Politecnico di Torino

Creation of foam-like geometries using PlugIm

Torino, 12 Marzo 2023
Alessio Bocca

GitHub

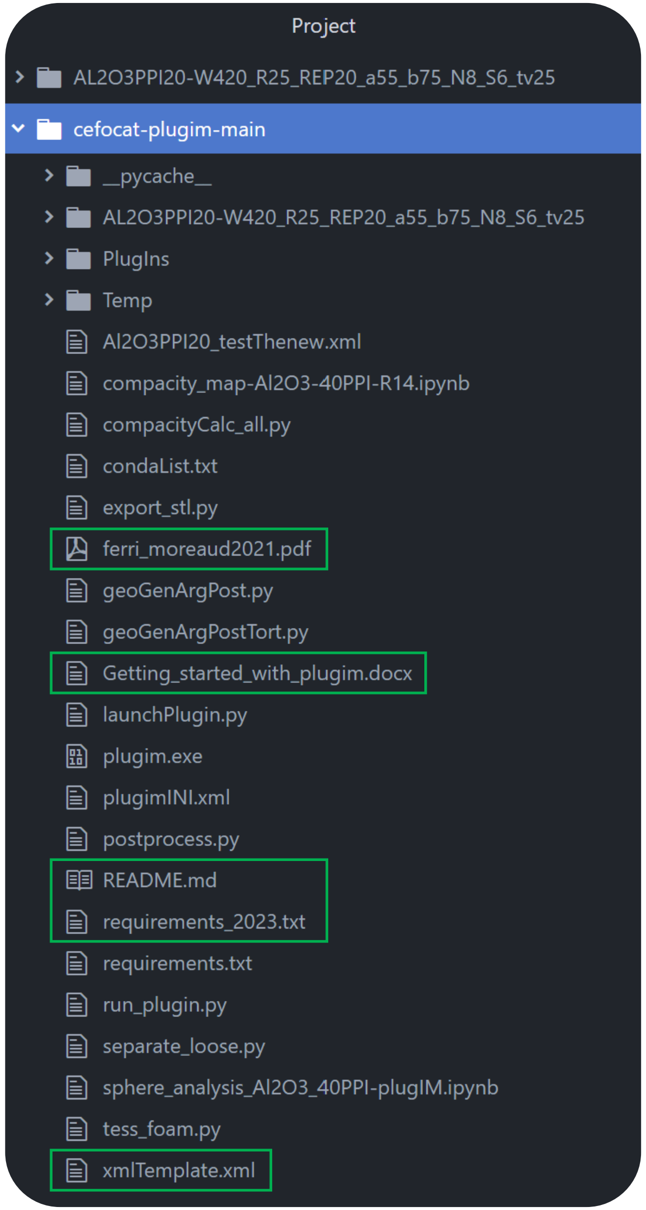
cefocat-plugim-main

The main folder, downloaded from GitHub, already contains all python scripts, the plugIm software with its plug-ins and those files useful for a quick installation of python and its libraries required to use this workflow.

To install python and its libraries it is recommended to create a new virtual environment using anaconda.

In order to build some wheels it could be necessary to install VS Build Tools from here.

All information about installation are available in the `README.md` file.



How this document is organized

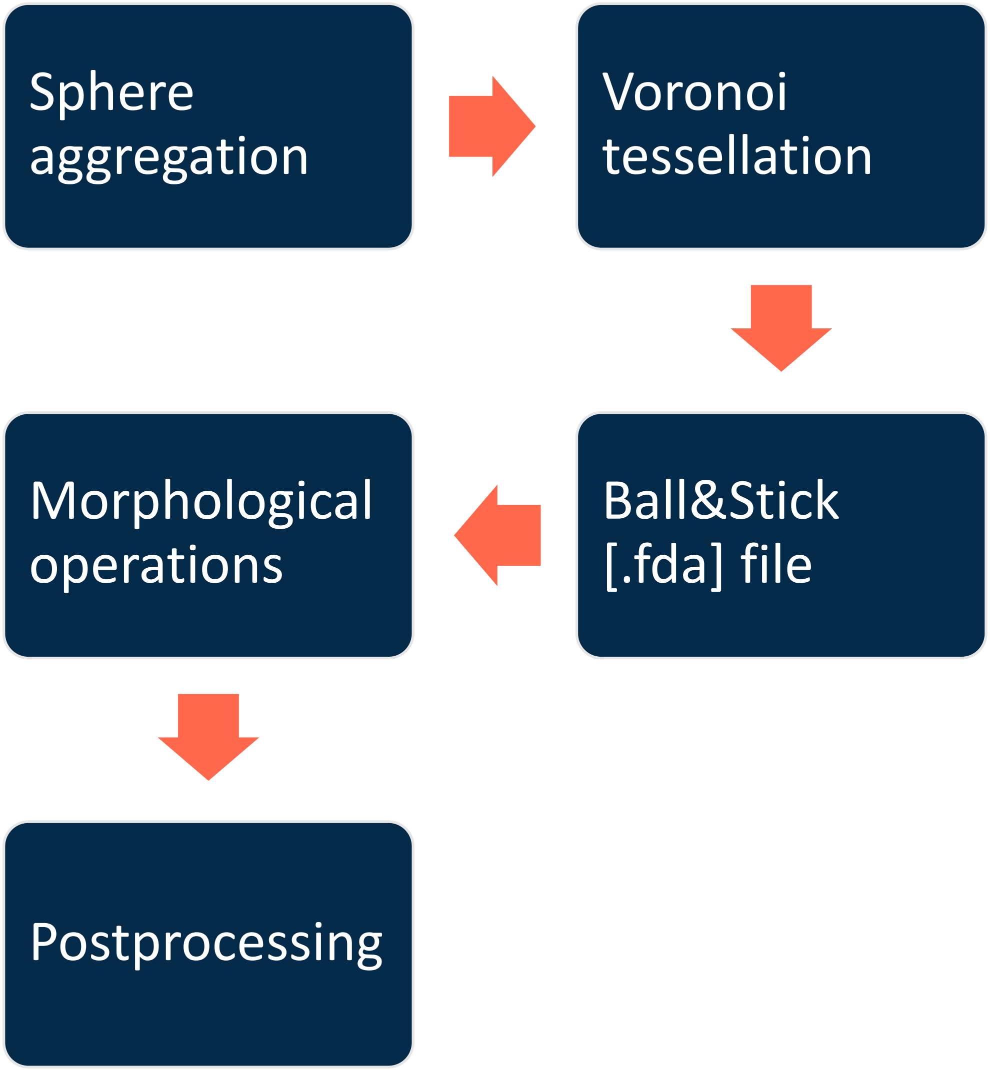
- First of all it will be briefly describe the whole workflow, introducing the role of plug-ins and how they are managed. All user-defined parameters (input) are also described.
- Then, the main script will be covered in all its parts, examining the function of the most important lines of code.
- Each step is now detailed, reporting main files and scripts that are called during execution. Input and output are clearly defined for each step.
- Finally, postprocessing operations are described.

