import stl

from stl import mesh

import numpy as np

import math

from pfea.geom import dschwarz

# Mapping between node sides in the base node

# and the transformed. Ensures that there are no

# holes in the final mesh

siderefnode0 = [[( 0, 0, 0),2],

[( 0, 0, 0),1],

[(-1, 0, 0),3],

[(-1, 0,-1),2],

[( 0,-1,-1),1],

[( 0,-1, 0),3]]

siderefnode1 = [[( 0, 1, 1),0],

[(-1, 0, 1),3],

[(-1, 0, 0),2],

[( 0, 0, 0),0],

[( 0, 0, 0),3],

[( 0, 1, 0),2]]

siderefnode2 = [[( 0,-1, 1),3],

[( 1, 0, 1),0],

[( 1, 0, 0),1],

[( 0, 0, 0),3],

[( 0, 0, 0),0],

[( 0,-1, 0),1]]

siderefnode3 = [[( 0, 0, 0),1],

[( 0, 0, 0),2],

[( 1, 0, 0),0],

[( 1, 0,-1),1],

[( 0, 1,-1),2],

[( 0, 1, 0),0]]

# Beam reference for each node in the unit cell

# in order to ensure that there are no hanging beams

beamrefnode0 = [[( 0, 0, 0),1],

[( 0,-1, 0),3],

[(-1, 0,-1),2],

[( 0, 0, 0),2],

[(-1, 0, 0),3],

[( 0,-1,-1),1]]

beamrefnode1 = [[(-1, 0, 1),3],

[( 0, 0, 0),-1],

[( 0, 0, 0),-1],

[( 0, 0, 0),-1],

[(-1, 0, 0),2],

[( 0, 0, 0),3]]

beamrefnode2 = [[( 0, 0, 0),-1],

[( 0,-1, 0),1],

[( 0, 0, 0),3],

[( 0,-1, 1),3],

[( 0, 0, 0),-1],

[( 0, 0, 0),-1]]

rot\_60\_tform = np.array([[0.5, -0.5*math.sqrt(3), 0.0],*

*[0.5*math.sqrt(3), 0.5, 0.0],

[0.0, 0.0, 1.0]])

def scale(meshobj,factor):

vects = meshobj.vectors

scalemat = np.diag(np.array(factor))

for vsi,vectset in enumerate(vects):

for vi,vector in enumerate(vectset):

meshobj.vectors[vsi][vi] = np.dot(vector,scalemat)

def translate(meshobj,tvect):

vects = meshobj.vectors

for vsi,vectset in enumerate(vects):

for vi,vector in enumerate(vectset):

meshobj.vectors[vsi][vi] = vector+tvect

def rev\_z(meshobj):

for vectset in meshobj.vectors:

tval = vectset[1].copy()

vectset[1] = vectset[2].copy()

vectset[2] = tval

# We should have a matrix that tells us whether a voxel has material, and

# a matrix that tells us what nodes in that voxel are present.

def nodeneighbors(invol, cl):

#node 1

node0sides = np.zeros(6)

node1sides = np.zeros(6)

node2sides = np.zeros(6)

node3sides = np.zeros(6)

for i in range(6):  
 node0sides[i] = -1\*(invol[cl[0]+siderefnode0[i][0][0]][cl[1]+siderefnode0[i][0][1]][cl[2]+siderefnode0[i][0][2]][siderefnode0[i][1]]-1)  
 node1sides[i] = -1\*(invol[cl[0]+siderefnode1[i][0][0]][cl[1]+siderefnode1[i][0][1]][cl[2]+siderefnode1[i][0][2]][siderefnode1[i][1]]-1)  
 node2sides[i] = -1\*(invol[cl[0]+siderefnode2[i][0][0]][cl[1]+siderefnode2[i][0][1]][cl[2]+siderefnode2[i][0][2]][siderefnode2[i][1]]-1)  
 node3sides[i] = -1\*(invol[cl[0]+siderefnode3[i][0][0]][cl[1]+siderefnode3[i][0][1]][cl[2]+siderefnode3[i][0][2]][siderefnode3[i][1]]-1)  
  
