Additional Thesis

MANUAL FOR MICRO-BATTERY PARAMETRIC STUDY

Tianxiang Wang
Student number: 4620127

Table of Contents

1. Overview

The emergence of new kinds of micro-technologies, e.g., MEMS (microelectromechanical systems) devices, biomedical microma- chines, remote sensors, etc., has given rise to new approaches in battery development. The additional thesis contributes part of Finite Element Analysis of microbatteries, to be specific, the trenched mirobatteries architecture. The project aims to perform a parametric study of micro-battery, including a python script implemented and an example of asymmetric trench model simulation. The python executable allows to run the parametric study in an automatic way. The asymmetric working example provides a generic trench model template.

The python executable consists of six functional blocks, namely: reading the input data from users, generating meshes with multiple parameters, splitting the interface nodes, refining the mesh files, running the simulations, and collecting the generic output parameter. Generating meshes allows users to create 2D triangle mesh with two parameters for a given geometry. Splitting the interface nodes uses a generic manner to split interfaces for an arbitrary structure. Once splitting interface is done, the executable is able to seek the new nodes on the interface and compile them in a prescribed fashion before simulation. Coming into the last part, the executable makes use of a set of external programs to run Jem-Jive carrying out the relative outcome. A simple case will go through every step how this script works, after which a more complicated model will be illustrated.

Generally, though taking the advantage of symmetry of trench makes the parametric study easier and straightforward to handle with, which only the half of trench is considered, an asymmetric trench model is more probable and realistic. An example is provided afterwards with five pillars which not only their length may vary, but imperfection is introduced here: pillars can be slightly inclined as well until they touch each other. An important property worth noticing is the top and bottom of the trench stay smoothie although inclination occurs and the neutral layer of pillar still remains the same length. The example gives a detailed universal geometric derivation and formulas with respect to heights and inclined angles, which are regarded as two relative variables implemented by the python executable mentioned above.

2. Script Introduction

The executable is written in python, expected be executed on Ubuntu.

In general, the script is designed to run a series of simulations with different user-defined parameters, and collect the data we are interested in for further analysis. The detailed description is as follow: Given an available reference input file constituted by parameters for Gmsh, the script substitutes users specified parameters (not more than 2) with lists of user-defined variables and generates new input files for Gmsh. By going through Gmsh for every input file, mesh files can be created. However, due to the interface nodes could not be shared by two different physical groups, which is not compatible in Jem-Jive, splitting meshes is necessary to allocate all nodes into independent groups. The script enables to find all the newly added nodes on the interfaces and labels them in a prescribed format for Jim-Jive. If all the above steps were done correctly, results could be collected after running the simulations, and the script would iterate through next variable.

Additionally, four external programs are required to be installed, namely:

Gmsh	Gmsh is a free 3D finite element mesh generator with a built-in CAD engine and post-processor. Here is used to generate the mesh for trenched structures with different parameters.
ciGen[1]	An open source program to generate zero-thickness cohesive interface elements in existing finite element discretization. The program is useful in numerical modeling of material/ structure failure using cohesive interface elements. In this case, it will split the interface nodes between Anode/Cathode and Electrolyte.
JemJive[2]	Jive is an open source, research-oriented C++ programming toolkit aimed at solving partial differential equations (PDEs). It is used to perform the Finite Element Analysis for the given micro-battery.
Additional package	A set of folders containing the C-language based script to set up initial conditions and input meshes for the Jem-Jive simulation.