The workflow

A brief introduction

The aim of this workflow is the development and analysis of solid foams. They are generated through several steps:

- Spheres aggregation to obtain spheres packing.
- Voronoi tessellation using as seeds (starting points) the centroids of the spheres.
- Ball and stick plugIm file generation `[.fda]`.
- Morphological operation, such as closing, with a defined size ***tv***.
- Postprocessing operations and computations.

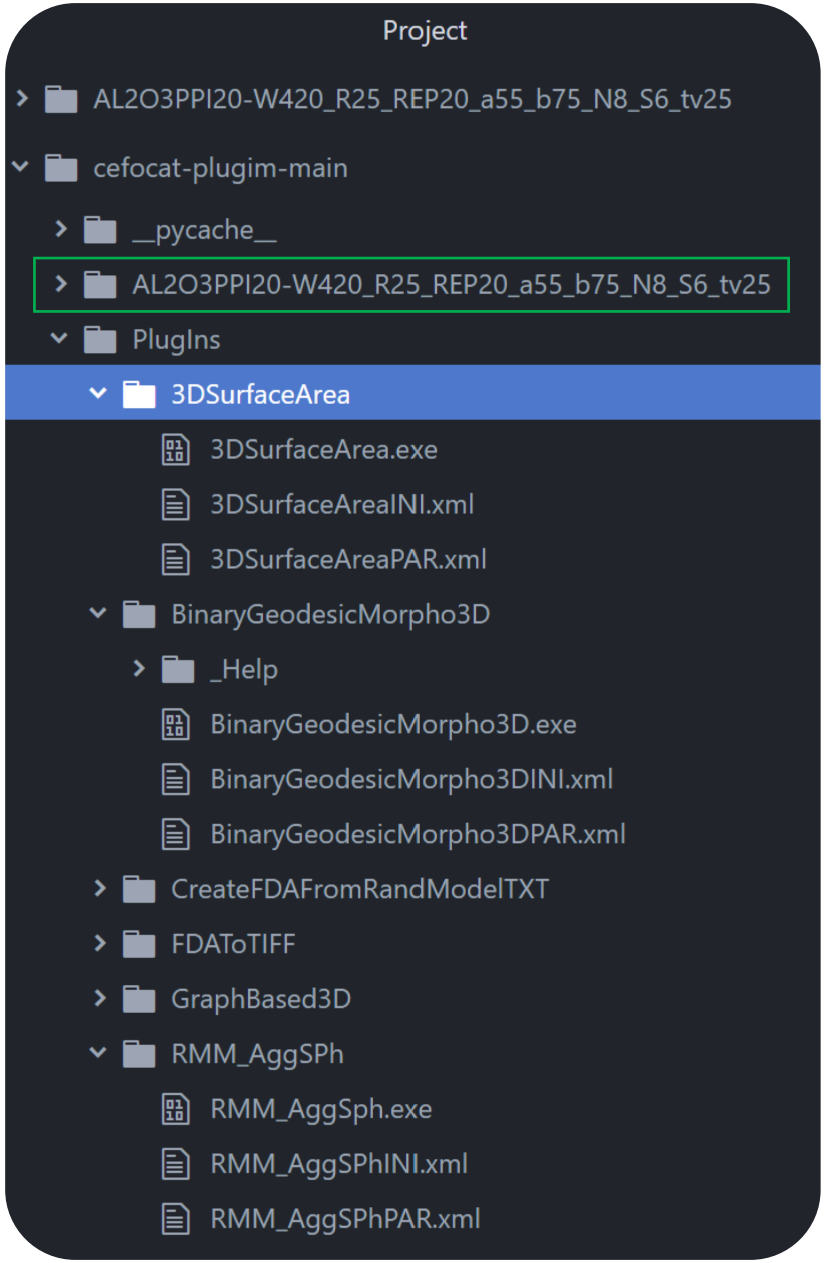


The workflow

Plug-ins role and test.xml file

Each of the previous steps is performed through the use of an appropriate software plug-in. They are called in the main code using the `run_plugim.py` script, which makes a copy of the executable and PAR files inside the working directory.

The required parameters are changed respecting what is defined in the `test.xml` file. It will be the only document to be edited by entering the values of the parameters that govern the development of the foam.



The workflow

Input parameters

- ***pOut***: name of output file.
- ***pIn***: name of input file.
- ***NB***: number of spheres to be aggregated.
- ***Alpha and Beta***: compactness factors.
- ***Repul***: repulsion.
- ***R***: spheres diameter.
- ***W***: box size in integers.
- ***R_node***: diameter of spheres that substitute nodes in tessellation step.
- ***R_strut***: diameter of cylinders that substitute edges in tessellation step.
- ***Iter***: number of iterations (Voronoi tessellation).
- ***Operation***: morphological operation (closing).
- ***tv***: voxel size for chosen morphological operation.
- ***vx_res***: voxel resolution.

geoGenArgPostTort.py | *xmlTemplate.xml* | tess_foam.py

```xml
<?xml version="1.0" ?>
<par>
    <paths>
        <pOut>AL2O3PPI20-W420_R25_REP20_a55_b75_N8_S6_tv25</pOut>
        <pIn>PlugIns</pIn>
    </paths>
    <aggSph>
        <NB>380</NB>
        <Alpha>0.55</Alpha>
        <Beta>0.75</Beta>
        <Repul>20</Repul>
        <R>25</R>
        <W>420</W>
    </aggSph>
    <tess>
        <R_node>8</R_node>
        <R_strut>6</R_strut>
        <iter>1</iter>
    </tess>
    <morph>
        <Operation>Closing</Operation>
        <tv>25</tv>
    </morph>
    <postProc>
        <vx_res>50e-06</vx_res>
    </postProc>
</par>
```

Main script

geoGenArgPostTort.py

First of all, the script acquires and stores all the parameters governing the development of the solid foam in order to be able to provide them to the various plug-ins when they are executed.
Next, the voxel resolution is set and the variables containing the input/output paths are created.