return np.vstack((node0sides,node1sides,node2sides,node3sides))

def beamneighbors(invol, cl):

beam0ref = np.zeros(6)

beam1ref = np.zeros(6)

beam2ref = np.zeros(6)

for i in range(6):  
 if invol[cl[0]][cl[1]][cl[2]][0] != 0:  
 beam0ref[i] = invol[cl[0]+beamrefnode0[i][0][0]][cl[1]+beamrefnode0[i][0][1]][cl[2]+beamrefnode0[i][0][2]][beamrefnode0[i][1]]  
 if beamrefnode1[i][1] != -1 and invol[cl[0]][cl[1]][cl[2]][1] != 0:  
 beam1ref[i] = invol[cl[0]+beamrefnode1[i][0][0]][cl[1]+beamrefnode1[i][0][1]][cl[2]+beamrefnode1[i][0][2]][beamrefnode1[i][1]]  
 if beamrefnode2[i][1] != -1 and invol[cl[0]][cl[1]][cl[2]][2] != 0:  
 beam2ref[i] = invol[cl[0]+beamrefnode2[i][0][0]][cl[1]+beamrefnode2[i][0][1]][cl[2]+beamrefnode2[i][0][2]][beamrefnode2[i][1]]  
  
return np.vstack((beam0ref,beam1ref,beam2ref))

def node(beam\_width, chamfer\_factor, side\_code):

topcap = np.zeros(12,dtype=mesh.Mesh.dtype)

side = np.zeros(12,dtype=mesh.Mesh.dtype)

chamside = np.zeros(12,dtype=mesh.Mesh.dtype)

center = np.array([0.0, 0.0, beam\_width/2.0])  
  
topbound = np.array([[ 0.5\*chamfer\_factor\*beam\_width, beam\_width+0.5\*math.sqrt(3)\*chamfer\_factor\*beam\_width, beam\_width/2.0],  
 [-0.5\*chamfer\_factor\*beam\_width, beam\_width+0.5\*math.sqrt(3)\*chamfer\_factor\*beam\_width, beam\_width/2.0]])  
for i in range(10):  
 vec = topbound[-2]  
 topbound = np.vstack((topbound,np.dot(rot\_60\_tform,vec)))  
  
botbound = np.copy(topbound)  
botbound.T[2] = -botbound.T[2]  
  
for i in range(12):  
 topcap['vectors'][i] = np.vstack((topbound[i],center,topbound[i-1]))  
  
botcap = mesh.Mesh(topcap.copy())  
botcap.rotate([0.0,1.0,0.0],math.radians(180))  
  
  
for i in range(0,12,2):  
 if side\_code[i/2] == 1:  
 side['vectors'][i] = np.vstack((botbound[i-1],botbound[i],topbound[i]))  
 side['vectors'][i+1]= np.vstack((topbound[i],topbound[i-1],botbound[i-1]))  
  
for i in range(-1,11,2):  
 chamside['vectors'][i] = np.vstack((botbound[i-1],botbound[i],topbound[i]))  
 chamside['vectors'][i+1]= np.vstack((topbound[i],topbound[i-1],botbound[i-1]))  
  
nodecapbase = mesh.Mesh(np.concatenate([  
 topcap.copy(),  
 botcap.data.copy(),  
 side.copy(),  
 chamside.copy()  
]))  
  
  
nodecapbase.rotate([0.0,0.0,1.0],math.radians(45))  
  
return nodecapbase

def strut(strut\_width,lattice\_pitch,cham\_factor):

data = np.zeros(2,dtype=mesh.Mesh.dtype)

