Code tutorial

# General

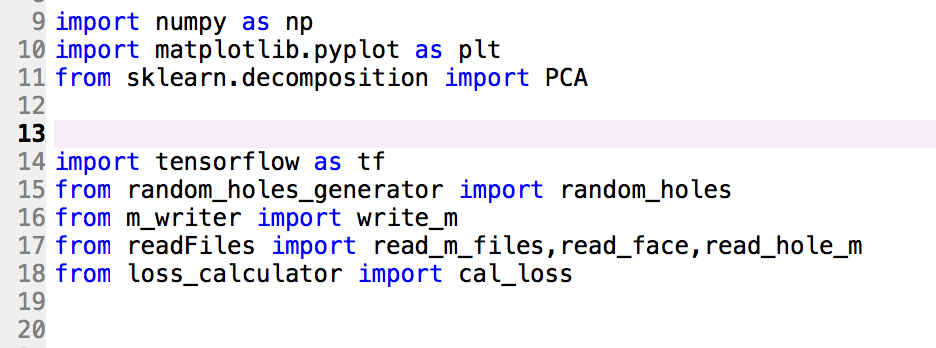
The code is designed to fix 3D hands model based on optimization strategy. The whole project includes 7 files :

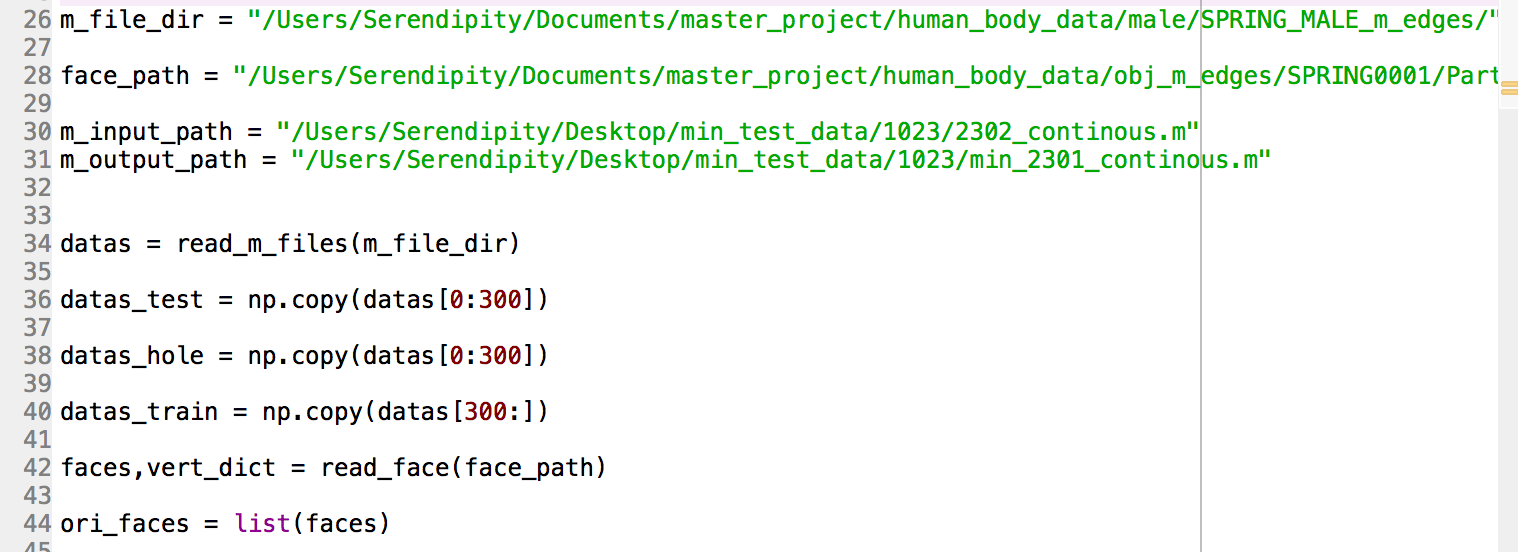
1. main.py
2. random\_holes\_generator.py
3. readFiles.py
4. m\_writer.py
5. loss\_calculator.py
6. hausdorff.py
7. AutoEncoder.py

# File explanation

2.1 main.py

This file is the entrance of the whole program, it calls functions from other file or library to finish the whole process of our method.

  
 we used PCA from sklearn library( Line 11) and used gradient descent from tensorflow( Line 14)  
 Line from 15 to 18 are the files I defined myself.