Then, the script generates the folder which will contain the results produced by the workflow and all the files generated by plugIm.

geoGenArgPostTort.py | xmlTemplate.xml | tess_foam.py | ballSticks.s

```python
def readxml(xmlfile):
    with open(xmlfile,'r') as f:
        parxml = f.read()
        parameters = xmltodict.parse(parxml)
        for elm in parameters['par']:
            parameters['par'][elm] = dict(parameters['par'][elm])
        parameters['par'] = dict(parameters['par'])
        parameters = parameters['par']
    return parameters

parser = argparse.ArgumentParser()

parser.add_argument('--name', type=str, required=True)
args = parser.parse_args()

t0 = time.time()

cwd = Path(os.getcwd())

par = readxml(args.name)

vx_res = float(par['postProc']['vx_res'])#50e-06
print('\nWARNING: voxel resulution is set to',vx_res,', you may need to change it!!\n')

pOut = Path(cwd / par['paths']['pOut'])
pIn = Path( cwd / par['paths']['pIn'])

print('creating folder',pOut)
# os.mkdir(pOut)
try:
    os.makedirs(pOut, exist_ok = True)
    print("Directory '%s' created successfully" %pOut.name)
except OSError as error:
    print("Directory '%s' already exist")
```

Main script

geoGenArgPostTort.py

The script also generates several dictionaries, one for each plug-in, which will be passed as an argument to the `run_plugin.py` function at run time.

- `aggShpDict`
- `nodEdgDict`
- `morphDict`
- `surfaceDict`
- `tiffDict`
- `tortDict`

geoGenArgPostTort.py xmlTemplate.xml tess_foam.py

```python
    print("Directory '%s' created successfully" %pOut.name)
except OSError as error:
    print("Directory '%s' already exist")


aggSphDict = {
        'Export' : str(pOut / 'sphPackCoord.txt'),
        'OUTPREV': str(pOut / 'sphPackPreview.fda'),
        'OUT' : str(pOut / 'sphPackOutput.fda'),
        'NB' : par['aggSph']['NB'],
        'NBType' : 'Constante',
        'Repul' : par['aggSph']['Repul'],
        'R' : par['aggSph']['R'],
        'RType' : 'Constante',
        'Beta' : par['aggSph']['Beta'],
        'Alpha' : par['aggSph']['Alpha'],
        'val' : 255,
        'Periodic' : True,
        'W' : par['aggSph']['W']
        }

nodEdgDict = {
    'FileNameFull': 'ballSticks.sb',
    'FileName': 'ballSticks',
    'IN': str(pOut / 'ballSticks.sb') ,
    'OUT': str(pOut / 'ballSticks.fda'),
    'OUTPREV': str(pOut / 'prevBallSticks.fda')
    }
```

Main script

Sequential execution of plugins

Once the user-specified parameters have been collected and the path variables, the test case folder and dictionaries are generated; the code sequentially recalls each plug-in.

During this operation some arguments are required by `run_plugim.py`:

- Plug-in name
- The dictionary previously defined
- Input path `(pIn)`
- Output path `(pOut)`

geoGenArgPostTort.py | xmlTemplate.xml | tess_foam.py | ba

```python
### Launch sphere packing plugin

aggSph = plm.run_plugin('RMM_AggSPh', aggSphDict, pIn,pOut)
aggSph.copy_exe()
aggSph.edit_xml()
aggSph.run_exe()

### Launch tessellation

bf.bfoam(pOut,par['tess']['R_node'],par['tess']['R_strut'],iters=int(par['tess']['iter']), periodic=False)

### Launch ball&Sticks model creation
ballSticks = plm.run_plugin('CreateFDAFromRandModelTXT', nodEdgDict, pIn, pOut)
ballSticks.copy_exe()
ballSticks.edit_xml()
ballSticks.run_exe()

### Launch morphology operations

morphOps = plm.run_plugin('BinaryGeodesicMorpho3D', morphDict, pIn, pOut)
morphOps.copy_exe()
morphOps.edit_xml()
morphOps.run_exe()

### Launch surface area calculation

surfArea = plm.run_plugin('3DSurfaceArea', surfaceDict, pIn, pOut)
surfArea.copy_exe()
surfArea.edit_xml()
surfArea.run_exe()

### Launch tiff creation

fdaToTIFF = plm.run_plugin('FDAToTIFF', tiffDict, pIn, pOut)
fdaToTIFF.copy_exe()
```

Main script

Sequential execution of plugins

Plug-in execution takes place directly from the test case folder because, as seen before:

1. A copy of both the executable `[.exe]` and the parameters file `[.xml]`, required by the plug-in, is made.
2. Parameters are modified according to what is specified by the user.
3. Executable file is finally launched.

geoGenArgPostTort.py | xmlTemplate.xml | tess_foam.py

```python
### Launch sphere packing plugin

aggSph = plm.run_plugin('RMM_AggSPh', aggSphDict, pIn,pOut)
aggSph.copy_exe()
aggSph.edit_xml()
aggSph.run_exe()

### Launch tessellation

bf.bfoam(pOut,par['tess']['R_node'],par['tess']['R_strut'],iters=int(par['tess']['iter']), periodic=False)

### Launch ball&Sticks model creation
ballSticks = plm.run_plugin('CreateFDAFromRandModelTXT', nodEdgDict, pIn, pOut)
ballSticks.copy_exe()
ballSticks.edit_xml()
ballSticks.run_exe()

### Launch morphology operations

morphOps = plm.run_plugin('BinaryGeodesicMorpho3D', morphDict, pIn, pOut)
morphOps.copy_exe()
morphOps.edit_xml()
morphOps.run_exe()

### Launch surface area calculation

surfArea = plm.run_plugin('3DSurfaceArea', surfaceDict, pIn, pOut)
surfArea.copy_exe()
surfArea.edit_xml()
surfArea.run_exe()

### Launch tiff creation

fdaToTIFF = plm.run_plugin('FDAToTIFF', tiffDict, pIn, pOut)
fdaToTIFF.copy_exe()
```

Discussion of single steps

- First of all a packing of spheres is created according to user-defined parameters, which manage the size of the geometry, the compactness (stochastically) and other characteristics.