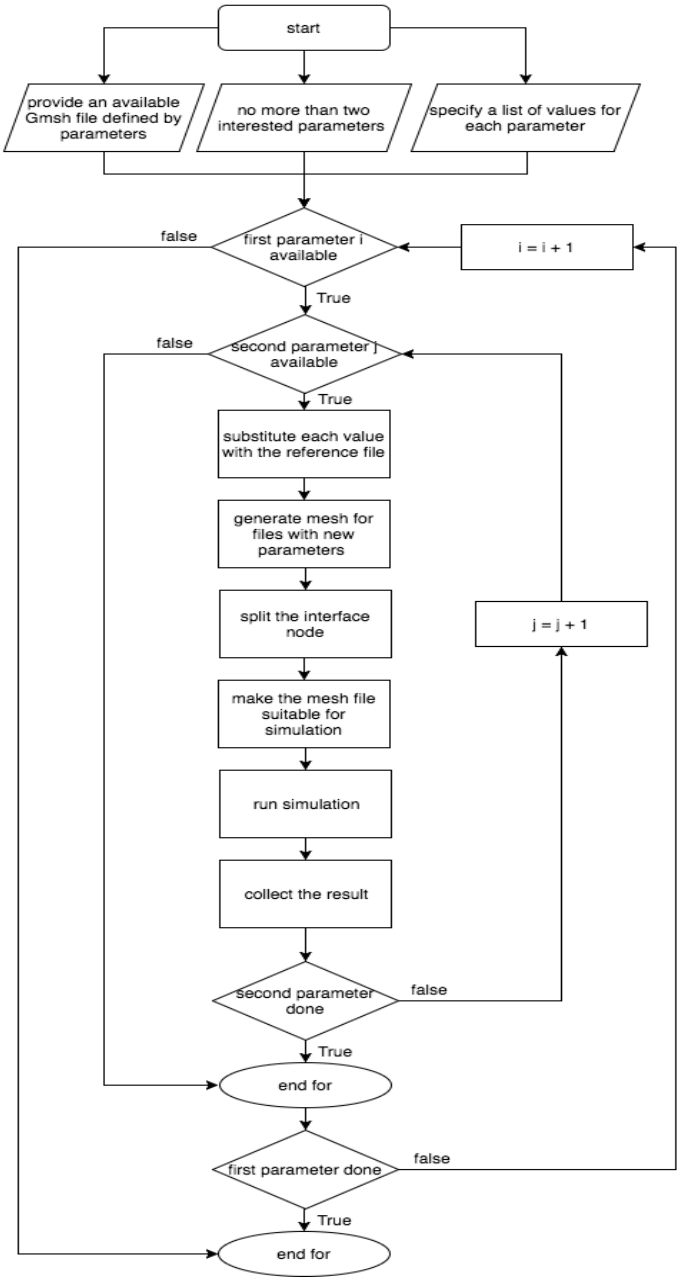
2.1 Advantages

(1) To run a Finite Element Analysis for a trenched micro-battery, few steps need to be done beforehand, the executable helps to substitute interested parameters, generate mesh, make the mesh suitable for simulations, run simulations, and read the output automatically.
(2) Users could input two unlimited lists of parameters they would like to analyze arbitrarily, the script is able to iterate through every single variable and collect all the results with one execution, which saves plenty of time in comparison with one variable.
(3) The script has a lot flexibilities for users, in the meanwhile, comprehensive enough. Different functions could be achieved by commenting out other parts. For example, it could be converted to a useful mesh generator without GUI.

2.2 Tasks of the Script

The script is composed of several functional blocks accomplishing the following tasks, the related programs are listed as well:

(1) Get the values for the parameters of interest: Python script
(2) Generate mesh for new input files: Gmsh
(3) Make each node independent from the others: ciGen
(4) Make mesh file suitable for the battery implementation in Jem-Jive: Python script
(5) Run a (Jem-Jive) simulation with the newly created mesh: Jem-Jive
(6) Read output of interest resulting from the Jem-Jive simulation: Python script



2.3 Directory Structure

Once all necessary programs are installed, users should continue to set up for the directories and folders. The structures are defined as follow, directories could be changed into user's preference, but all those folders must be under working directory.

Working Directory: includes all the files to perform parametric study.
Default directory: workDir = `"/home/tianxiangwang/Desktop/Additional-Thesis/"`

Directory for Gmsh: stores the updated input files and corresponded output mesh files.
parameters-containing directory: `paraDir = workDir/parameters`
simulations-containing directory: `simuDir = workDir/simulations`

Directory for ciGen: contains all the files that refer to ciGen code.
ciGen-executable directory: `ciGen = workDir/interface-generator/src/interface-elem`

Directory for Jim-Jive: includes all information to run simulations
Input Mesh directory: `workDir/Meshes`
Initial condition for FEA directory: `workDir/inputFiles`
Additional package directory: `workDir/src/`
Output directory: `workDir/inputFiles/timeValues/`

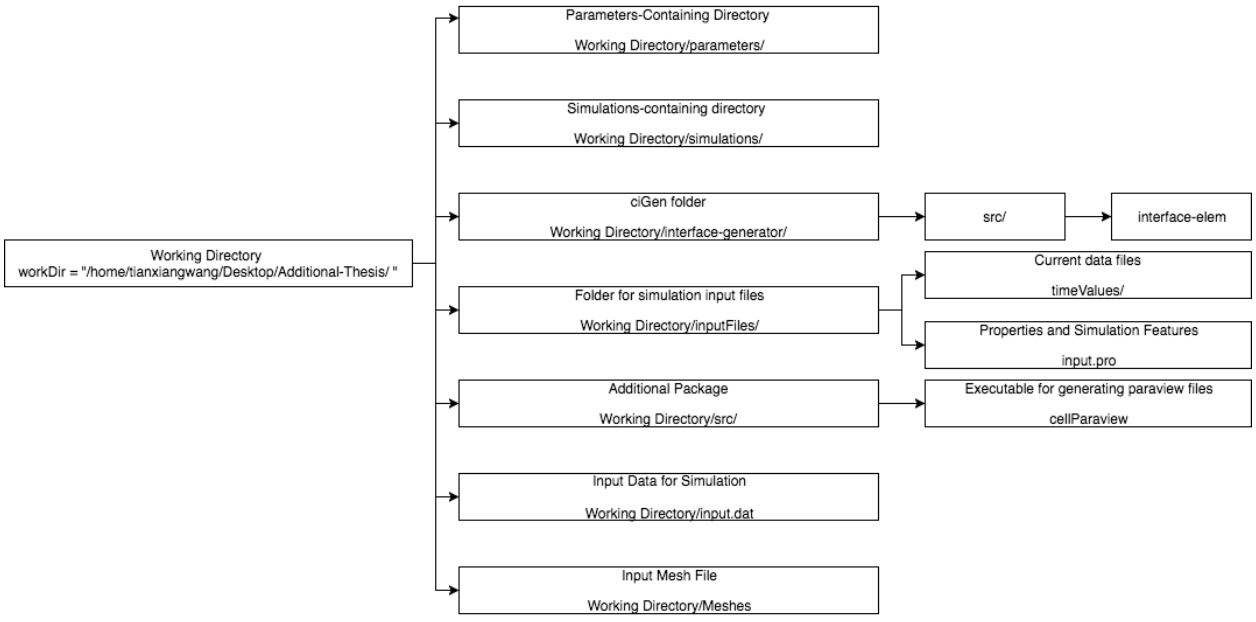


Figure 1. Subfolders structure

```
# Working directory
workDir = "/home/tianxiangwang/Desktop/Additional-Thesis/"

# Parameters-containing directory
paraDir = workDir + "parameters/"

# Simulations-containing directory
simuDir = workDir + "simulations/"

# ciGen folder
ciGen = "interface-generator/src/interface-elem"

# Folder for simulation input files
Simuinput = "inputFiles/input.dat"

# Executable for generating paraview files
paraview = "src/cellParaview input.pro"

# Current data files
current = workDir + "inputFiles/timeValues/"

# Result
result = workDir + "result.txt"

# Gmsh filename
Filename1 = "multitrenches"
```

