**(4/n)-(1-r)**(4/n))/(2*r**(2/n)))*180./np.pi
    theta2 = np.arccos((1+(1-r)**(4/n)-r**(4/n))/(2*(1-r)**(2/n)))*180./np.pi

    #Angle development
    #Sign list includes the variety of angle types: positive or negative
    #Combinaiton list is temperoary list to store all combinations of angle types
    #Collection list is the final list which will be assigned to Branch development
    sign = [theta1, -theta2]
    combination = []
    for i in xrange(count):
        if not collection or len(collection) == 1:  #Base case for level = 2
            collection.append(sign[i])
        else:
            for j in xrange(len(collection)):
                for k in xrange(len(sign)):
                    combination.append(collection[j]+sign[k])
            collection = combination[:] #Collection contains all the combination of angles
            break

    #Branch development
    '''
    For variables of Si, Sp, Vb, Vd, Nb, please review the literature paper "Development of a model of the coronary arterial
    tree for the 4D XCAT phantom".
    '''
    Si = 0
    Sp = 0
    count_l = 0
    #**************************Calculation**************************
    for i in xrange(count):
        #The length for each branch is following the rule of normal distribution, which is random
        ran_ratio = random.choice(ini_ratio)
        l1 = ran_ratio*d1
        l2 = ran_ratio*d2
        if count_l < 1:
            L = l1
            count_l += 1
        else:
            L = l2
            count_1 = 0

        #Self-avoidance algorithm Vs
        Vs = 0
        if Xc_step < 2:
            Xc_step += 1
        else:
            Xc_step = 1
            Xc_i += 1

        #Sp define
        if Sp_step < 2:
            Sp = tuple(np.subtract(Xc[Xc_i], Xi[times]))
            Sp_step += 1
        else:
            times += 1
            Sp = tuple(np.subtract(Xc[Xc_i], Xi[times]))
            Sp_step = -1

        #Si calculation
        for j in xrange(len(Xi)):
            Si = tuple(np.subtract(Xc[Xc_i], Xi[j]))
            if Si == (0,0,0):
                continue
            Si_mag = np.linalg.norm(Si,ord=2)
            Eq = ((L/Si_mag)**zeta/(1+(L/Si_mag)**zeta))*(Si/Si_mag)
            Vs = tuple(np.add(Vs,Eq))

        #Boundary-avoidance algorithm Vb
        Vb = 0
        for k in xrange(len(Area)):
            Expon = tuple(np.subtract(Xc[Xc_i], cm[k]))
            Expon_mag = np.linalg.norm(Expon,ord=2)
            Eq_2 = tuple(np.multiply(Area[k]*np.exp(-Expon_mag/(2*L)),Normal_vector[k]))
```

```
            Vb = tuple(np.add(Vb,Eq_2))

        #Combination Vd
        Vd = 0
        Cs = 0.5
        Cb = 0.5
        Vs_1 = tuple((Vs/np.linalg.norm(Vs,ord=2))*Cs)
        Vb_1 = tuple((Vb/np.linalg.norm(Vb,ord=2))*Cb)
        Vd = tuple(np.add(Vs_1,Vb_1))

        #Nomral Vector nb: (sp x vd) x vd
        Nb_1 = tuple(np.cross(Sp,Vd))
        Nb = tuple(np.cross(Nb_1, Vd))

        #**********************SALOME SKETCH DEVELOPMENT********************************

        #Create a Plane
        Plane_vertex = geompy.MakeVertex(x[i/2], y[i/2],z[i/2])
        Plane_vector = geompy.MakeVectorDXDYDZ(Nb[0],Nb[1],Nb[2])
        Branching_Plane = geompy.MakePlane(Plane_vertex, Plane_vector, 200)
        geompy.addToStudy( Plane_vertex, 'Plane_vertex' )
        geompy.addToStudy( Plane_vector, 'Plane_vector' )
        geompy.addToStudy( Branching_Plane, 'Group_' + str(count/2) + '_Branching Plane')

        #Create a Sketch
        sk = geompy.Sketcher2D()
        sk.addPoint(0.000000, 0.000000)
        sk.addSegmentAngleLength(collection[i], L)
        Sketch_2 = sk.wire(Branching_Plane) #geomObj_1
        [Vertex_1,Vertex_2] = geompy.ExtractShapes(Sketch_2, geompy.ShapeType["VERTEX"], False)
        geompy.addToStudy(Sketch_2, 'Group_' + str(count/2) + '_Sketch')
        #geompy.addToStudyInFather( Sketch_2, Vertex_2, 'Vertex_2' )

        coords = geompy.PointCoordinates(Vertex_2) #Only use the coordinate of end point of each branch
        Xi.append(coords) #Software always append the coordinate of end point of left branching first, then right branching
        Xc.append(coords) #End point of each branch is also the branching point for the next round of branching
        temp_x.append(coords[0]) #Assign x coordinate of end point vertex
        temp_y.append(coords[1])
        temp_z.append(coords[2])

    x = temp_x[:] #Transfer all stored x coordinate for each burfication point to a permanent one
    y = temp_y[:]
    z = temp_z[:]
    temp_x = [] #Clearn the temperoary x list and prepare for the next round
    temp_y = []
    temp_z = []
    count = count * 2 #Ensure the number of branching in next round is 2 times more.
    return f(d1,x,y,z,count,collection,temp_x,temp_y,temp_z,Xi,Xc,times,Sp_step,Xc_step,Xc_i)
#----------------------------------------------------------------------------------------------------

#Reivew the paper "Kitaoka et al. - 1999 - A three-dimensional model of the human airway tree" for definition
n = 3.  #Murray's law n factor for vessel diameter equation
r = 0.5  #Flow-dividing ratio
k = 60. #Vessel length k parameter
d0 = 4. #Root vessel diameter

# Normal-like distribution with mu = 2.8 & std = 1.0 for the ratio of length to diameter
mu = 350
sigma = 1
ini_ratio = np.random.normal(mu, sigma, 1) # initial_length_diameter_ratio
l0 = ini_ratio*d0

geomObj_1 = geompy.MakeMarker(0, 0, 0, 1, 0, 0, 0, 1, 0)
sk = geompy.Sketcher2D()
sk.addPoint(0.000000, 0.000000)
sk.addSegmentAngleLength(0, l0)
Sketch_1 = sk.wire(geomObj_1)

[Vertex_1,Vertex_2] = geompy.ExtractShapes(Sketch_1, geompy.ShapeType["VERTEX"], True)
geompy.addToStudy( Sketch_1, 'Initial_vessel')
geompy.addToStudyInFather( Sketch_1, Vertex_1, 'Vertex_1' )
geompy.addToStudyInFather( Sketch_1, Vertex_2, 'Vertex_2' )

coords_1 = geompy.PointCoordinates(Vertex_1) #Store the coordinates of the start point of a vextex for parent branching
coords = geompy.PointCoordinates(Vertex_2) #Store the coordinates of the end point of a vextex for parent branching

zeta = 2 #See literature paper "Development of a model of the coronary arterial tree for the 4D XCAT phantom" for definition

Xi = []
Xi.append(coords_1)
Xi.append(coords)

Xc =[] #Burfication point
Xc.append(coords)

x = []
```

```
y = []
z = []

temp_x = [] # Create an empty list for storing the x coordinates in each branch level
temp_y = []
temp_z = []

x.append(coords[0])
y.append(coords[1])
z.append(coords[2])

collection = [] # This list is to contain all the branching angles in each branch level

f(d0,x,y,z,2,collection,temp_x,temp_y,temp_z,Xi,Xc,0,0,0,0)

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