- Subsequently the Voronoi tessellation is performed starting from the centroids of the spheres. Instead of nodes and edges, spheres and cyliders are inserted.

- The next step is the application of the morphological closing operation over the ‘tessellation skeleton’, resulting in the final characteristic geometry of foams.

- Finally, calculations and postprocessing operations are performed.

Spheres aggregation

FIRST STEP

Input values are supplied through the aggSphDict dictionary which contains, in a first part, the names of the output files including a text file `[.txt]` which will contain all the coordinates of the centroids of the generated spheres and their radii.

In the second part of the dictionary all the values specified by the user, and previously stored in `par` are defined.

Finally ***W*** is defined, which is the size of the cubic domain.

```
aggSphDict = {
        'Export' : str(pOut / 'sphPackCoord.txt'),
        'OUTPREV': str(pOut / 'sphPackPreview.fda'),
        'OUT' : str(pOut / 'sphPackOutput.fda'),
        'NB' : par['aggSph']['NB'],
        'NBType' : 'Constante',
        'Repul' : par['aggSph']['Repul'],
        'R' : par['aggSph']['R'],
        'RType' : 'Constante',
        'Beta' : par['aggSph']['Beta'],
        'Alpha' : par['aggSph']['Alpha'],
        'val' : 255,
        'Periodic' : True,
        'W' : par['aggSph']['W']
        }
```

Spheres aggregation

FIRST STEP

The output of the spheres aggregation algorithm is a text file `[.txt]` which will be the starting point for all subsequent plug-ins.

The first row contains the dimensions of the cubic domain `(420x420x420)` and the number of generated objects `(380 spheres)`.

From the second line onwards, the spheres are listed, first defining their centroids in a three-dimensional space (x_C, y_C, z_C) and then their radii (r_C).

The last element of each row is a running index.

sphPackCoord.txt

```
420 420 420 380 1
210 210 210 25.00 1
213 262 258 25.00 2
141 197 205 25.00 3
222 276 190 25.00 4
97 252 199 25.00 5
77 169 198 25.00 6
94 303 150 25.00 7
158 184 138 25.00 8
266 210 167 25.00 9
```

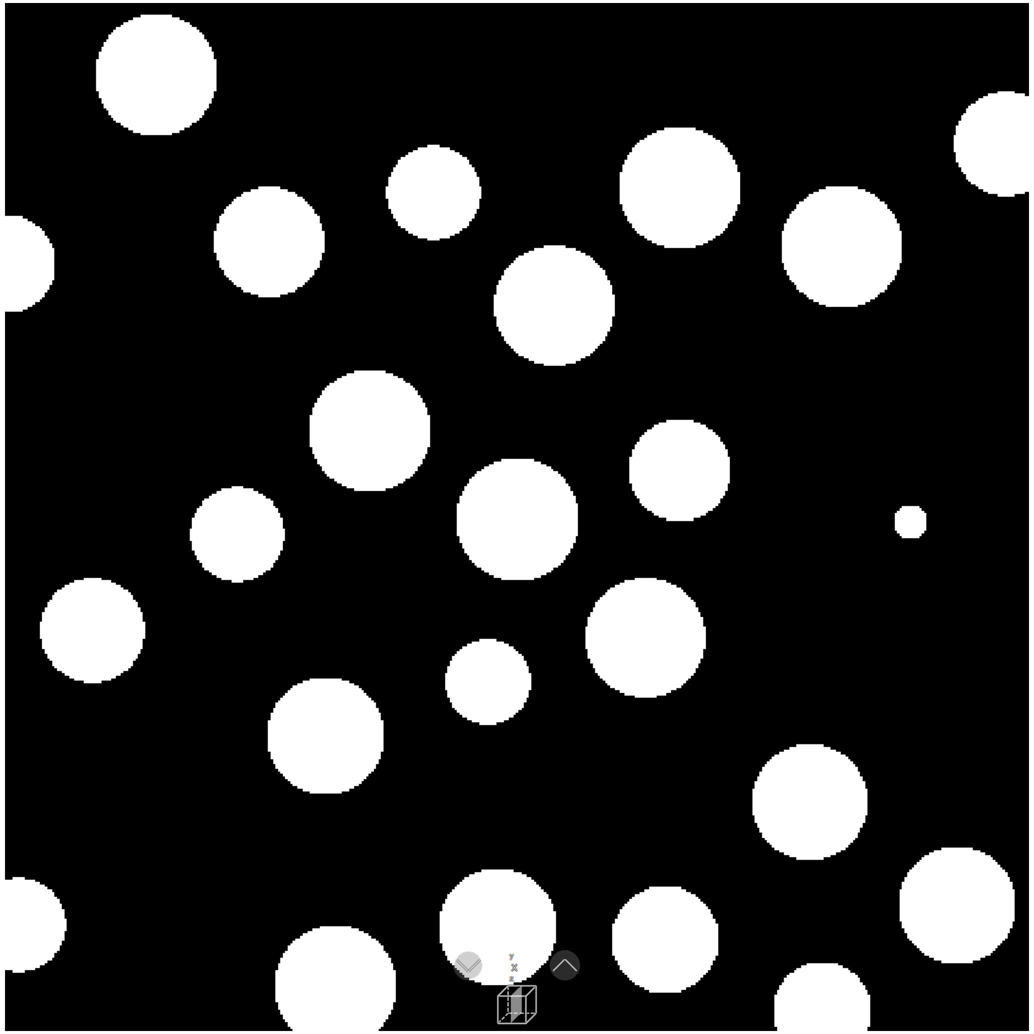
Spheres aggregation

FIRST STEP

A second output of the first step is a file with the very same name of the previous one but in a different format `[.fda]` which can be visualized using the 'PlugIm' software.