3. Tasks Description

This part is the detailed interpretation of tasks above, each point is corresponding to the brief description in 2.2:

(1) Python script lets users first specify no more than two interested parameters, afterwards input a list of value for each parameter. The script will memorize those values and substitute them individually with the reference file to create new input for meshing with new parameters defined.

(2) 2D triangle mesh scheme is selected for meshing the updated file.

(3) In order to be compatible for simulation, the nodes should be assigned to different node groups, so once the mesh has been created, use ciGen to split the interface to assure each node has been allocated in its own node group.

(4) Right after splitting the nodes on the interfaces, the new nodes on the interface haven't been labeled yet. The script manipulates the file generated by ciGen by adding their labels as the same fashion with other node groups. Moreover, constrains on nodes must be defined only once, the overlapped one needs to be deleted. Overall, two properties must be reached before simulation:
(a) All nodes have been allocated into a certain node group.
(b) For nodes enforced by constrains could only be defined once.

(5) The script would run Jem-Jive as all the requirements are fulfilled and initial condition is set up properly which is updated every iteration.

(6) There are a few of outcomes for battery behavior. In this case, Python script prints out the current integral initially, which could be changed into another property according to use's interest.

3.1 Parameters of interest

(1) Function: Initialize, specify which two parameters in the reference geometry*(referenceFile.geo)* will be substituted by a list of values from user's input. According to user's interest, it generates a bunch of new .geo-files*(updatedFile.geo)* with different parameters as defined by users, and all of them will be stored at the folder `workDir/parameters`

(2) Related program: python script, no external program is called

(3) Limitations: The executable is only available for less than 2 variables, if one is interested in multiple parameters, the executable should be run multiple times.

(4) Necessary files and folders: A reference geometry*(referenceFile.geo)*, a folder to store new geometry files (as default, `workDir/parameters/`)

(5) Input & Output:

Input	Output
An available referenceFile.geo	updatedFile.geo

Declare interested parameters	
Assign values for those parameters	

3.2 Gmsh Execution

(1) Function: Generate 2D triangle mesh for the *updatedFile.geo* created by last step and save the mesh files for further execution.
(2) Related program: Gmsh
(3) Limitations: The executable does not have a self-check scheme, which means one should open the mesh file and Gmsh to check whether the node distribution is correct.
(4) Necessary files and folders: updatedFile.geo, a folder to store mesh files (as default, `workDir/simulations/`)
(5) Input & Output

Input	Output
updatedFile.geo	updatedFile.msh

3.3 Make Every Node Independent

(1) Function: For the nodes on the interface, Gmsh only gives one node assignment for each of them, which may lead to the incompatibility for simulation. The script utilizes an external program ciGen to split those nodes into two, guaranteeing either side of the interface would have one unique node though their coordinates are same. ciGen is not the specialized open source code for splitting battery structures, but a generic method to generate zero-thickness cohesive interface elements. After splitting nodes, new nodes will be generated along the interfaces. More importantly, the outcome of ciGen categorizes different node groups by an ascend sequence of numbers as defined in the original reference file that is related to the sequence of setting up different physical lines in Gmsh. The script will eventually read from different node groups so that it is better to have a physical meaning instead of pointless numbers for group names. In accordance with the original reference file, the script here manages to rename the node groups. Generally, users should modify the sequence of the strings beforehand to match their own convention.
(2) Related program: ciGen, rename part is done by python script
(3) Limitations: The script does not display the result from ciGen where all splitting information is listed including the amount of newly added nodes. In terms of checking the result, it could only be done by virtualizing the mesh file. For renaming part, the script is not intellectual enough to distinguish which set of data belongs to which groups. Users are supposed to adjust the sequence to connect with their convention manually.
(4) Necessary files and folders: updated mesh file from Gmsh, working under simulation directory.
(5) Input & Output

Input	Output

updatedFile.msh	updatedFile.mesh

3.4 Make mesh file suitable for Jem-Jive

(1) Function: One deficiency we have to improve with is from ciGen the newly added interface nodes are not labeled and do not belong to any of node groups that have been created before. Moreover, in this case, the constrain for interface nodes on left/right edge should not be defined multiple times. What the script does is it could find the missing points caused by splitting nodes then append them in the same format with other node groups and delete the overlapped nodes on boundaries, which guarantees the simulation will not crash.
(2) Related program: python script
(3) Limitation: The script is so specific that it is restrained with three layers trenched model, though the algorithm to find the missing nodes could be referred wildly.
(4) Necessary files and folders: ciGen outcome files, working under simulation directory.
(5) Input & Output

