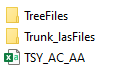
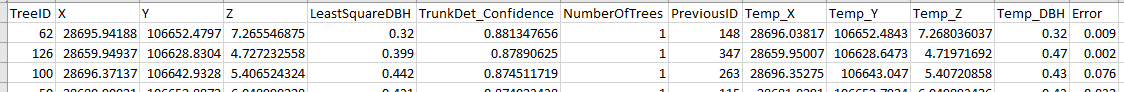
Knowledge Transfer – Open3D

Date: 9th February 2022 Session 1,

Algorithm Objective: Gives us tree segments, tree files, trunk files and a main .csv file.





Contents of TSY\_AC\_ACC.csv:

1. TreeID: Unique tree id for every tree
2. X,Y,Z: co-ordinates of tree(x,y) and height at which the diameter(DBH) is marked.
3. LeastSquareDBH: The diameter of tree (meters)
4. TrunkDet\_Confidence:
5. NumberofTrees:
6. PreviousID:
7. Temp\_X:
8. Temp\_Y:
9. Temp\_Z:
10. Temp\_DBH:
11. Error:

Our Objective: To see that whether the algorithm has done its work right OR the output generated by it is right or wrong.

Procedure of the program:

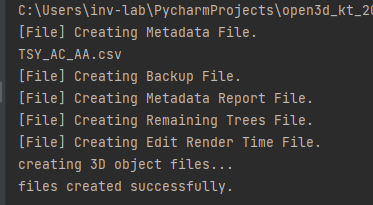
Visualize and give responses

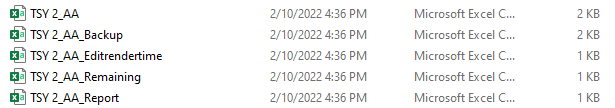
Convert .laz/.las to .ply (because Open3d cannot read .laz/.las files

Generate Reports

Generate Reports

1. Create environment in Pycharm.
2. Install the requirements from requirements.txt file
3. Give i/p to the variable “forestName” variable in the “config.py” file.
4. Run the main “Open3d\_2.py” file.
5. Reports will be created.



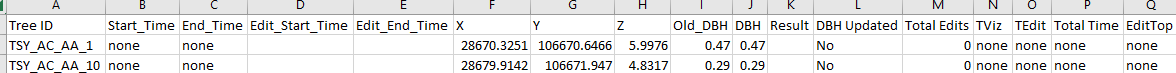


In total 4 files would be created(not counting the backup file):

1. TSY 2\_AA
2. TSY 2\_AA\_Backup
3. TSY 2\_AA\_Editrendertime
4. TSY 2\_AA\_Remaining
5. TSY 2\_AA\_Report

“TSY 2” here our forest name, and all the nomenclature is defined by the forestName only.

1. TSY 2\_AA:



Tree\_ID: Tree ID in relation to the forest name and the files in the Tree folder.

Start\_Time: Time when we start a viz on a tree

End\_Time: Time when we end a viz on a tree.

Edit\_Start\_Time: Time when Editing has begun on a tree, None when no editing takes place.

Edit\_End\_Time: End time of editing.

X,Y,Z: Same co-ordinates as the main .csv file.

Old\_DBH: Diameter dimensions from the main .csv.

New\_DBH: Diameter dimensions after editing.

Result: Response given to each vis.

DBH\_Update: Boolean, depends on whether updated or not.

Total\_Edits: No. of times edit window is opened.

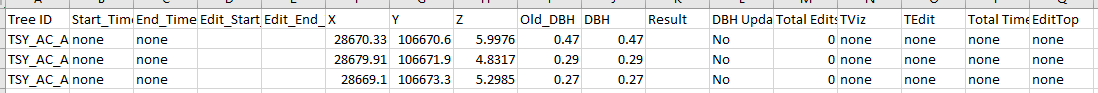
TViz: time taken for vis. (in seconds)

TEdit: time taken for editing.

Total\_Time: total time for viz.

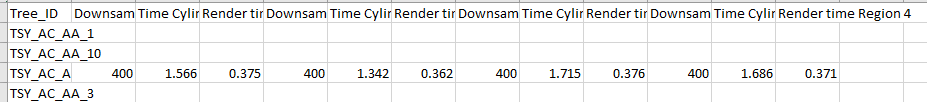
Edit\_top: what response we give in editing mode.

1. TSY 2\_AA\_Backup:

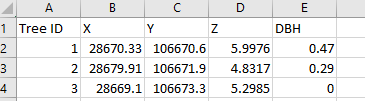


Backup file for the previously mentioned file, required because the file has to be accessed many times and this can sometimes corrupt the file itself. THE BACKUP FILE IS BACKEDUP AFTER EVERY 5 ITERATIONS OF TREES.

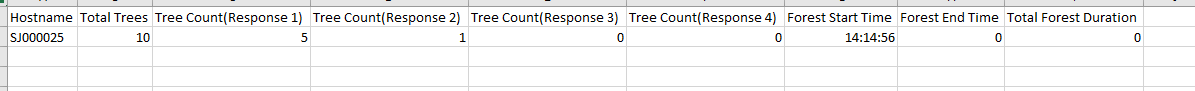
1. TSY 2\_AA\_Edit\_RenderTime: Only trees which are edited, information of those trees will be stored.



1. TSY\_AA\_Remaining: Only response 1 & 2 will be stored here.



1. TSY\_AA\_Report: Report of the whole forest would be stored.

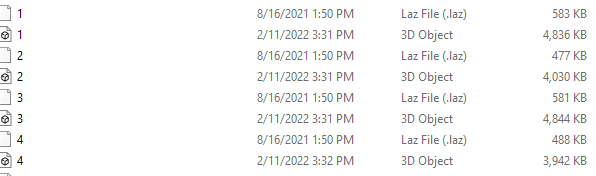


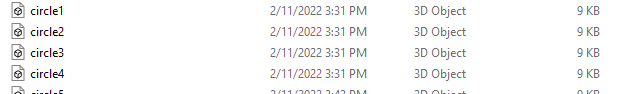
Convert .laz/.las to .ply

1. Inside the config.py =>
   1. Variables –
      1. TreeFileFormat => .laz/.las, requires manual updating if files in either format.
      2. plyFileFormat => .ply

Now give i/p of the first tree\_id from the TSY\_AA to the variable StartCreationfrom., and run the Open3d\_2.py file again.