It is therefore possible to browse the different slices of the generated geometry and observe the results.

The image visualize a slice (208) of the spheres described in the text file `(sphPackCoord.txt)`.



Voronoi tessellation

SECOND STEP

The second step performs the Voronoi tessellation (starting from the centroids of the spheres/seeds) by dividing the three-dimensional space into cells which are defined by nodes and edges.

The script that does this operation, using the python `tess` library, is `tess_foam.py`.
At the end of the execution are indicated:

- Total nodes
- Total struts
- Total cells
- Nodes R
- Struts R

tess_foam.py geoGenArgPostTort.py sphPackC

```python


def bfoam(dest_p, sph_r, edg_r, iters=1, periodic=False):
    t0 = time.time()

    f_dataIN = dest_p / 'sphPackCoord.txt'
    f_dataOUT = dest_p / 'ballSticks.sb'

    print('Get input data...')
    centers, rads, n_sph, BoxTess = get_data(f_dataIN)

    print('Performing tessellation...')
    foam = foam_iter(centers, rads, BoxTess, itr=iters,per=periodic)

    allEdges = allEdgy(foam, decim=4)
    allEdges = allEdges[~delBoundEdges(allEdges, np.unique(BoxTess))]   # Rem
    allPoints = getPoints(allEdges)

    pp = allPoints - allPoints.min(axis=0) #.round().astype(int)
    ee = allEdges - allPoints.min(axis=0)
    newBox = np.ceil(np.vstack([pp.min(axis=0),pp.max(axis=0)])).astype(int)

    print('Writing output text file')
    write_obj(f_dataOUT, pp, sph_r, newBox, 'node')
    write_obj(f_dataOUT, ee, edg_r, newBox, 'edge')

    t1 = time.time()

    print('Total runtime is', t1 - t0, 'seconds')
    print('Total nodes =', len(pp))
    print('Total struts =', len(ee))
    print('Total cells = ', len(foam))
    print('Nodes R = ', sph_r)
    print('Struts R = ', edg_r)
```

Voronoi tessellation

SECOND STEP

The output of the tessellation is the list of nodes and edges which are saved in a text file (`ballSticks.sb`).

Nodes are memorized by saving their position in three-dimensional space (x_N, y_N, z_N) and their radii (***R_node***), which is a user-defined parameter.

First row shows the size of the cubic domain and the number of nodes.

The last element of each row is a running index.

ballSticks.sb

```
420 420 420 1357 1
36.8997 28.3442 350.3329 8 0
0.0000 59.0333 323.8070 8 1
43.3954 0.0000 331.3655 8 2
19.6183 0.0000 293.8985 8 3
0.0000 24.6854 288.4182 8 4
23.1798 0.0000 390.3949 8 5
56.8513 40.2623 142.3739 8 6
52.5110 66.5222 120.8631 8 7
49.7790 6.3946 149.3017 8 8
```

Voronoi tessellation

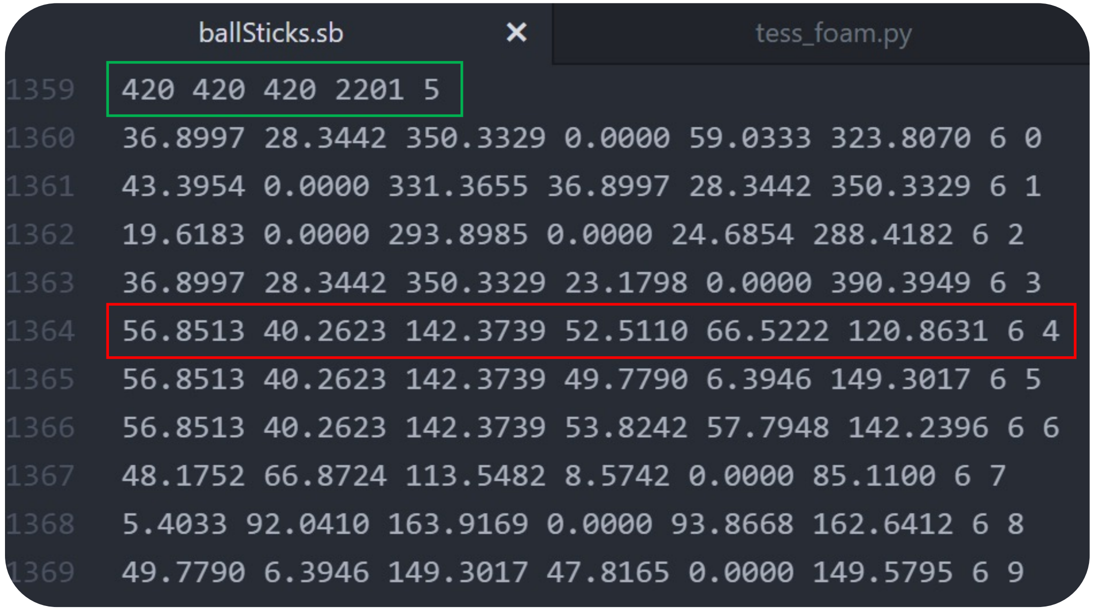
SECOND STEP

The output of the tessellation is the list of nodes and edges which are saved in a text file `(ballSticks.sb)`.

Struts are memorized by saving the position of the two nodes at the end of the strut and their radii (***R_strut***), which is a user-defined parameter.

First row shows the size of the cubic domain and the number of struts.

The last element of each row is a running index.




```
420 420 420 2201 5
36.8997 28.3442 350.3329 0.0000 59.0333 323.8070 6 0
43.3954 0.0000 331.3655 36.8997 28.3442 350.3329 6 1
19.6183 0.0000 293.8985 0.0000 24.6854 288.4182 6 2
36.8997 28.3442 350.3329 23.1798 0.0000 390.3949 6 3
56.8513 40.2623 142.3739 52.5110 66.5222 120.8631 6 4
56.8513 40.2623 142.3739 49.7790 6.3946 149.3017 6 5
56.8513 40.2623 142.3739 53.8242 57.7948 142.2396 6 6
48.1752 66.8724 113.5482 8.5742 0.0000 85.1100 6 7
5.4033 92.0410 163.9169 0.0000 93.8668 162.6412 6 8
49.7790 6.3946 149.3017 47.8165 0.0000 149.5795 6 9
```

PlugIm file generation

THIRD STEP

At this point the text file, containing the coordinates of spheres and struts, must be converted into `[.fda]` in order to be able to proceed with last foam development operations. Hence, a conversion plug-in is used.