The code in this figure shows how to read data and preprocess data from our local directory.

m\_file\_dir describes the absolute address of your directory which 3d model data located.

face\_path is a file that contains “Face” information of our 3d models. Since the files in “m\_file\_dir” directory just include Vertex, we need to read face from other file.

m\_input\_path describes a 3d model file with holes that need to be repaired ( the input of the optimization part)

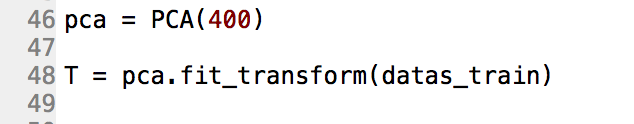
m\_output\_path describes the address that you want your result to store.

These variables depend on your actual running environment

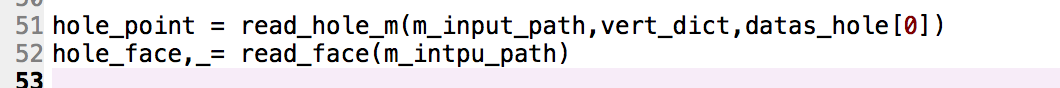
Calling “read\_m\_files” function could give you a 2d array that contain every data from “m\_file\_dir”. In this 2d array, each row represents a 3d hands model, for each row, every continuous 3 data represents a vertex.

Then slice data into training part and testing part, we keep another copy here as “data\_hole” for randomly generating holes.

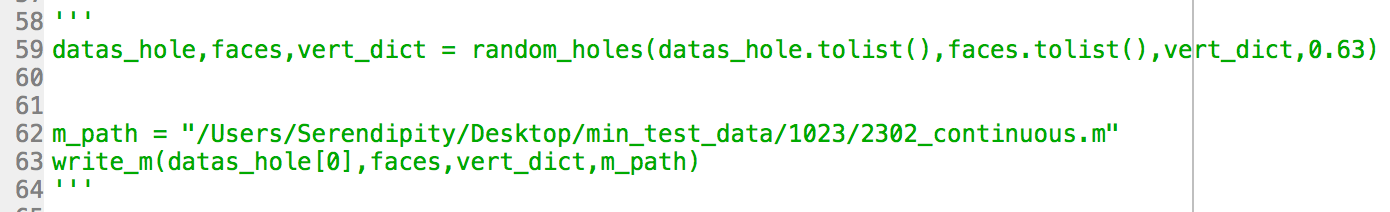
Calling “read\_face” function could give you two results, “faces” is a 2d array with size N by 3, every row in “faces” represent a face consist with 3 vertexes. “vert\_dict” is a dictionary that maps real index ( not start from 0 ) of Vertexes to fake index ( start from 0 )



These two lines code show how to apply PCA. Initialize PCA with an integer parameter which describes how many dimensions you want to keep after applying pca. Then call “fit\_transform” function to apply PCA.