O/p => .ply files from the starting point will be generated along with circle files in the “Tree files” folder of the forest.





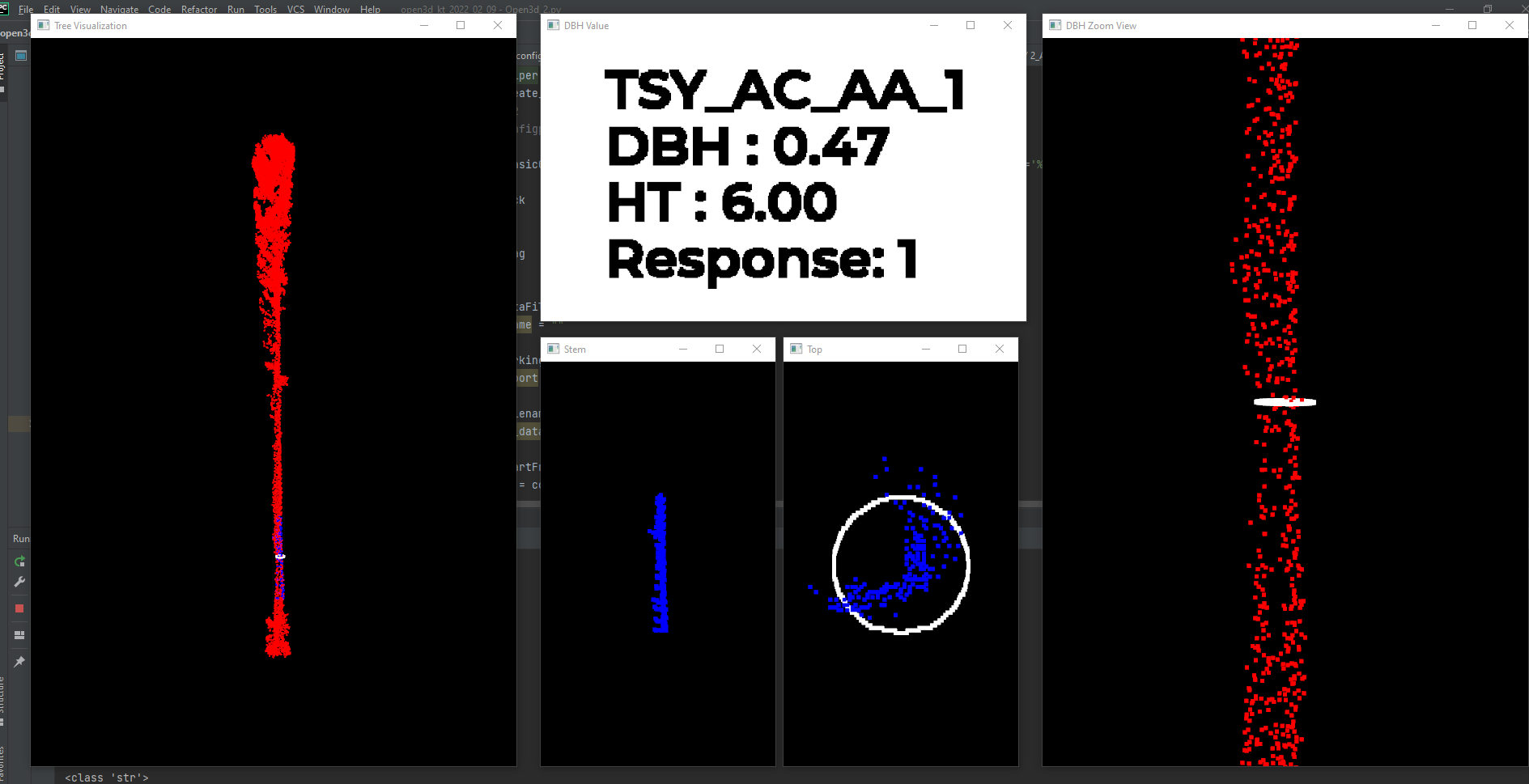
Visualize and give responses

Now in the config.py file again give the 1st tree\_id to variable “StartFrom”, to specify that from this tree\_id, we have to begin our viz & also change “files3dcreated” to “True” (before it should be False).



Now again run Open3d\_2.py file to begin the visualization.

1. Tree ID
2. DBH or Diameter
3. HT : Height from where DBH is marked.
4. Response: what we have given as response to the tree.



DBH is viz’ed here.

Trunk/stem is also visualized here.

Zoomed in version of the main tree with focus on DBH.