The input file will be the one previously obtained from the tessellation operation `(ballStick.sb)` while the output will be `ballStick.fda`.

geoGenArgPostTort.py ballSticks.sb

```
nodEdgDict = {
    'FileNameFull': 'ballSticks.sb',
    'FileName': 'ballSticks',
    'IN': str(pOut / 'ballSticks.sb') ,
    'OUT': str(pOut / 'ballSticks.fda'),
    'OUTPREV': str(pOut / 'prevBallSticks.fda')
    }
```

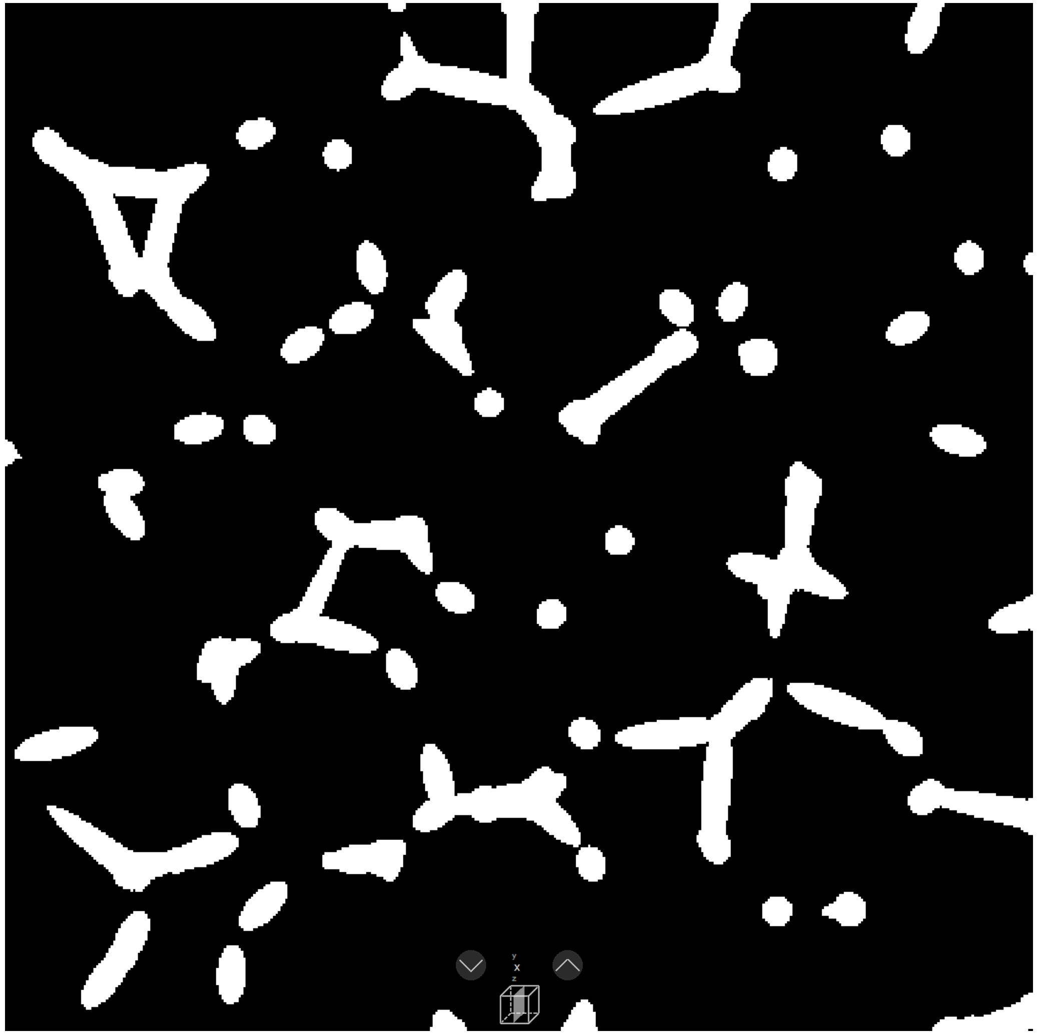
PlugIm file generation

THIRD STEP

At this point the text file, containing the coordinates of spheres and struts, must be converted into `[.fda]` in order to be able to proceed with last foam development operations. Hence, a conversion plug-in is used.

The input file will be the one previously obtained from the tessellation operation `(ballStick.sb)` while the output will be `ballStick.fda`.

The image visualize a slice (208) of the ball and stick geometry obtained from Voronoi tessellation.



Morphological closure

FOURTH STEP

Finally, using the file obtained in the previous step as input, the morphological closing operation is applied.

It generates the final geometry, very similar to a real solid foam.

The dictionary contains input and output files, the type of morphological operation (closing) and the voxel size for the chosen morphological operation.

geoGenArgPostTort.py ballSticks.sb

```
morphDict = {
#     'FileNameFull': 'test1.sb',
#     'FileName': 'test1',
    'IN': str(pOut / 'ballSticks.fda') ,
    'OUT': str(pOut / 'morphOut.fda'),
    'OUTPREV': str(pOut / 'prevMorphOut.fda'),
    'Operation': par['morph']['Operation'],
    'tv': par['morph']['tv']
    }
```