Input	Output
updatedFile.mesh	modifiedFile.mesh

3.5 Run the Simulations

(1) Function: With all the steps completed, input files for simulation still need to be manipulated. The script is able to move the modified mesh file to a specific folder to initialize the simulation as well as update the input information before each iteration. Once everything is done properly, the script will call the additional package to run Jem-Jive.
(2) Related program: Additional package, Jem-Jive
(3) Limitation: The script does not provide an indicator noticing whether the simulation is executed successfully or not. This could only be tested from the outcome.
(4) Necessary files and folders: a correct directory for additional package as mentioned in Figure 1 , the modified mesh file from last step, working under the additional package directory. Mesh file must be put into the folder *Meshes*
(5) Input & Output

Input	Output
modifiedFile.mesh	All data in timeValues folder

3.6 Collect Results

(1) Function: This block is used to collect the relative results after simulations. The outcome is written in a text file with two columns: first column is the variables, in other words,

different pairs of parameters; second column is the data we focused on, specifically, current integral in this case. Current integrals are tractable from the text file.

(2) Related program: Python script

(3) Limitation: Due to the iteration and simplicity of script, only one category of results is recorded. It should be modified manually if users are interested in other aspects.

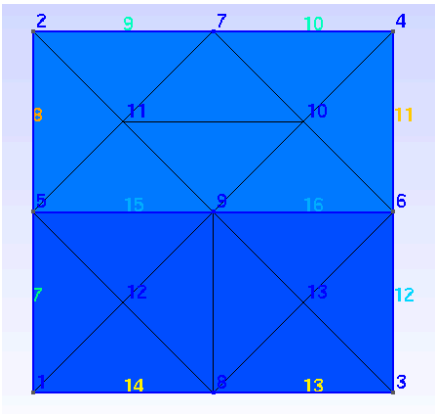
(4) Necessary files and folders: If the simulation runs correctly, there would be data files related to battery behavior, working under the additional package directory/inputFiles

(5) Input & Output

Input	Output
One kind of data in timeValues folder	Result.txt

4. Simple Working Example

In this section, an example is illustrated to explain the principle how the executable tackles with the Task 2-4. A simple squared model (length = 1) is given in Figure 1. We first define points, lines, interfaces and planes for the square model in Gmsh, then we define two different physical surfaces. As soon as we get the available input file, Gmsh is executed to generate the mesh. This is basically what the Task 2 does.



Squared example

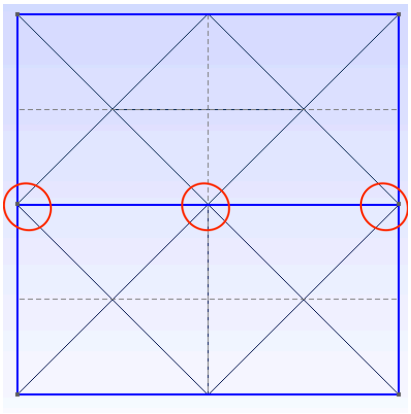


Figure 2. Nodes on the interface

Then execute the following code:

```
1.  python
2.  import os
3.  geo_file_path = "/home/tianxiangwang/Desktop/squared.geo"
4.  mesh_file_path = "/home/tianxiangwang/Desktop/mesh.msh"
5.  generatemesh = "gmsh" + geo_file_path + "-2 -o" + mesh_file_path
6.  # working directory
7.  working_Dir = "/home/tianxiangwang/Desktop/Addtional-Thesis/"
8.  # interface-elem location
9.  ciGen = "interface-generator/src/"
10. # working directory
11. os.chdir(working_Dir+ciGen)
12. output_file = “/home/tianxiangwang/Desktop/mesh.mesh”
13. splitmesh = ".interface-elem --mesh-file" + mesh_file_path +\
14. "--out-file"+ output_file +" --polycrystal"
15. os.system(splitmesh)
```

By doing so, we are able to easily grasp the working mechanism. As can been seen in the Figure 2, there are three nodes lying on the interface separating two parts, ciGen first duplicates each along the interface (one for each part) which is exactly what aims at for Task 3. In this simple example, we have 13 nodes in total, node 5, 6, 9 on the interface, shown in Figure 3. ciGen creates 3 new nodes 14, 15, 16 corresponding to the interface nodes. Reading the output file, we notice that 14, 15, 16 do not appear in any node groups, as shown in Figure 4, therefore our job for Task 4 is to group those nodes into node groups by finding the relation with the original nodes and adding them in the context of normal node group.

```
<Nodes>
1 0 0;
2 0 1;
3 1 0;
4 1 1;
5 0 0.5;
6 1 0.5;
7 0.5 1;
8 0.5 0;
9 0.5 0.5;
10 0.75 0.75;
11 0.25 0.75;
12 0.25 0.25;
13 0.75 0.25;
14 0 0.5;
15 1 0.5;
16 0.5 0.5;
</Nodes>
```

Figure 3. node coordinates after splitting

```
<NodeGroup name="12">
{2,5}
</NodeGroup>

<NodeGroup name="13">
{1,5}
</NodeGroup>

<NodeGroup name="14">
{2,4,7}
</NodeGroup>

<NodeGroup name="15">
{4,6}
</NodeGroup>

<NodeGroup name="16">
{3,6}
</NodeGroup>

<NodeGroup name="17">
{1,3,8}
</NodeGroup>

<NodeGroup name="18">
{5,6,9}
</NodeGroup>
```