Top view/2D view of DBH

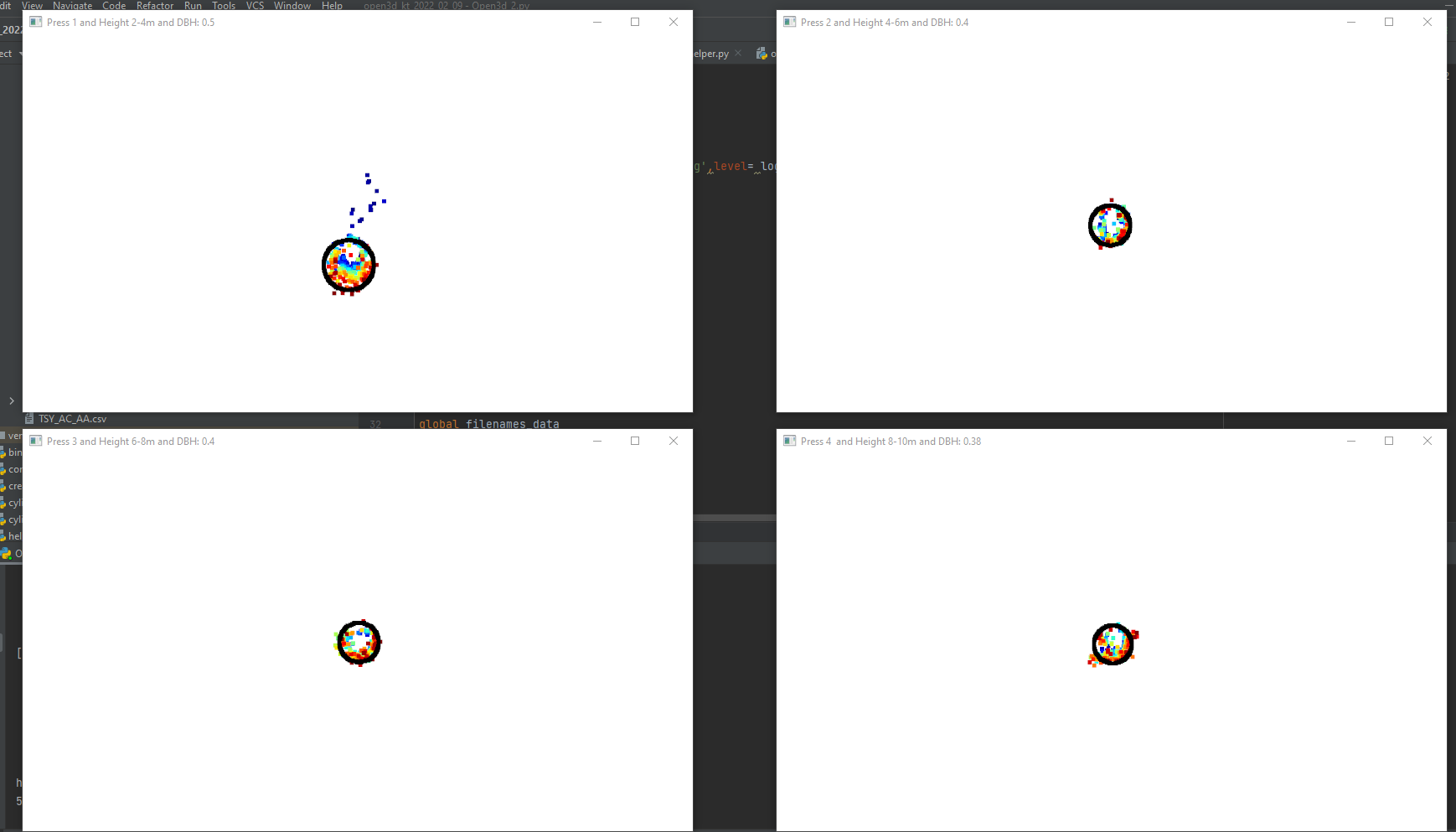
Stem/trunk(portion range where DBH marked)

Main tree vis

Based on our obs., we can specify 3 responses:

* 1 => If we think that DBH marked is correct, also moves to next tree automatically after storing response.
* 2 => If we think that marked DBH is False, then we get 2 options :
  + Esc => skip to next tree.
  + E => editing window.

Editing window:

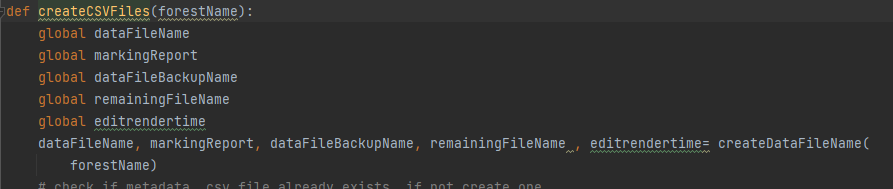


This is the editing window in which we have option for choosing any out of 4 options provided. IF WE FEEL THAT NONE OF THE 4 OPTIONS SATISFY OUR REQUIREMENTS THEN WE CAN CHOOSE TO PRESS TAB, WHICH MEANS THAT NEW HEIGHT WILL BE MARKED AS ZERO.

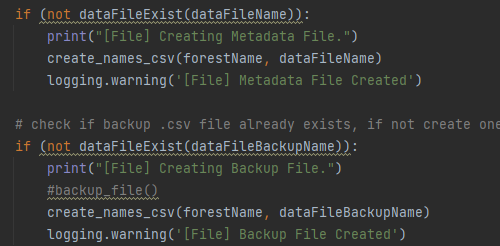
* 3 => If the algorithm has detected something which is not a tree, but marked it as a tree, this response would be given.

Flow of Program

1. Start by giving i/p to the variable “forestName” in config.py file.
2. Then in the main Open3d\_2.py file, the first function call is for “createcsvFile” function in Helper.py file.
   1. createcsvFile:
      1. Defines nomenclature for each report.
      2. Also defines the basic structure of the reports.
      3. Then the files are created individually.

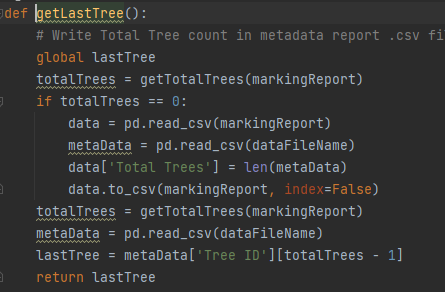


Nomenclature and structure defining function.

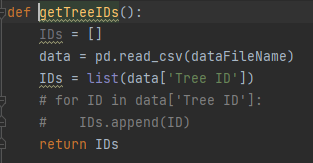


Creating csv files like this in the createcsv function.