#Top of the strut  
data['vectors'][0] = np.array([[ 0.5 , 0, 0.5 ],  
 [-0.638071, 1.0, 0.304638],  
 [ 0.304738, 1.0, 0.638071]])  
data['vectors'][1] = np.array([[ 0.5 , 0, 0.5 ],  
 [-0.5 , 0, 0.5 ],  
 [-0.638071, 1.0, 0.304638]])  
  
strut = mesh.Mesh(data.copy())  
  
for i in range(3):  
 tmpside = mesh.Mesh(data.copy())  
 tmpside.rotate([0.0,1.0,0.0],math.radians(90\*(i+1)))  
  
 strut = mesh.Mesh(np.concatenate([  
 strut.data,  
 tmpside.data.copy()  
 ]))  
  
node\_width = (0.5\*math.sqrt(3)+cham\_factor)\*strut\_width  
scale(strut,[strut\_width,lattice\_pitch-2\*node\_width,strut\_width])  
translate(strut,(0,node\_width,0))  
  
return strut

def find\_pitch(l,rd, cf, sw, bsv):

return (6*(4*cf+np.sqrt(3)*cf*cf+np.sqrt(3))-12*bsv*(0.5*np.sqrt(3)+cf))*sw*sw*sw + sw*sw*l - rd*2*np.sqrt(2)*l*l\*l

def pitch\_from\_relden(relden, cf, sw):

bsv = 0.869825 #mm^3

c1 = relden*np.sqrt(2.0)*2.0

c2 = -(12*bsv)*sw*sw*

*c3 = -(6*(4*cf+np.sqrt(3)*cf*cf+np.sqrt(3))-12*bsv*(np.sqrt(3)+2.0*cf))*sw*sw\*sw

return max(np.roots([c1,0,c2,c3]))

strut\_width = 0.62 #mm

min\_width = strut\_width

strut\_width = strut\_width/0.8167#/0.6504

cham\_factor = 0.3

relative\_density = 0.01

lattice\_pitch = pitch\_from\_relden(relative\_density,cham\_factor,strut\_width)

unit\_cell = lattice\_pitch*np.sqrt(2)*

*thresh = 0.01*

*size = 12*

*print(unit\_cell*(size-2)/25.4)

mat\_matrix = np.zeros((3,3,3))

mat\_matrix[1][1][1] = 1

invol = np.zeros((size,size,size,4))#dschwarz.gen\_111\_invol(6,32,lattice\_pitch)#np.zeros((size,size,size,4))

meshes = []

for x in range(1,len(invol)-1):

for y in range(1,len(invol[0])-1):

for z in range(1,len(invol[0][0])-1):

if x == 1 and y == 1:

invol[x][y][z] = [0,1,1,1]

elif x == len(invol)-2 and y == 1:

invol[x][y][z] = [1,1,0,1]

elif x == 1 and y == len(invol[0])-2:

invol[x][y][z] = [1,0,1,1]

elif x == len(invol)-2 and y == len(invol[0])-2:

invol[x][y][z] = [1,1,1,0]

else:

invol[x][y][z] = [1,1,1,1]

for x in range(1,len(invol)-1):

for y in range(1,len(invol[0])-1):

for z in range(1,len(invol[0][0])-1):

sides = nodeneighbors(invol,(x,y,z))

strutref = beamneighbors(invol,(x,y,z))

struts = []  
  
 struts.append(strut(strut\_width,lattice\_pitch,cham\_factor))  
 struts.append(mesh.Mesh(struts[0].data.copy()))  
 struts.append(mesh.Mesh(struts[0].data.copy()))  
  
 struts[1].rotate([0.0,0.0,1.0],math.radians(120))  
 struts[2].rotate([0.0,0.0,1.0],math.radians(240))  
  
 struts.append(mesh.Mesh(struts[0].data.copy()))  
 struts.append(mesh.Mesh(struts[1].data.copy()))  
 struts.append(mesh.Mesh(struts[2].data.copy()))  
  