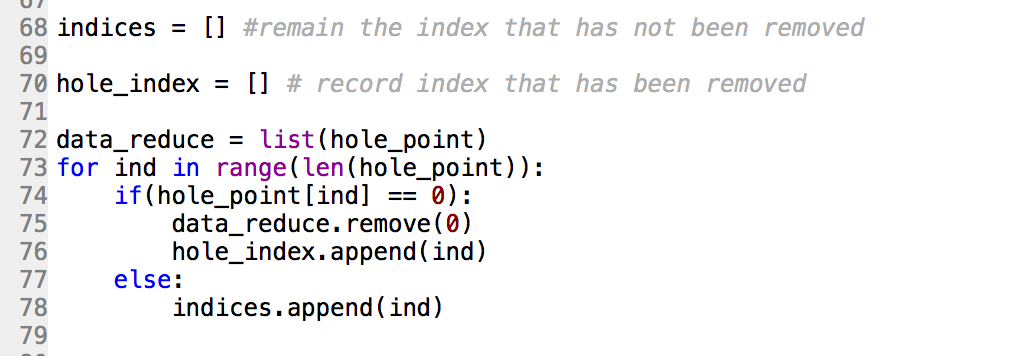


These 2 lines codes show how to read a file that has holes inside.



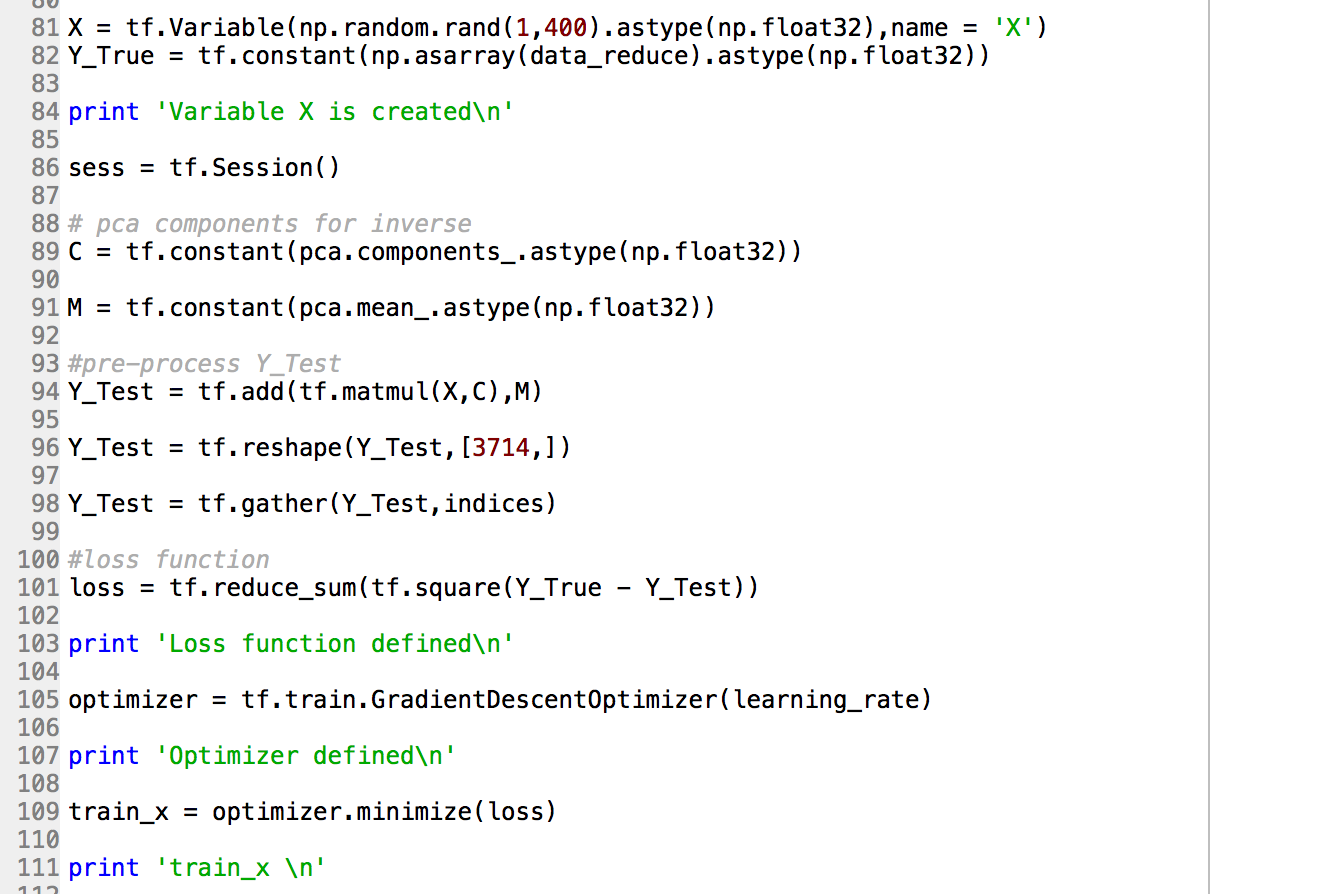
These 3 lines codes show how to randomly generate holes, you might also want to output the results.