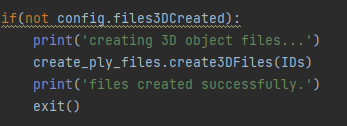
1. Then in the main Open3d\_2.py file, next function call is for storing the last tree id. This is done because while visualizing, if we encounter the last tree, and still continue with the viz, the program while hang or show blank windows because of no .ply file or data.



1. Then the next function call is of storing the “Tree\_ID’s” in a list. This list will come in handy when we are converting the .laz/.las files to .ply files.

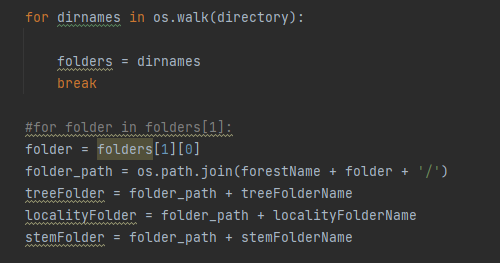


1. Next our program heads towards creation of .ply files from the .laz files. This begins by first giving the first tree id from the reports just created to the variable “StrartCreationFrom” in the “Config.py” file.
2. Then the function of the creation of creation of .ply files is called from the “create\_ply\_files.py” file.

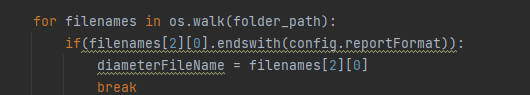


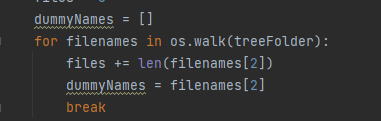
Function call for converting .laz to .ply.

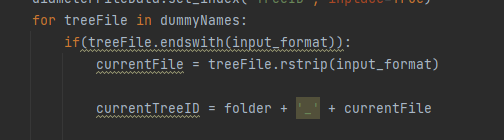
1. Define the path for each folder storing the .laz files, ie. , tree folder, stem folder.



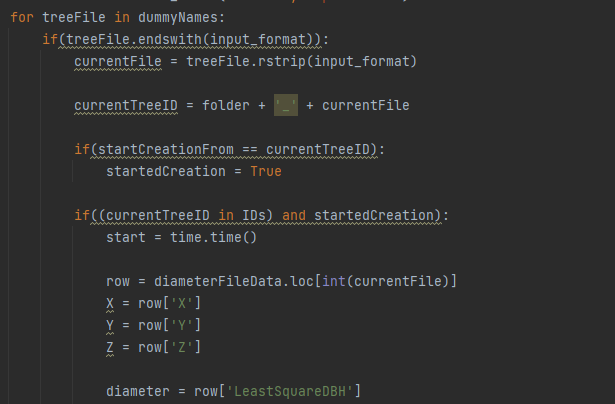
1. Store the main csv file from the algorithm in a variable for later access.



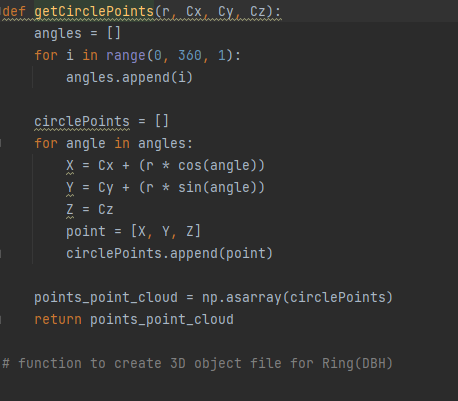
1. Now store the contents of the tree folder inside a list named so that we can access them directly inside that list.  
   
2. Now check that if any file inside that list is with extension “.laz”, if yes, then strip the extension and save only the file name in a variable.

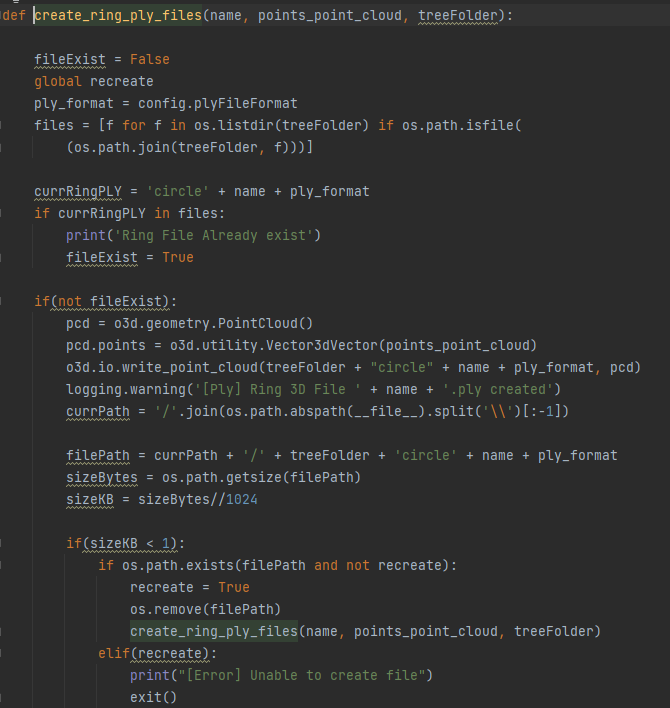


1. Now check if the current file name formed is there inside the ID’s, if yes then store the data for that tree id from the main csv of algorithm.



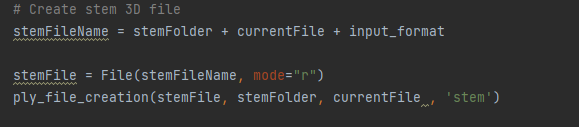
1. Now coming to the circle for the tree, we have to make the visualization for that on our own. We have the x,y,z coordinates for the center, now we can make a circle out of these points by rotation the center point 360 degrees. This process is done inside a function called “getCirclePoints”

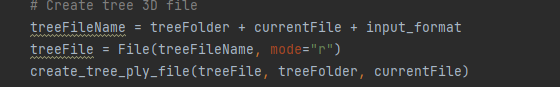


1. Now we will make the .ply for this circle by again calling another function called “create\_ring\_ply\_files ”. 

This function also check if our file is made properly or not by checking the file size. If the file size is less than 1KB, then the file is more likely to be broken.

