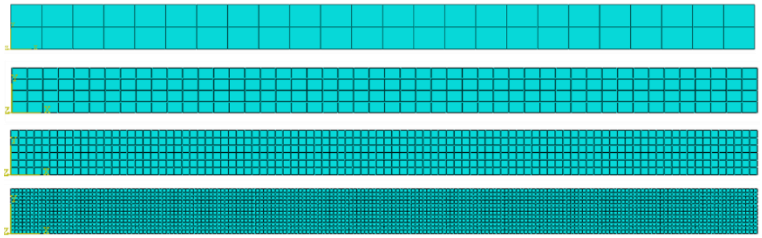
Version 1.0

Sunil Anandatheertha

CHECKERBOARD MODEL



Process step number	Process	Details/comments
1	Python pre-process	Generates the checkerboard model in ABAQUS
2	ABAQUS generate input file	Generate CHKmodel_IDinfo-append0.inp file *FILE GENERATED*: CHKmodel_IDinfo-append0.inp
3	MATLAB generate material data from texture	Generates texture informed material template material data in material_IDinfo.txt in umat format for ABAQUS *FILE GENERATED*: material_IDinfo.txt
4	Manually copy contents of material_IDinfo.txt into the material data location in CHKmodel_IDinfo-append0.inp to get CHKmodel_IDinfo-append1.inp	This step readies the input file for analysis This should be automated using python script itself
5	Manually generate custom CHKmodel_IDinfo-append1.slurm file needed for HPC	This should be automated using python script itself
6	Copy the [CHKmodel_IDinfo-append1.inp] and [CHKmodel_IDinfo-append1.slurm] to HPC folder	Either individually or in batch
7	Solve the model either individually or in batch	Try parallel processing on HPC
8	Download the ODB files	
9	Python in ABQUS post-processing to generate data and image files	
10	Python external – result assimilation	
11	Python external – result analysis and presentation accumulation	

GENERATION OF 2D CHECKERBOARD GRAIN STRUCTURE

PYTHON program: Vasudeva_Python_Script_01.py

Version 1.0

Sunil Anandatheertha

External call: none

Description: Model bottom corner is origin situated at “Model_origin_x” and “Model_origin_y”. Give the model dimensions in mm in “Model_enddim_x” and “Model_enddim_y”; 100 and 6 can represent sub-size tensile specimen. Set the number of partitions needed along y direction in “NumPartitions_y” and the number of grains needed in “TOTAL_NO_GRAINS”. Set the “ElementSize” and “TotalStrain_X” values. Choose the “MeshControlOption” option from the below set of choices as per the type of meshing needed and the type of the element

 # MeshControlOption 1: QUAD Free, medial axis, minimize the mesh transitions yes
 # MeshControlOption 2: QUAD Free, medial axis, minimize the mesh transitions no
 # MeshControlOption 3: QUAD Free, advancing front
 # MeshControlOption 4: QUAD Structured, minimize mesh transiion yes

 # MeshControlOption 5: QUAD DOMINATED Free, medial axis
 # MeshControlOption 6: QUAD DOMINATED Free, advancing front

 # MeshControlOption 7: TRI free, use mapped meshing where appropriate yes
 # MeshControlOption 8: TRI structured

Choose the “ElementTypeOption” from the below choices as per mesh quality and functionality defining parameters

 # ElementTypeOption 01: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_default --- HGC_default
 # ElementTypeOption 02: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_default --- HGC_enhanced
 # ElementTypeOption 03: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_yes --- HGC_default
 # ElementTypeOption 04: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_yes --- HGC_enhanced
 # ElementTypeOption 05: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_no --- HGC_default
 # ElementTypeOption 06: LINEAR Quad --- RI_yes --- 2ndOA_no --- DC_no --- HGC_enhanced
#.....
 # ElementTypeOption 07: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_default --- HGC_default
 # ElementTypeOption 08: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_default --- HGC_enhanced
 # ElementTypeOption 09: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_yes --- HGC_default
 # ElementTypeOption 10: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_yes --- HGC_enhanced
 # ElementTypeOption 11: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_no --- HGC_default
 # ElementTypeOption 12: LINEAR Quad --- RI_yes --- 2ndOA_yes --- DC_no --- HGC_enhanced
#.....
 # ElementTypeOption 13: LINEAR Quad --- RI_no
#.....
 # ElementTypeOption 14: LINEAR tri --- 2ndOA_no --- DC_default
 # ElementTypeOption 15: LINEAR tri --- 2ndOA_no --- DC_yes
#.....
 # ElementTypeOption 16: LINEAR tri --- 2ndOA_yes --- DC_default
 # ElementTypeOption 17: LINEAR tri --- 2ndOA_yes --- DC_yes
 # ElementTypeOption 18: LINEAR tri --- 2ndOA_yes --- DC_no
#.....
 # ElementTypeOption 19: QUADRATIC Quad --- RI_yes
 # ElementTypeOption 20: QUADRATIC Quad --- RI_no

```
#.............................
# ElementTypeOption 21: QUADRATIC tri --- MF_yes --- 2ndOA_no --- DC_default
# ElementTypeOption 22: QUADRATIC tri --- MF_yes --- 2ndOA_no --- DC_yes
# ElementTypeOption 23: QUADRATIC tri --- MF_yes --- 2ndOA_no --- DC_no
#.............................
# ElementTypeOption 24: QUADRATIC tri --- MF_yes --- 2ndOA_yes --- DC_default
# ElementTypeOption 25: QUADRATIC tri --- MF_yes --- 2ndOA_yes --- DC_yes
# ElementTypeOption 26: QUADRATIC tri --- MF_yes --- 2ndOA_yes --- DC_no
#.............................
# ElementTypeOption 27: QUADRATIC tri --- MF_no
```