 for i in range(6):  
 struts[i].rotate([0.0,0.0,1.0],math.radians(15))  
  
 for i in range(3,6):  
 tval = struts[i].vectors.T[0].copy()  
 struts[i].vectors.T[0] = struts[i].vectors.T[1].copy()  
 struts[i].vectors.T[1] = tval  
  
 nodes = [node(strut\_width,cham\_factor,sides[0]),  
 node(strut\_width,cham\_factor,sides[1]),  
 node(strut\_width,cham\_factor,sides[2]),  
 node(strut\_width,cham\_factor,sides[3])]  
  
 ucmeshes = [[],[],[],[]]  
  
 for i in range(4):  
 if invol[x][y][z][i] == 1:  
 ucmeshes[i].append(nodes[i])  
  
  
 for i in range(3):  
 for j in range(6):  
 if strutref[i][j] == 1:  
 ucmeshes[i].append(mesh.Mesh(struts[j].data.copy()))  
  
  
 for tmesh in ucmeshes[0]:  
 tmesh.rotate([1.0,-1.0,0.0],-math.atan(math.sqrt(2)))  
 translate(tmesh,np.array([x+0.25,y+0.25,z+0.25])\*unit\_cell)  
  
 for tmesh in ucmeshes[1]:  
 tmesh.rotate([1.0,-1.0,0.0],math.atan(-math.sqrt(2)))  
 tmesh.rotate([0.0,0.0,1.0],math.radians(-90))  
 translate(tmesh,np.array([x+0.25,y+0.75,z+0.75])\*unit\_cell)  
  
 for tmesh in ucmeshes[2]:  
 tmesh.rotate([1.0,-1.0,0.0],math.atan(-math.sqrt(2)))  
 tmesh.rotate([0.0,0.0,1.0],math.radians(90))  
 translate(tmesh,np.array([x+0.75,y+0.25,z+0.75])\*unit\_cell)  
  
 for tmesh in ucmeshes[3]:  
 tmesh.rotate([1.0,-1.0,0.0],-math.atan(math.sqrt(2)))  
 tmesh.rotate([0.0,0.0,1.0],math.radians(180))  
 translate(tmesh,np.array([x+0.75,y+0.75,z+0.25])\*unit\_cell)  
  
 meshes.append(ucmeshes)

latticedata = []#np.zeros(1,dtype=mesh.Mesh.dtype)

for ucmeshes in meshes:

for meshset in ucmeshes:

for mesh in meshset:

latticedata.append(mesh.data.copy())

latticedata = np.concatenate(latticedata)

latticedata = stl.mesh.Mesh(latticedata)

# Perform a rotation to orient it so that (111) aligns with (001)

# latticedata.rotate([0.0,0.0,1.0],math.radians(45))

# latticedata.rotate([0.0,1.0,0.0],math.atan(np.sqrt(2)))

# align\_points(latticedata,thresh)

latticedata.save(‘{0}*{0}dsch*rd\_{2}sw.stl’.format(size-2,relative\_density,min\_width))

’‘’

# Optionally render the rotated cube faces

from matplotlib import pyplot

from mpl\_toolkits import mplot3d

# Create a new plot

figure = pyplot.figure()

axes = mplot3d.Axes3D(figure)

axes.set\_xlim(unit\_cell,3*unit\_cell)*

*axes.set\_ylim(unit\_cell,3*unit\_cell)

axes.set\_zlim(unit\_cell,3\*unit\_cell)

# for i in range(6):

#axes.add\_collection3d(mplot3d.art3d.Poly3DCollection(struts[i].vectors))

# Render the cube

for ucmeshes in meshes:

for meshset in ucmeshes:

for mesh in meshset:

axes.add\_collection3d(mplot3d.art3d.Poly3DCollection(mesh.vectors))

axes.add\_collection3d(mplot3d.art3d.Poly3DCollection(latticedata.vectors))

# Show the plot to the screen

pyplot.show()

‘’‘