Therefore, the above two are different input data. If you want your input from your local file, you just keep line 51 – 52. But if you want to randomly generate holes yourself, you need to keep the green part codes and uncomment it.



Previously, we have a input model which contains holes. To mark holes, I replaced the old value with zero. So, every zero value in our input array means a hole.

In order to do optimization later, I removed all 0 values in our input.



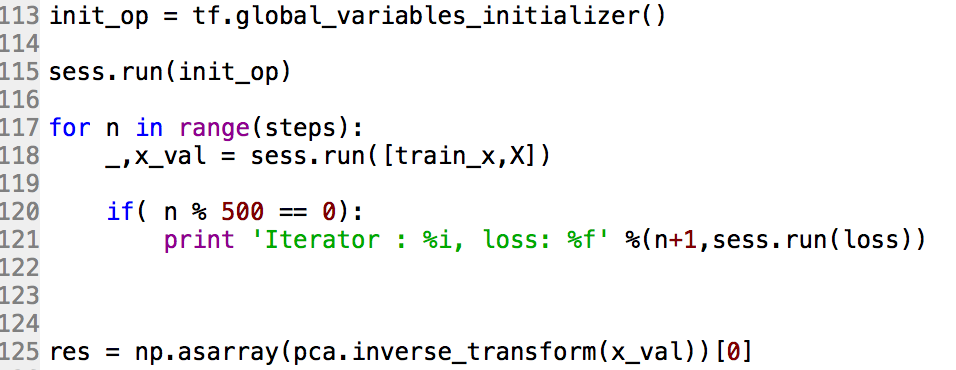
These codes are the optimization part which is implemented in tensorflow.

X represents the variable we want to train.