Set thew total percentage strain in “TotalStrain_x”

At the end, copy paste the following code if info is needed on elements and nodes immediately

```
mdb.models['Model-1'].rootAssembly.regenerate()
instance = model.rootAssembly.instances['partname']
TotalNumOfNodes = len(instance.nodes)
TotalNumOfElem  = len(instance.elements)
usethis = 0
if usethis==1:
       if str(instance.elements[1].type)=='CPS4R' or str(instance.elements[1].type)=='CPS4':
              Total_DOF = TotalNumOfNodes*4
       elif str(instance.elements[1].type)=='CPS8R' or str(instance.elements[1].type)=='CPS8':
              Total_DOF = TotalNumOfNodes*8
print('%d Elements --- %d Nodes --- '% (TotalNumOfElem, TotalNumOfNodes))
print('%d Grains'% (TOTAL_NUM_GRAINS))
```

GENERATION OF TEXTURE MODEL – IDEAL ORIENTATIONS

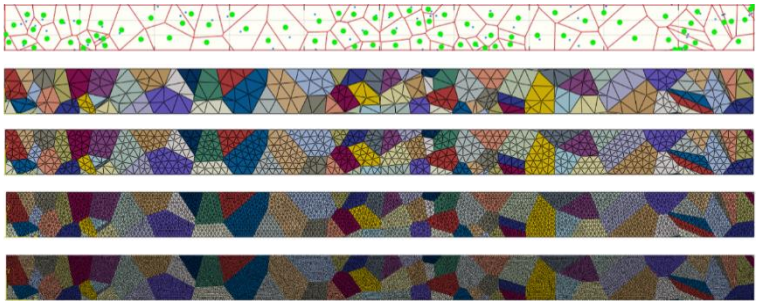
MATLAB program: WriteMatDataFOR_CPFEM_NEW_VORONOI.m
External call: none
Description: This generates N_GRAINS number samples of the orientation by respecting the volume fraction specified. Alter “Vf_loc = Vf_loc_B;” to A or C to construct texture model for locations A and C. The correct “Grain_Structure_ID_10digit_number” should be specified at the beginning to use the right dataset. This number is output from “voronoigrainstructure_new.m”.
Output: Materials data in the format needed by ABAQUS UMAT (Haungs) is produced and compiled for all grains and stores results in the file “FILE10” (see below), original data and grain structure image are in the following files
FILE10: GRAINSTRUCTUREIDNUMBER_materials_data.txt --- UMAT data

++

VORONOI MODEL

Version 1.0

Sunil Anandatheertha



GENERATION OF 2D VORONOI TESSELATED GRAIN STRUCTURE

MATLAB program: voronoigrainstructure_new.m
External call: VoronoiLimit.m
Description: Startx, starty, lengthx, widthy define the dimensions of the tensile specimen test section. Only test section is used. Modelling of grips is a 'to-do' for later. Nopoints_i and Nopoints_j define the total number of seed points that will be available. Only their product is important at the moment. For example, entering 10, 4 or 4, 10 would not make a difference as of now. xscale and yscale decide how much of the available seed point data can be used. They should be >1 for now. Greater it is smaller will be the number of grains. They stretch the underlying points but not the limiting rectangle, thus achieving scaling.
Output: Results, original data and grain structure image are in the following files
(1) FILE01: GRAINSTRUCTUREIDNUMBER_ReadMe_Information.txt --- information about the grain structure
(2) **FILE02**: GRAINSTRUCTUREIDNUMBER_x_data_tess_2dvor.txt --- x coordinates of the polygons
(3) **FILE03**: GRAINSTRUCTUREIDNUMBER_y_data_tess_2dvor.txt --- y coordinates of the polygons
(4) **FILE04**: GRAINSTRUCTUREIDNUMBER_c_data_tess_2dvor.txt --- centroid coordinates
(5) FILE05: GRAINSTRUCTUREIDNUMBER_xoriginal_unscaled_2dvor.txt --- unscaled seed data – x coordinate
(6) FILE06: GRAINSTRUCTUREIDNUMBER_yoriginal_unscaled_2dvor.txt --- unscaled seed data – y coordinate
(7) FILE07: GRAINSTRUCTUREIDNUMBER_xoriginal_scaled_2dvor.txt --- scaled seed data – x coordinate
(8) FILE08: GRAINSTRUCTUREIDNUMBER_yoriginal_scaled_2dvor.txt --- scaled seed data – y coordinate
(9) FILE09: GRAINSTRUCTUREIDNUMBER_2dVorTesselation.jpeg --- jpeg image of the grain structure