Figure 4. Node group category

5. Symmetric trenches working example

Now the simple squared example can be expanded into a more complicated symmetric geometry of battery with several parameters controlling to better prove task 4-6. The Task 1 is substituting the original parameters with the user's defined lists creating a series of Gmsh files as input, shown in Figure 5. Then based on those modified files, which are actually task 2 and 3, the executable generates mesh and splits mesh using the same scheme as mentioned above.

```
// Characteristic length
cl = 0.8;
clCorners = 0.3;

// Height of the trenches
/*h = 50.0;*/
h = 25.0;

// Thickness of the Active material trenches
tac = 10.0;

// Thickness of the Electrolyte layers
tel = 10.0;

// Radius of curvature of the electrode ends
Rc = 1.0;
/*Rc = 4.8;*/
```

Figure 5. Prescribe parameters

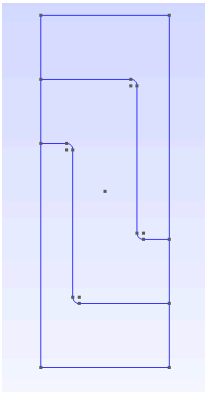


Figure 6. Geometry of battery

By executing the script, we are able to obtain the mesh file after splitting the nodes on the interface, here task 4 is explained with detailed node labels. For the sake of simplicity, we take a fixed parameter for example, we choose a character length of 2.5 to substitute 2.0 in the original Gmsh file, and height of 20, same as the original one, refer to Figure 7.

```
Which parameter as first variable:(cl,h,tac,tel,Rc):
cl
The elements in the first list:(use , to sperate)
2.5
which parameter as second varible:(cl,h,tac,tel,Rc):
h
The elements in second list:(use , to sperate)
20
original data 1 to be substituted
with format like: cl = 2.0:
cl = 2.0
original data 2 to be substituted
with format like: h = 20.0:
h = 20.0
```

Figure 7. Input interface

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
393
1 0 0 0
2 20 0 0
3 20 50 0
4 0 50 0
5 0 30 0
6 20 10 0
7 20 20 0
8 0 40 0
```

Figure 8. Node amount before splitting

```
<NodeGroup name="intAnElAnode">
{394, 395, 398, 399, 400, 401, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420}
</NodeGroup>

<NodeGroup name="intCatElCathode">
{396, 397, 402, 403, 404, 405, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435}
</NodeGroup>
```

Figure 9. Newly added interface nodes

Initially, Gmsh generates 393 nodes in total for the given geometry, which means the new interface nodes must start from 394, and its amount should be equal to the node number lying on the interfaces because it is a duplication of the original nodes. Verified by Figure 9, there are 42 new interface nodes having the same coordinates as the original interfaces, the node

number follows a rigid sequence matching with each other. It proves the executable is capable to find the missing nodes and compile them into new node groups. Right after splitting meshes, as can be seen in Figure 8, node 5,6,7,8 appear both in left/right edge group and interface node group where constrains are applied. The script eventually gets rid of these nodes on left/right edge. If the executable runs correctly, the folder: `workDir/inputFiles/timeValues/` will be filled with outcomes of simulation, result.txt records the current integral with respect to different parameters.

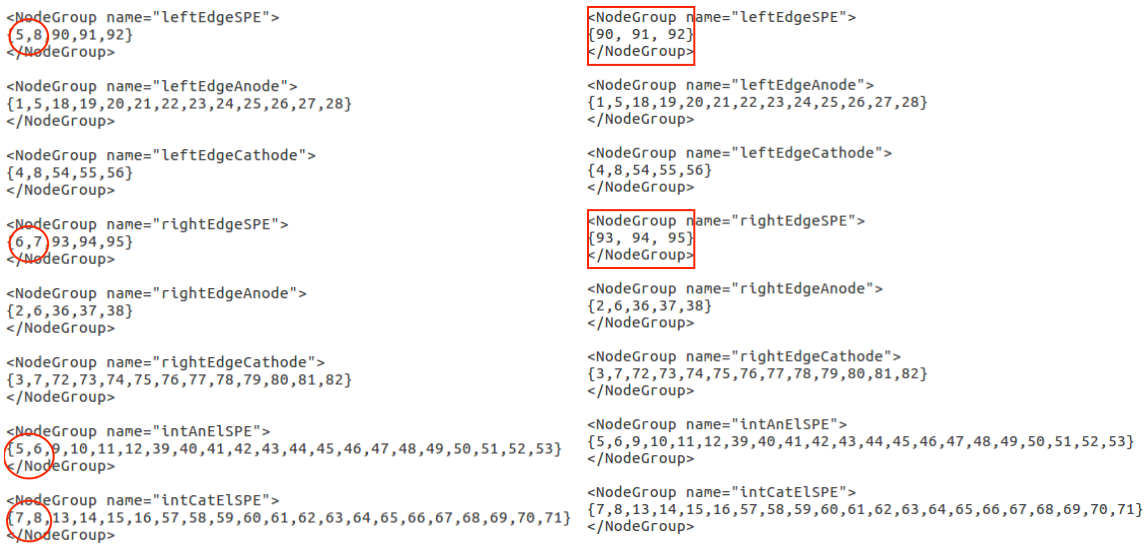


Figure 10. Difference after eliminating the overlapped nodes

Asymmetric examples are provided here, and its according python file.

6. Reference

[1] *An open source program to generate zero-thickness cohesive interface elements Vinh Phu Nguyen. Advances in Engineering Software 74 (2014) 27–39*

[2] *https://jive.dynaflow.com/*

Below is the code with detailed explanation:

```
# =============Main================

# =============Input===============

# Interface

var1 = raw_input("Which parameter as first variable:(cl,h,tac,tel,Rc):\n")
s1 = raw_input("The elements in the first list:(use , to sperate) \n")
var2 = raw_input("which parameter as second varible:(cl,h,tac,tel,Rc):\n")
s2 = raw_input("The elements in second list:(use , to sperate) \n")

va1 = "\""+var1+"\""
va2 = "\""+var2+"\""

# para is the original string in Gmsh file which will be substituted later
para1 = raw_input("original data 1 to be substituted\nwith format like: cl = 2.0:\n")
para2 = raw_input("original data 2 to be substituted\nwith format like: h = 20.0:\n")

# items1 is the list of the first varible read from user's input
# items2 is the list of the second varible read from user's input
items1 = map(str, s1.split(','))
items2 = map(str, s2.split(','))

sub1 = var1+" = {}"
sub2 = var2+" = {}"

# To name the file, get rid of decimal if there is one
# This is due to ciGen could not recognize a file name which contains decimal

spl1 = []
spl2 = []


# Create .geo input files
# Substitute decimal with space: 0.5.mesh will be 05.mesh

# First loop for the first variable
# a1 & a2 (without decimal) will be used to name mesh file
for i in items1:
    a1 = str(i).replace('.','')
    spl1.append(a1)

    # Substitute parameters with user's input
    with open(workDir+Filename1+".geo","r") as gmshfile:
        with open(paraDir+var1+str(i)+".geo","w") as overwrite:
            for line in gmshfile:
                overwrite.write(line.replace(para1,sub1.format(i)))

    # Second loop embedded in first loop for the second variable
    for j in items2:
        a2 = str(j).replace('.','')
        spl2.append(a2)

        # Replace 'cl = 2.0' with 'cl = {}'.format(i)
        with open(paraDir+var1+str(i)+".geo","r") as gmshfile:
            with open(paraDir+var1+str(i)+var2+str(j)+".geo","w") as overwrite:
                for line in gmshfile:
                    overwrite.write(line.replace(para2,sub2.format(j)))

```

```
#==========================Generate Mesh=============================

# Generate mesh files (2D, triangle elements) for new .geo file

        GenerateMesh = "gmsh "+paraDir+var1+str(i)+var2+str(j)+\
        ".geo"+" -2 -o "+simuDir+"mesh"\
        +str(a1)+str(a2)+".msh"
        os.system(GenerateMesh)

        print ("For value:"+var1+str(i)+var2+str(j)+" Generate Mesh done...")

#==========================Split Mesh=============================

# Splitting the interface nodes

        # Specify the working directory

        os.chdir(simuDir)

        # Go through Cigen for each file

        SplitMesh = " ../" + ciGen +" --mesh-file mesh"\
            +str(a1)+str(a2)+".msh"+" --out-file "+simuDir+"mesh"+str(a1)+str(a2)\
            +".mesh --polycrystal"

    os.system(SplitMesh)

        print ("For value:"+var1+str(i)+var2+str(j)+" Split mesh done...")

        # Replace the name according to the gmsh file
        with open(simuDir+"mesh"+str(a1)+str(a2)+".mesh",'r') as file1:
            filedata = file1.read()

    # This could be changed if necessary
    # Users should assure the number of nodegroups are matching with its name
        filedata = filedata.replace\
            ('<NodeGroup name=\"1\">','<NodeGroup name=\"leftEdgeCathode\">')\
                .replace('<NodeGroup name=\"2\">','<NodeGroup name=\"leftEdgeSPE\">')\
                .replace('<NodeGroup name=\"3\">','<NodeGroup name=\"leftEdgeAnode\">')\
                .replace('<NodeGroup name=\"4\">','<NodeGroup name=\"rightEdgeCathode\">')\
                .replace('<NodeGroup name=\"5\">','<NodeGroup name=\"rightEdgeSPE\">')\
                .replace('<NodeGroup name=\"6\">','<NodeGroup name=\"rightEdgeAnode\">')\
                .replace('<NodeGroup name=\"7\">','<NodeGroup name=\"yTop\">')\
                .replace('<NodeGroup name=\"8\">','<NodeGroup name=\"yZero">')\
                .replace('<NodeGroup name=\"9\">','<NodeGroup name=\"intCatElSPE\">')\
                .replace('<NodeGroup name=\"10\">','<NodeGroup name=\"intAnElSPE\">')\
                .replace('<ElementGroup name=\"11\">','<ElementGroup name=\"Anode\">')\
                .replace('<ElementGroup name=\"12\">','<ElementGroup name=\"Cathode\">')\
                .replace('<ElementGroup name=\"13\">','<ElementGroup name=\"SP\">')\

        with open(simuDir+"mesh"+str(a1)+str(a2)+".mesh",'w') as file2:
            file2.write(filedata)

        # Rename part done

    # Extract two lists from interfaces between both anode/cathode and electrolyte
        with open(simuDir+"mesh"+str(a1)+str(a2)+".mesh") as inputfile,\
        open(simuDir+"Lists"+str(a1)+str(a2)+".txt",'w') as outputfile:
            copy = False
            for line in inputfile:
                if line.strip() == "<NodeGroup name=\"intAnElSPE\">" or \
```