1. This same process is then repeated for all the other folders (tree folder and stem folder).





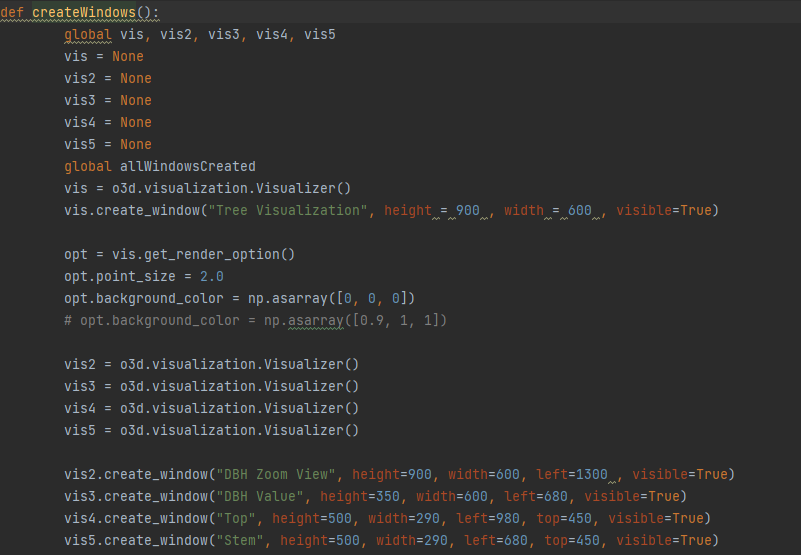
1. The function has now executed completely and then control continues with the next line of Open3d\_2.py file, which is print statement, then the program will exit.

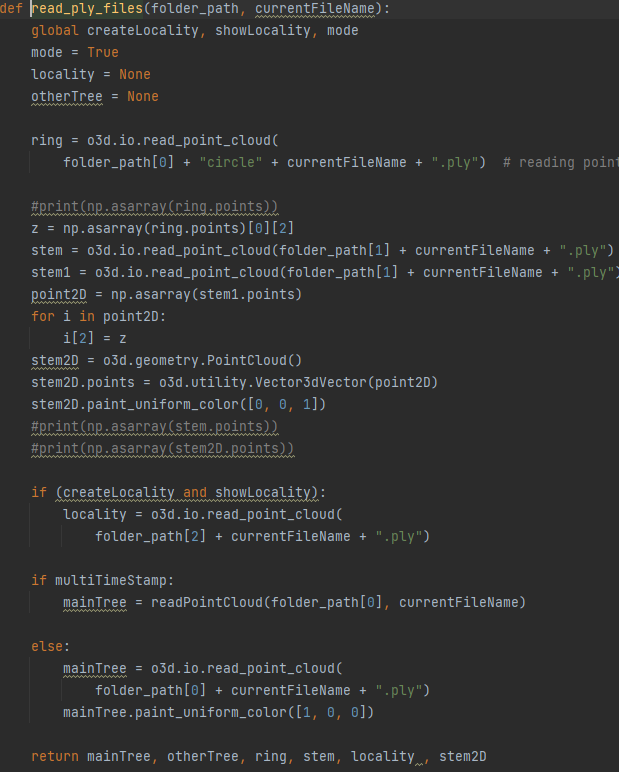


Visualization of the created files

The program has now entered its final leg, that is the visualization. To begin this we first give the input to the variable “startfrom” in the config.py. Then run the program.

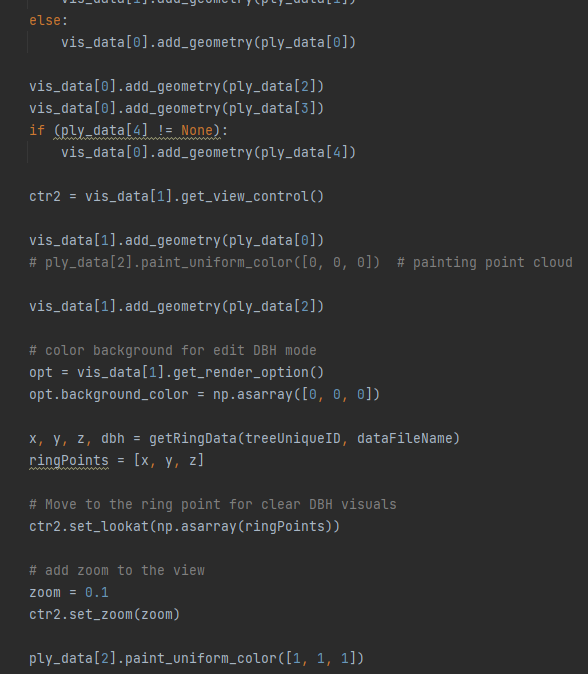
1. The flow begins from the main function inside the Open3d\_2.py file, by defining the visualization windows.



1. Then the same process of defining the folder paths and dummylist is defined for the accessing of the .ply files.
2. Then from these defined path, our ply files are read for visualizing them in the defined windows. The reading part is again done with the help of a function. 
3. Then we store all the files and windows inside a list for later use, and color our rings in white.



1. Then with the help of a function, we visualize all our files inside there respective windows and give them colors accordingly.