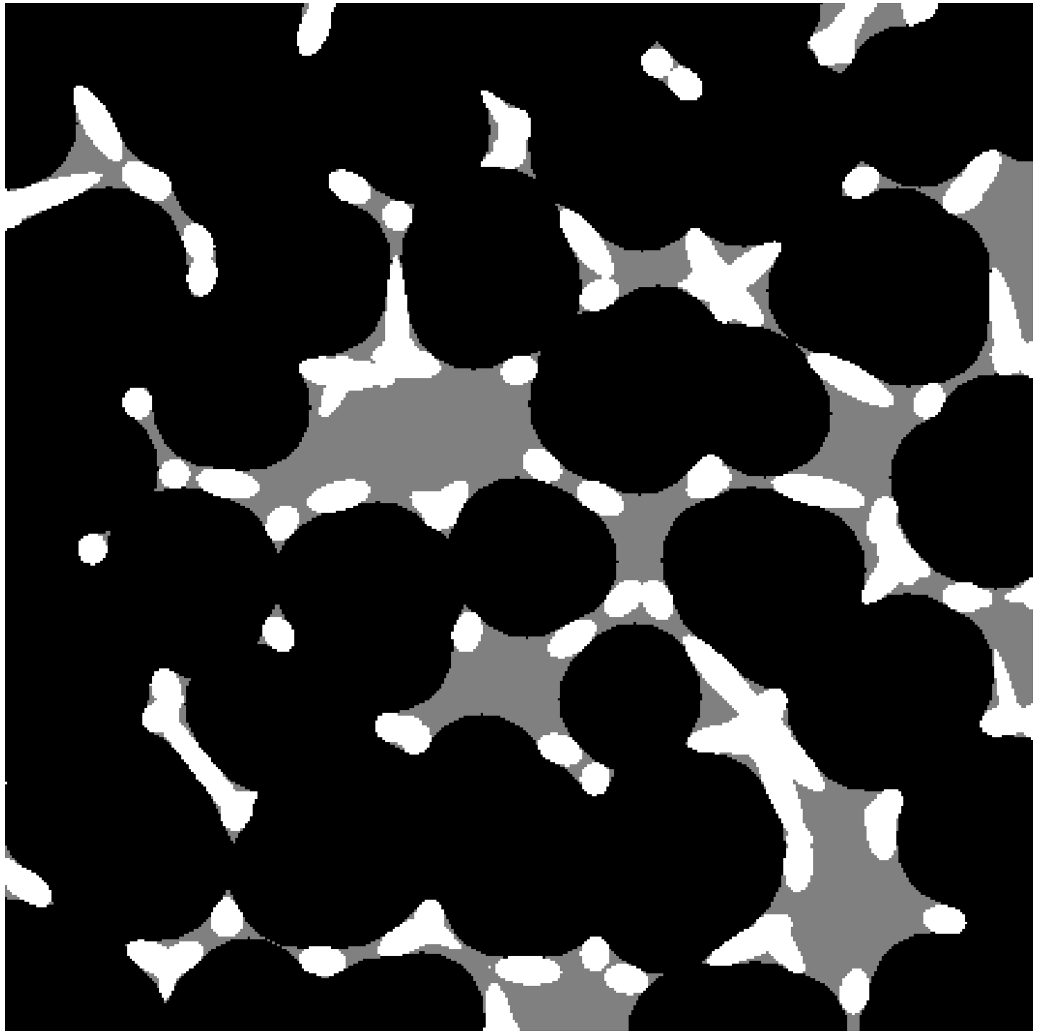
Morphological closure

FOURTH STEP

Finally, using the file obtained in the previous step as input, the morphological closing operation is applied.

It generates the final geometry, very similar to a real solid foam.

Using PlugIm, it is now possible to display a preview of the morphological operation and finally open the file containing all the slices of the final geometry, once the process is complete!



Postprocessing
Calculations and results

- Other plugins are used later to calculate surface area, solid volume fraction, tortuosity, porosity and specific surface area. Each plug-in generates a text file `[.txt]` with calculated results.
- An image file containing the stack with all slice images `[.tif]` is also exported. It can be opened with any image viewing program.
- Finally, using the `export_stl.py` script, a 3D stereolithography file is exported.
  It will be used with the openFoam tool snappyHexMesh to build the solid foam mesh domain.

Surface Area

POSTPROCESSING

The first postprocessing plug-in calculates the total surface area of the solid foam and the volume fraction occupied by the solid phase.

Results are reported in the output text file `surfaceArea3d.txt`.

Also the total volume is computed.

surfaceArea3d.txt

```
File : morphOut
Vv : 0.151707914911997
Total volume : 11239736
Total surface area : 2482753.18962446

```

geoGenArgPostTort.py   GBTortuosity.txt

```python
surfaceDict = {
        'Export': str(pOut / 'surfaceArea3d.txt'),
        'FileNameFull': 'morphOut.fda',
        'FileName': 'morphOut',
        'FileDir':  str(pOut),
        'IN': str(pOut / 'morphOut.fda') ,
        'OUT': str(pOut / 'morphOutSurf.fda'),
        'OUTPREV': str(pOut / 'morphOutSurfPrev.fda')
    }
```

FDA to TIF image file

POSTPROCESSING

The plugIm file, containing all the slides of the final geometry is converted into an image file `[.tif]` for quick visualization, even on external programs.

surfaceArea3d.txt

```
File : morphOut
Vv : 0.151707914911997
Total volume : 11239736
Total surface area : 2482753.18962446