```
                line.strip() == "<NodeGroup name=\"intCatElSPE\">":
                    copy = True
                elif line.strip() == "</NodeGroup>":
                    copy = False
                elif copy:
                    outputfile.write(line)

    # Get rid of redundant branket at the begining and end
        with open(simuDir+"Lists"+str(a1)+str(a2)+".txt",'r') as readfile :
            filedata = readfile.read()
        filedata = filedata.replace('{',' ').replace('}',' ')
        with open(simuDir+"Lists"+str(a1)+str(a2)+".txt",'w') as writefile :
            writefile.write(filedata)

#====================Create new nodesgroup for newly added interface nodes=====================

        with open(simuDir+"Lists"+str(a1)+str(a2)+".txt") as f:
            lst = []
        # Read list from previous files

            for line in f:
                lst.append(line.strip().split(','))

    # lst contains two elements, the first item is node numbers for anode interface (SPE side)
        # the second item is node numbers of cathode interface (SPE side)

        anode = []
        anode = lst[0]

        cathode = []
        cathode = lst[1]

        interfacenodes = []
        interfacenodes = anode + cathode

        # Ascendant list
        interfacenodes.sort(key=int)

        size = len(anode)

        # Create equal number for new nodes with SPE node groups
        # The new node number should start with total amount of nodes
        # Read total amount of nodes from .msh files, the amount is between "$Nodes" and "1 0 0 0"

        with open(simuDir+"mesh"+str(a1)+str(a2)+".msh",'r') as infile:
            copy = False
            for line in infile:
                if line.strip() == "$Nodes":
                    copy = True
                elif line.strip() == "1 0 0 0":
                    copy = False
                elif copy:
                    nodesnumber = line

        # Total number of newly created nodes
        # newnodes is an ascendant list starting from the amount of node number
        # Has equal number of nodes with anode/cathode interface
        end = int(nodesnumber) + size*2 +1
        newnodes = range(int(nodesnumber)+1,end)

        # Find a matching relation between original nodes and new nodes
```

```
        dct = dict((a,b) for a, b in zip(interfacenodes, newnodes))


        # Write down the newly added nodes in new text files
        with open(simuDir+"Newnodegroups"+str(a1)+str(a2)+".txt",'w') as f:
            newanode = []

            # Separate the nodes to corresponding lists
            # Print out the outcome into text files with the prescribed manner

            for item in anode:
                newanode.append(dct[item])
            print("<NodeGroup name=\"intAnElAnode\">",\
                  "\n",newanode,"\n","</NodeGroup>","\n",sep='',file=f)

            newcathode = []
            for item in cathode:
                newcathode.append(dct[item])
            print("<NodeGroup name=\"intCatElCathode\">",\
                  "\n",newcathode,"\n","</NodeGroup>",sep='',file=f)

            print("For value:"+var1 +str(i)+var2+str(j)+" New interface nodes added...")


        #[] should be changed into {} to have same manner

        with open(simuDir+"Newnodegroups"+str(a1)+str(a2)+".txt",'r') as file :
            filedata = file.read()
            filedata = filedata.replace('[','{').replace(']','}')

        with open(simuDir+"Newnodegroups"+str(a1)+str(a2)+".txt",'w') as file :
            file.write(filedata)

#================================Append files====================================

        # Combine files together, add the new text content to the original gmsh file
        # Outputs are the final files incluing two newly added lists of nodes

        with open(simuDir+"mesh"+str(a1)+str(a2)+".mesh",'r') as readfile:
            with open(simuDir+"Finishadding"+str(a1)+str(a2)+".txt",'w') as outfile:
                for line in readfile:
                    outfile.write(line)

        with open (simuDir+"Newnodegroups"+str(a1)+str(a2)+".txt",'r') as infile:
            with open(simuDir+"Finishadding"+str(a1)+str(a2)+".txt",'a') as outfile1:
                for line in infile:
                    outfile1.write(line)

#========================Delete overlapped node number=========================

    # Nodes should not be shared by both interfaces and left/right edges
    # Read the left and right edges node number
        with open(simuDir+"mesh"+str(a1)+str(a2)+".mesh") as inputfile,\
        open(simuDir+"Overlappinglist"+str(a1)+str(a2)+".txt",'w') as outputfile:
            copy = False
            for line in inputfile:
                if line.strip() == "<NodeGroup name=\"leftEdgeSPE\">" or \
                line.strip() == "<NodeGroup name=\"rightEdgeSPE\">":
                    copy = True
                elif line.strip() == "</NodeGroup>":
                    copy = False
```