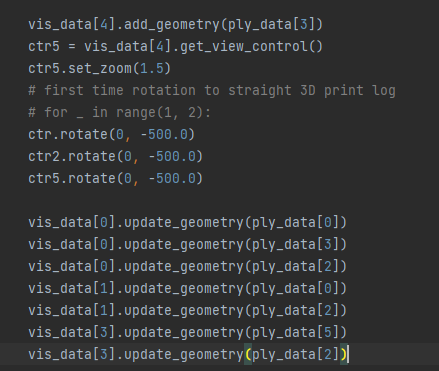


Code for storing the main tree and assigning them colors. This fashioned code is repeated for every viz window.

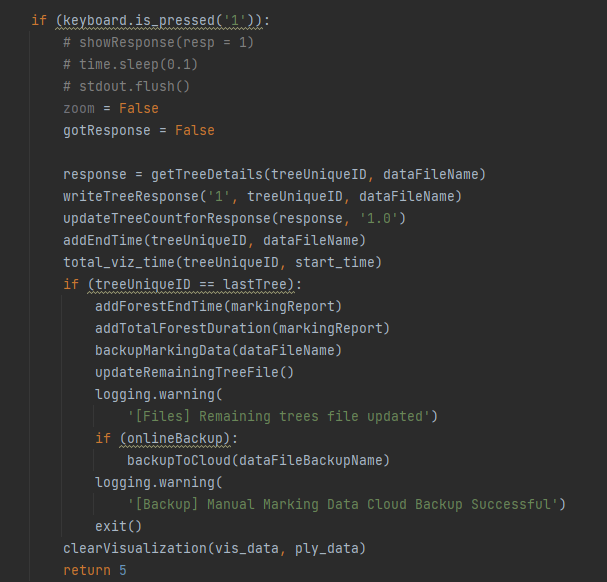


This code is for defining the tree detail window content.

1. And then we keep on updating our viz. windows regularly.

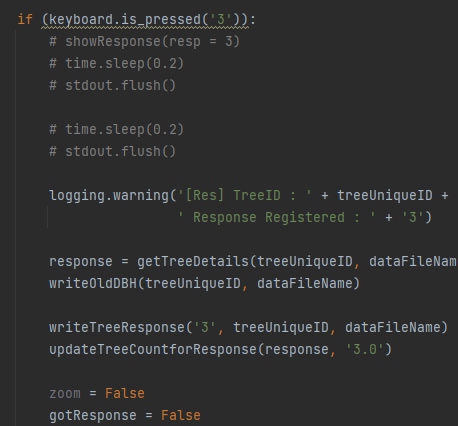


1. Now we come to the response part of the code, where we have mapping for button 1,2,3,b,esc & z.
2. When we press 1, it means that our tree and dbh is marked properly and requires no updation or changes.

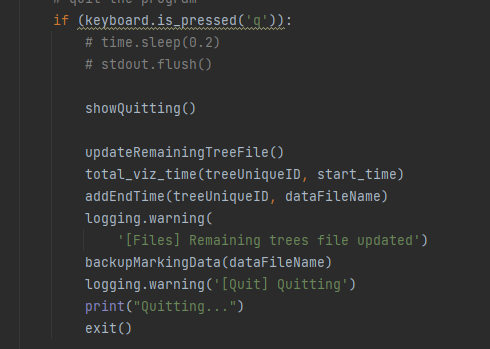


Pressing 1 also automatically changes the tree to the next after capturing the response.

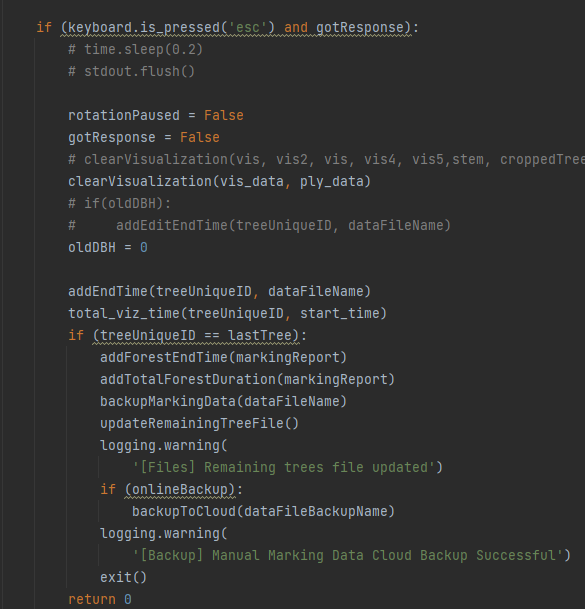
1. Pressing 3 means that the algorithm has detected a tree, but in reality it is not a tree but an environmental noice, inshort a False-Positive.



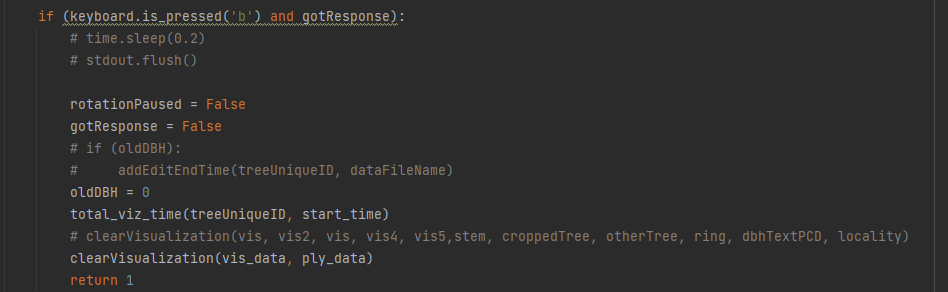
1. Pressing “q” will quit the viz,destroying every window and exiting the program.