Number of rows in the data inside FILE04 is the number of grains (N_GRAINS). It is also given in FILE01.

GENERATION OF TEXTURE MODEL – IDEAL ORIENTATIONS

MATLAB program: WriteMatDataFOR_CPFEM_NEW_VORONOI.m
External call: none
Description: This generates N_GRAINS number samples of the orientation by respecting the volume fraction specified. Alter "Vf_loc = Vf_loc_B;" to A or C to construct texture model for locations A and C. The correct "Grain_Structure_ID_10digit_number" should be specified at the beginning to use the right dataset. This number is output from "voronoigrainstructure_new.m".
Output: Materials data in the format needed by ABAQUS UMAT (Haungs) is produced and compiled for all grains and stores results in the file "FILE10" (see below), original data and grain structure image are in the following files

Version 1.0

Sunil Anandatheertha

FILE10: GRAINSTRUCTUREIDNUMBER_materials_data.txt --- UMAT data

RE-GENERATE THE VORONOI TESSELLATION IN ABAQUS AND MAKE CPFEM .INP FILE

PYTHON program: Voronoi_GS_01.py

External call: none

Description: This reads data from FILE02, FILE03 and FILE04 and re-creates the grain structure. Input the "filenumber" correctly to required GRAINSTRUCTUREIDNUMBER. Currently individual element seeding is not possible. But the global element size is controlled using "GlobalElementSize". Comment all codes under "# TRI---LINEAR----1st order accuracy" or "# TRI---QUADRATIC----2nd order accuracy" as necessary. This py script file should be in the ABAQUS working directlory. In ABAQUS, File-> run script -> point to this script file. Wait for the model to generate. Then, regenerate part, go to mesh module and check if mesh ok. Else re-do the process. If satisfied, we can go ahead. Job-> Create a new job NEWJOBNAME and then write input. Open NEWJOBNAME.inp in NOTEPAD++ (Recommended).

Output: **FILE11**: NEWJOBNAME.inp

At the end, copy paste the following code if info is needed on elements and nodes immediately

```
mdb.models['Model-1'].rootAssembly.regenerate()
instance = model.rootAssembly.instances['partname']
TotalNumOfNodes = len(instance.nodes)
TotalNumOfElem  = len(instance.elements)
usethis = 0
if usethis==1:
       if str(instance.elements[1].type)=='CPS4R' or str(instance.elements[1].type)=='CPS4':
              Total_DOF = TotalNumOfNodes*4
       elif str(instance.elements[1].type)=='CPS8R' or str(instance.elements[1].type)=='CPS8':
              Total_DOF = TotalNumOfNodes*8
print('%d Elements --- %d Nodes --- '% (TotalNumOfElem, TotalNumOfNodes))
```

STEP	***DETAILS***
1	EXECUTE **voronoigrainstructure_new.m** with the right values Look for the file name displayed at the end of run in the command window. Note the file ID number at the beginning of the filename Copy the output files FILE02, FILE03 and FILE03 bearing the corresponding ID number to ABAQUS working directory
2	EXECUTE **WriteMatDataFOR_CPFEM_NEW_VORONOI.m** with the right values This produces FILE10
3	Open FILE11 in NP++ and delete everything between ** ** MATERIALS ** and ** ** BOUNDARY CONDITIONS. Open FILE10 in NP++ and copy all contents Paste these in the place of material information just deleted from FILE11. Save the file.
4	Copy FILE11 to HPC working folder "MYFOLDER"

	Copy HKumat.f to MYFOLDER Copy cpfem.slurm to MYFOLDER Open cpfem.slurm in NP++. Replace INPUT value to NEWJOBNAME. Alter job-name value if needed
5	Open Putty. >>cd MYFOLDER >> module purge >> module load abaqus >> module load intel >> module load impi (ignore if this returns an error) >> module load intel/13 >>sbatch cpfem.slurm Wait for the solution Solution updates happen regularly. Progress can be seen in the NEWJOBNAME.sta file. Solution is only successful if it is displayed so in this file. Total number of degrees of freedom can be seen inside NEWJOBNAME.msg file Copy the NEWJOBNAME.odb file into the desktop ABAQUS working folder. Open in abaqus and do further processing.