```
243.                 elif copy:
244.                     outputfile.write(line)
245.
246.     # Extract the list, get rid of {} at the beginning and end
247.         with open(simuDir+"Overlappinglist"+str(a1)+str(a2)+".txt",'r') as readfile :
248.             filedata = readfile.read()
249.             filedata = filedata.replace('{','').replace('}','')
250.
251.         with open(simuDir+"Overlappinglist"+str(a1)+str(a2)+".txt",'w') as writefile :
252.             writefile.write(filedata)
253.
254.         # nlst consists of two items, first is node number on the left edge
255.         # second is node number on the right edge
256.         with open(simuDir+"Overlappinglist"+str(a1)+str(a2)+".txt") as f:
257.             nlst = []
258.             for line in f:
259.                 nlst.append(line.strip().split(','))
260.
261.         # Get rid of the overlapping one on left/right edge
262.         leftresult=[]
263.         left=[]
264.
265.         for n in nlst[0]:
266.             if n not in interfacenodes:
267.                 leftresult.append(n)
268.         for n in leftresult:
269.             n = int(n)
270.             left.append(n)
271.
272.         rightresult=[]
273.         right = []
274.
275.         for n in nlst[1]:
276.             if n not in interfacenodes:
277.                 rightresult.append(n)
278.         for n in rightresult:
279.             n = int(n)
280.             right.append(n)
281.
282.         # Find the original edge groups
283.     # Substituted by the deleted lists
284.     # Start with the left edge
285.         with open(simuDir+"Finishadding"+str(a1)+str(a2)+".txt",'r') as inputfile:
286.             with open(simuDir+"Deleteoverlapping"+str(a1)+str(a2)+".mesh",'w') as outputfile:
287.                 copy = False
288.                 for line in inputfile:
289.                     if copy == False:
290.                         outputfile.write(line)
291.                     if line.strip() == "<NodeGroup name=\"leftEdgeSPE\">":
292.                         copy = True
293.                     elif line.strip() == "</NodeGroup>":
294.                         copy = False
295.                     elif copy:
296.                         print(left,"\n","</NodeGroup>",sep='',file=outputfile)
297.
298.         # Then the right edge
299.         # As soon as this block is executed,  all requirements should be met for simulation
300.         with open(simuDir+"Deleteoverlapping"+str(a1)+str(a2)+".mesh",'r') as inputfile:
301.             with open(simuDir+var1+str(i)+var2+str(j)+".mesh",'w') as outputfile:
302.                 copy = False
303.                 for line in inputfile:
```

```
                    if copy == False:
                        outputfile.write(line)
                    if line.strip() == "<NodeGroup name=\"rightEdgeSPE\">":
                        copy = True
                    elif line.strip() == "</NodeGroup>":
                        copy = False
                    elif copy:
                        print(right,"\n","</NodeGroup>",sep='',file=outputfile)

    # Get rid of []
        with open(simuDir+var1+str(i)+var2+str(j)+".mesh",'r') as outputfile:
            filedata = outputfile.read()
            filedata = filedata.replace('[','{').replace(']','}')

        with open(simuDir+var1+str(i)+var2+str(j)+".mesh",'w') as file:
            file.write(filedata)
======================Move and clean up files==========================

        # Move files generated above towards the corresponding folders
        # This is to create multiple folders
        # Each fold contains ultimate mesh file
        os.makedirs(simuDir+var1+str(i)+var2+str(j))

        # The following codes is to move files

        finalfile = simuDir+var1+str(i)+var2+str(j)+"/"+var1 +str(i)+var2+str(j)+".mesh"

        os.remove(simuDir+"Newnodegroups"+str(a1)+str(a2)+".txt")
        os.remove(simuDir+"mesh"+str(a1)+str(a2)+".msh")
        os.remove(simuDir+"Finishadding"+str(a1)+str(a2)+".txt")
        os.remove(simuDir+"mesh"+str(a1)+str(a2)+"-interface.mesh")
        os.remove(simuDir+"Lists"+str(a1)+str(a2)+".txt")
        os.remove(simuDir+"mesh"+str(a1)+str(a2)+".mesh")
        os.remove(simuDir+"Overlappinglist"+str(a1)+str(a2)+".txt")
        os.remove(simuDir+"Deleteoverlapping"+str(a1)+str(a2)+".mesh")
        os.rename(simuDir+var1+str(i)+var2+str(j)+".mesh",finalfile)

#*******************************Simulation*******************************

        # working flow:
        # 1.move files into Meshes folder as input mesh
        # 2.modify the string in inputfiles input.dat
        # 3.go to the directory: cd /home/../src/
        # 4.make clean, make
        # 5.go to the inputfiles folder
        # 6.run command ../src/cellParaview input.pro
        # 7.read the current

        # 1
        copyfile(finalfile,workDir+"Meshes/"+var1+str(i)+var2+str(j)+".mesh")

        # 2
        sentence1 = "<Include source = \"../Meshes/trench2D.mesh\"/>"
        sentence2 = "<Include source = \"../Meshes/"+var1+str(i)+var2+str(j)+".mesh\"/>"

        with open(workDir+"input.dat",'r') as readfile:
            with open(workDir+Simuinput,'w') as writefile:
               for line in readfile:
                   writefile.write(line.replace(sentence1,sentence2))

        # 3-6
```

```
365.         os.chdir(workDir+"src/")
366.
367.         os.system("make clean")
368.         os.system("make")
369.
370.         os.chdir(workDir+"inputFiles/")
371.         os.system("../"+paraview)
372.
373.         # 7
374.         # The current value is from the 4th character to 12th character
375.         with open(current+"currentIntegral.txt",'r') as currentdata:
376.             currentdata.seek(4)
377.             data = currentdata.read(12)
378.         with open(result,'a') as write_data:
379.             print(str(i), data,sep='',file=write_data)
380.             ## print(var1+str(i)+var2+str(j),"\n",data,sep='',file=write_data)
381.
382.         os.remove(workDir+"Meshes/"+var1+str(i)+var2+str(j)+".mesh")
383.
384.         print("For value:"+var1 +str(i)+var2+str(j)+" Read the current integration done...")
385.
386.
387.     os.remove(paraDir+var1+str(i)+".geo")
388.
389. print ("All work is done")
```