```

geoGenArgPostTort.py

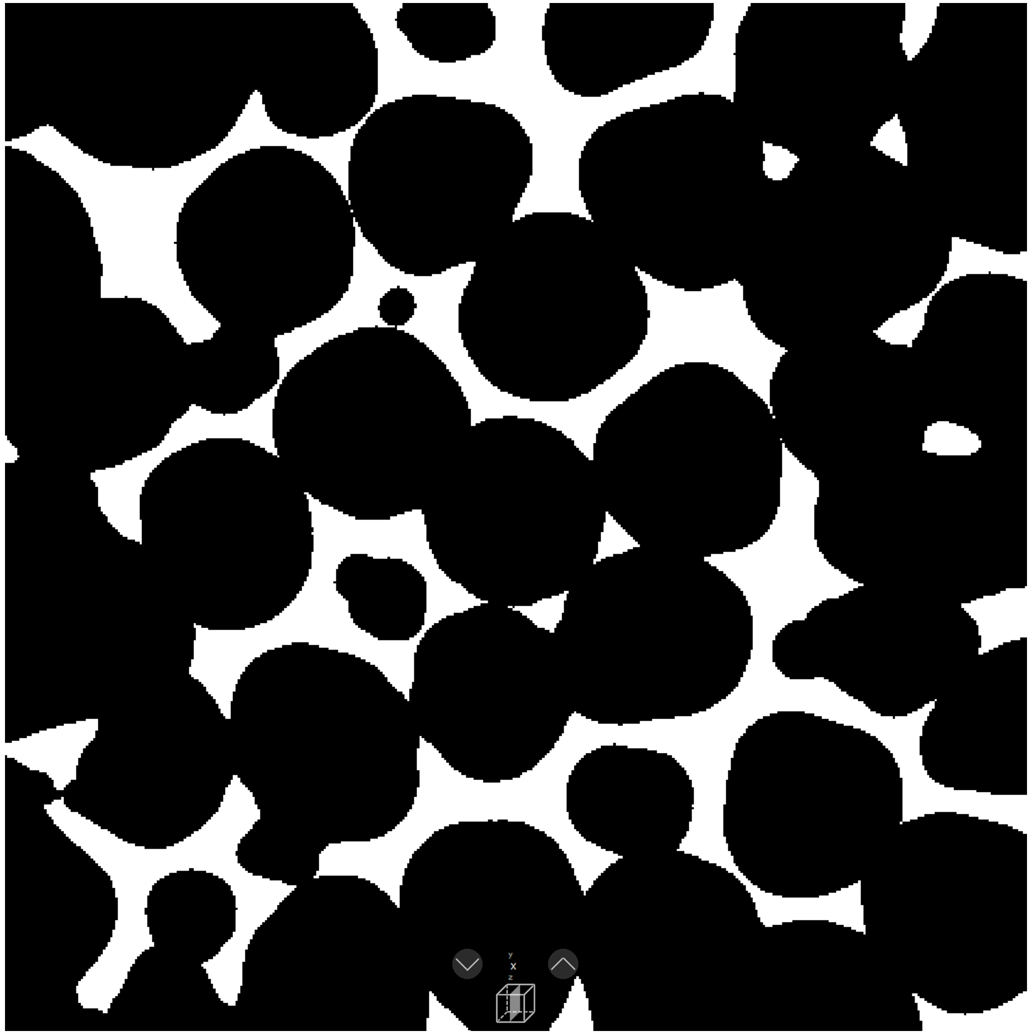
```

tiffDict = {
    'IN': str(pOut / 'morphOut.fda') ,
    'OUT': str(pOut / 'foamTest.tif'),
    }
```

FDA to TIF image file

POSTPROCESSING

The plugIm file, containing all the slides of the final geometry is converted into an image file `[.tif]` for quick visualization, even on external programs.

The image visualize a slice (208) of the final solid foam geometry obtained from morphological closure step.



GB Tortuosity

POSTPROCESSING

Graph based tortuosity calculation is performed using the `graphBased3D` plug-in.

Results are reported in the output text file `GBTortuosity.txt`.

```
Graph-based tortuosity : 1.172
Cycle Graph Computation time (44.075s.)
```

geoGenArgPostTort.py GBTortuosity.txt

```python
tortDict = {
    'nb_samples': str(50),
    'Weight': 'Dist',
    'Export': str(pOut / 'GBTortuosity.txt'),
    'FileNameFull': 'morphOut.fda',
    'FileName': 'morphOut',
    'FileDir':  str(pOut),
    'IN': str(pOut / 'morphOut.fda') ,
    'OUT': str(pOut / 'morphOut.fda'),
    'OUTPREV': str(pOut / 'morphOutSurfPrev.fda')
    }
```

macroDescriptors

POSTPROCESSING

Finally, in the main script, performed all the workflow operations, some macro descriptors such as porosity and specific surface area are calculated using the `postprocess.py` script.

Results are reported in a text file called `macroDescriptors.txt`.

macroDescriptors.txt

```
porosity= 0.848
Sv= 747.270

```

```python
def macroPar(img, vx_res, pOut):
    print('\npostprocessing the image')
    im =io.imread(img)
    s, v, f = surf(im)
    sv = sv_calc_uneven(s, vx_res, im.shape)
    por = 1- (im/255).mean()
    with open(pOut / 'macroDescriptors.txt','w') as f:
        f.write('porosity= {:.3f}\n'.format(por))
        f.write('Sv= {:.3f}\n'.format(sv))
    print('\nporosity =',por)
    print('\nspecific surface =', sv)
    print('\n')

def padSTL(img, pOut):
    im =io.imread(img)
    impad = padding(im,2).astype(np.uint8)
    tif.imwrite( pOut / 'paddedFoam.tif', impad, dtype=np.uint8) #,co
    s, v, f = surf(impad)
    t0 = time.time()
    export_stl(v,f, pOut / 'foamSTL.stl')
    t1 = time.time()
    print('\nTotal stl creation time is', t1 - t0, 'seconds')
```

FDA to SLT file

POSTPROCESSING

`export_stl.py` provides an effective method for obtaining a stereolithography file from the final geometry.

It is fundamental in the meshing process of the computational domain regarding to computational fluid dynamics simulations on solid foams.

export_stl.py geoGenArgPostTort.py

```python
import numpy as np
from skimage.measure import marching_cubes
from skimage.measure import mesh_surface_area
import trimesh
from skimage import io

def binarize(im, threshold):
    boolean = im > threshold
    binarized = np.multiply(boolean, 1)
    return binarized

def surf(image):
    verts,faces,_,_ = marching_cubes(image,0)
    surface = mesh_surface_area(verts,faces)
    return surface, verts, faces

def export_stl(verts, trifaces,fileout):
    mesh = trimesh.Trimesh(vertices=verts, faces=trifaces)
    mesh_stl = mesh.export(fileout+'.stl')

def sv_calc(surface, voxsize, boxsize):
    vx_vol = voxsize ** 3
    vx_surf = voxsize ** 2
    spec_surf = ( surface * vx_surf )/ (vx_vol * boxsize ** 3 )
    return spec_surf

im = io.imread(filein)

im = np.logical_not(binarize(im, 254))

cutvx = 10
cut = im[cutvx:-cutvx, cutvx:-cutvx, cutvx:-cutvx]

pad = np.pad(im, 2, 'constant', constant_values=True)
```

Resulting SLT file

ON PARAVIEW