1. Pressing “esc” will move to the previous tree.



1. Pressing “b” will move to the next tree by clearing the window and loading the next tree.

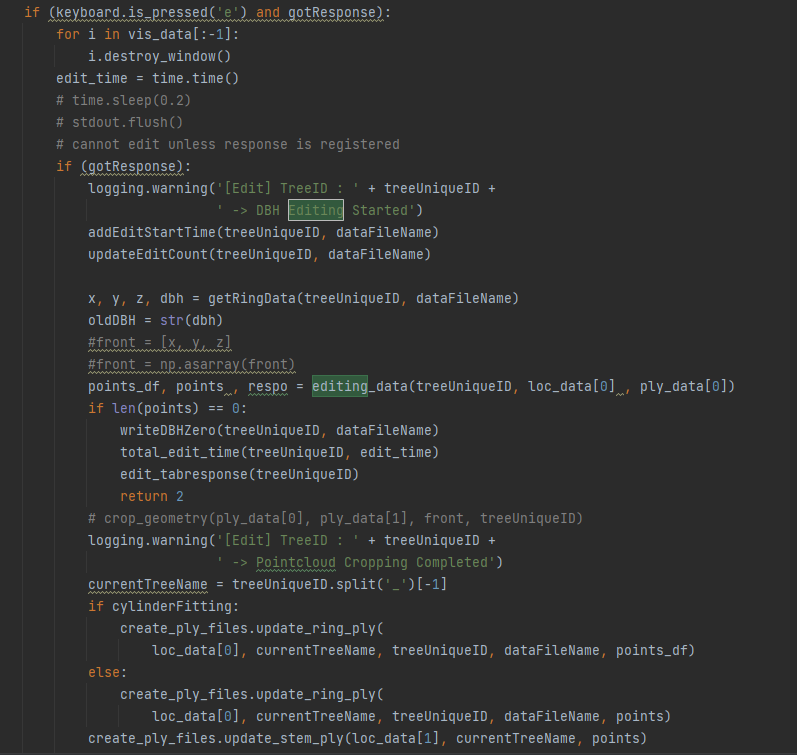


1. Pressing 2 will first result in turning a flag named “RotationPaused” to True, which will result in pausing the tree rotation.

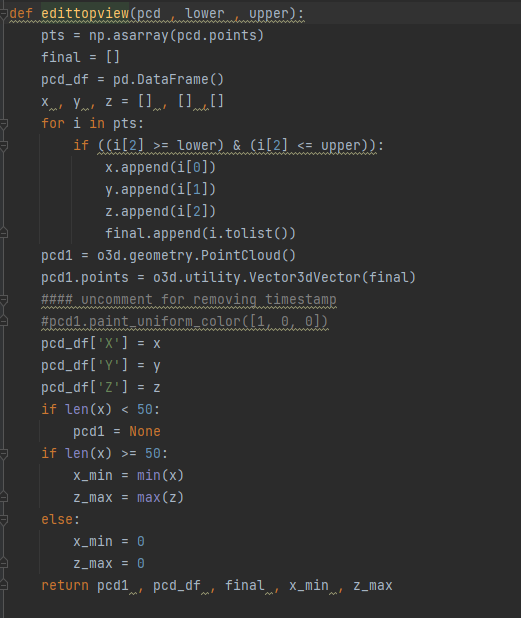


After pressing 2 we have option for going to the editing mode by pressing “e”.

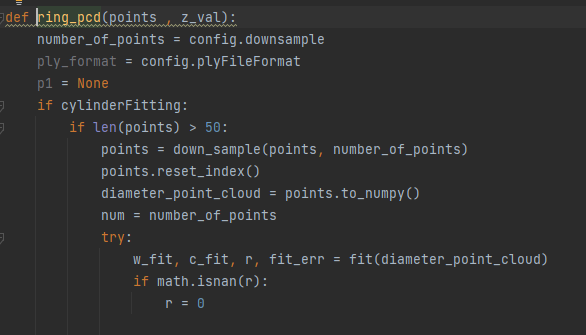
1. Pressing “e” will open the Editing mode.



By pressing “e” our circle points go through a process of Cylindrical fitting on the stem range given (2-4,4-6,6-8,8-10).

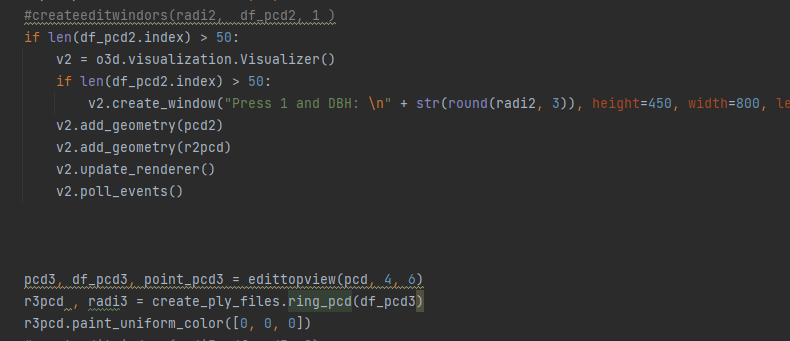


Defining the data points for Cylindrical fitting on stem.



Giving the points to Cylindrical fitting.

1. Define all the window and give the ply files to the windows.

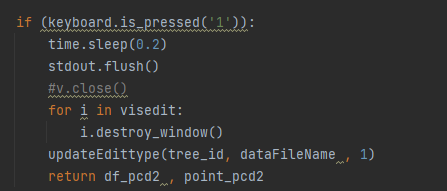


This same process is repeated for all the windows.

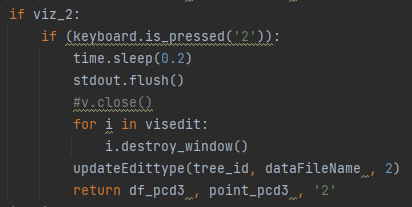
1. Now inside the window if “Tab” is pressed, this means that we are not happy with the Cylindrical fitting results and still think that DBH marked si wrong, so we give “Tab” as input resulting in height as “Zero”.

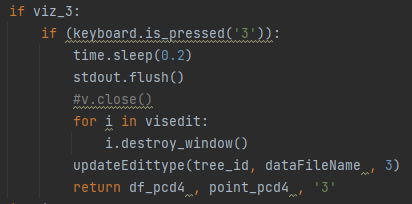


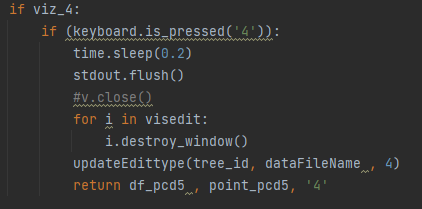
1. If 1 is pressed inside the editing mode, then the First window will be selected.



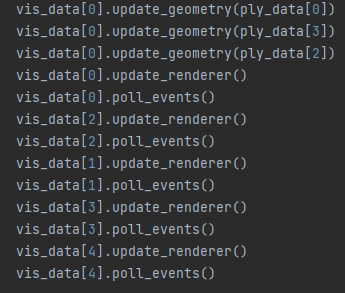
1. The same process for 1 i/p is carried out for other responses.







1. Now update all these windows as usual.



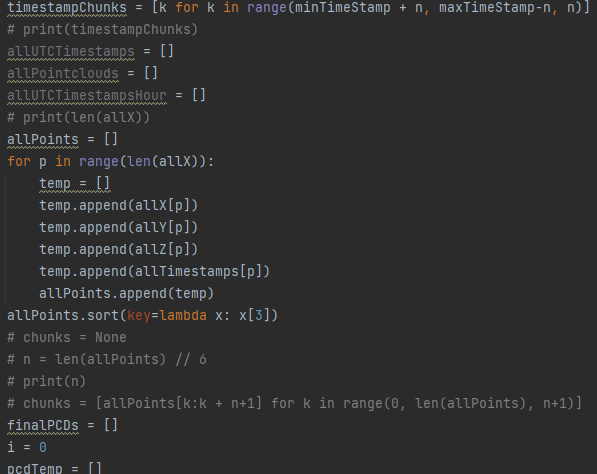
1. Pressing “z” unlocks the Multi-timestamp functionality of the program. Multi-timestamp occurs when we have LiDar data for a same tree, but taken on different time of the day. This data is color coded differently and we get 6 files for the timestamp.

For this functionality, we first have to set the “Multitimestamp” variable in the config.py file to be True. Then the files for the trees would be created for each timestamp.



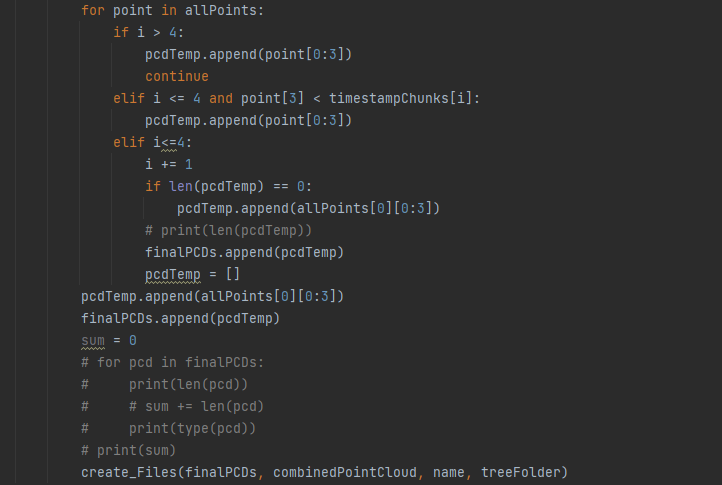
We first take all the co-ordinates for the center, including the Timestamp and then append them in an empty list called “allTimestamps”. Then we define a range inside which we have to take our Timestamps, and divide the range by 6.

This is done because we get 6 timestamp files, so to get those we divide by 6.



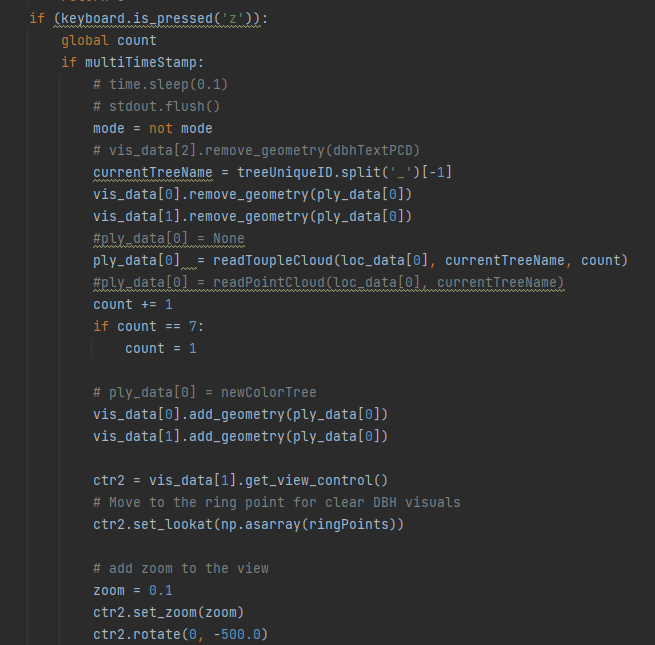
Then we initialize a list and populate the values of the range which we derived above using list comprehension of Python.

Then an empty list is again defined which is populated with the x,y,z and timestamp values, and the list on basis of timestamps.



Then we again initialize an empty list which we populate with the x,y,z and timestamp values for each division (6) which we did earlier, and create .ply files of them using “create\_files” function.

And them visualize by pressing “z” on the keyboard during visualization.



This is the code for visualizing the timestamp files.

In earlier builds, we had a functionality for “Tab” button which allowed us to select the points for the tree if it is visible to us. This allowed the user to go and manually select the points of the trunk which can be seen as a tree trunk. This was done through another library in Python known as PPTK.

PPTK is a library which is used to visualize 3D data, and has different functions made in itself, but is comparatively slow as compared to Open3d so it was discarded