Y\_True is our input which is a hand model with holes

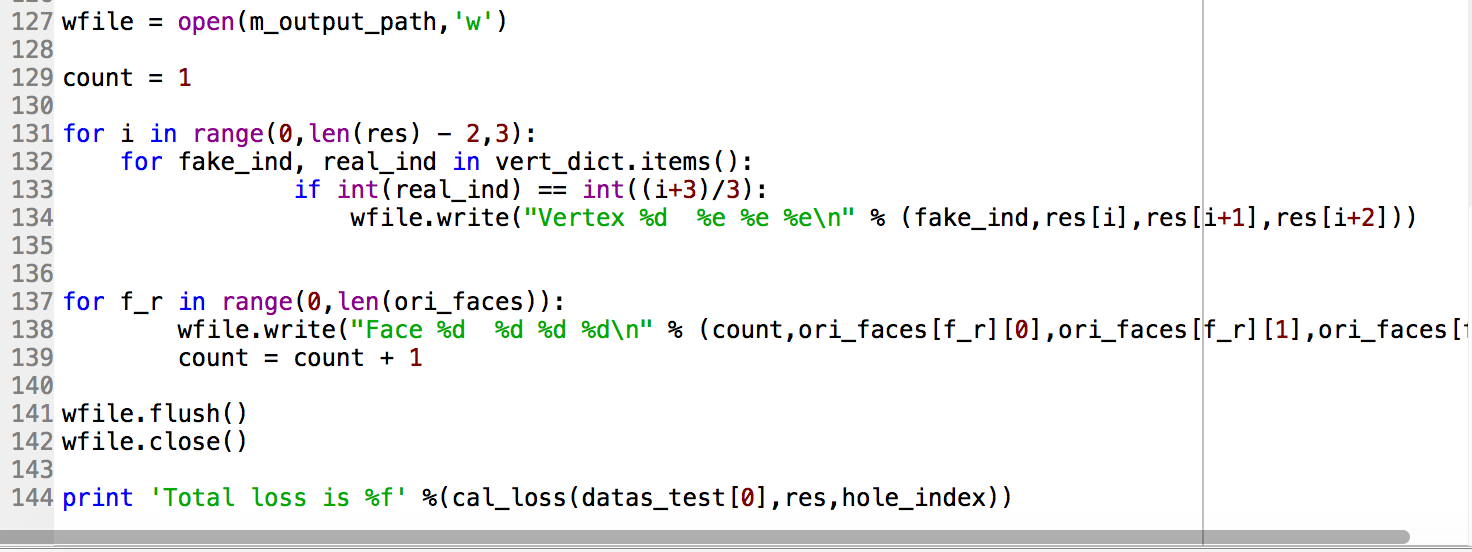
C and M are the data from PCA

After inverse X from low dimension to high dimention Y\_Test ( up to line 96), we also need to remove the corresponding holes index so that the dimention of Y\_Test is equal to Y\_True. This operation is done by calling tf.gather function.



Then is to run gradient descent algorithm.

After that, we inverse our result back to high dimension. The final result of our method is “res”.



Output the result to your local environment.

2.2 random\_holes\_generator.py

This file includes only one function called random\_holes. This function takes 4 parameters which are points(2d array, stores multiple models and every row represents a model), faces(2d array, read in main.py), vert\_dict( a dictionary records a index mapping, defined in main.py), rate( a value that tells what percentage of faces you want to remove). It returns 2 values, the points which already have holes(size same as input parameter, but holes are marked as 0), faces(new face after remove)



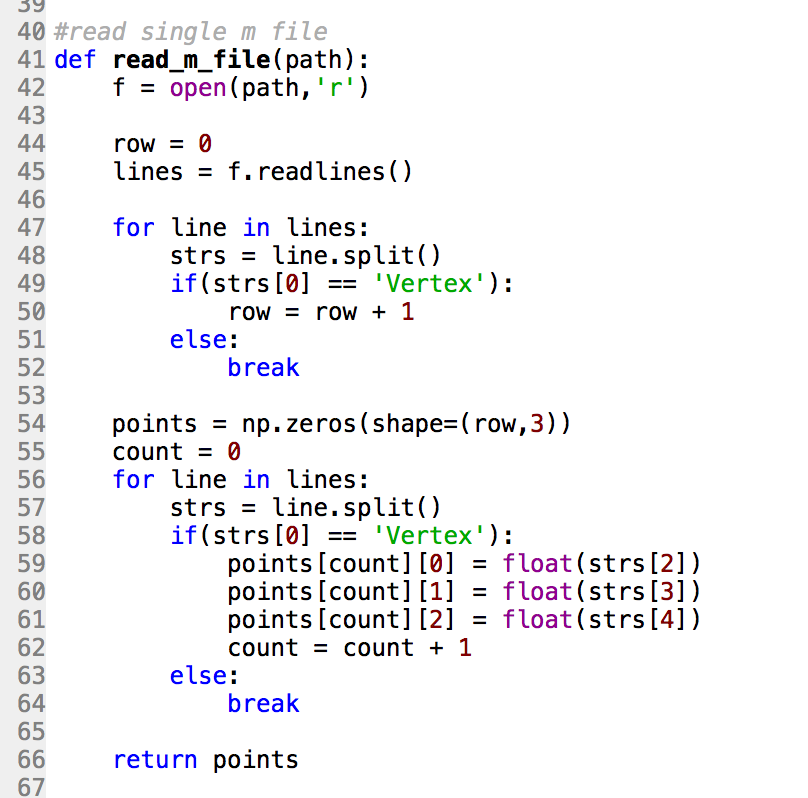
The idea here is to randomly choose a vertex, then remove it. Every faces related to this vertex are also removed.

2.3 random\_holes\_generator.py

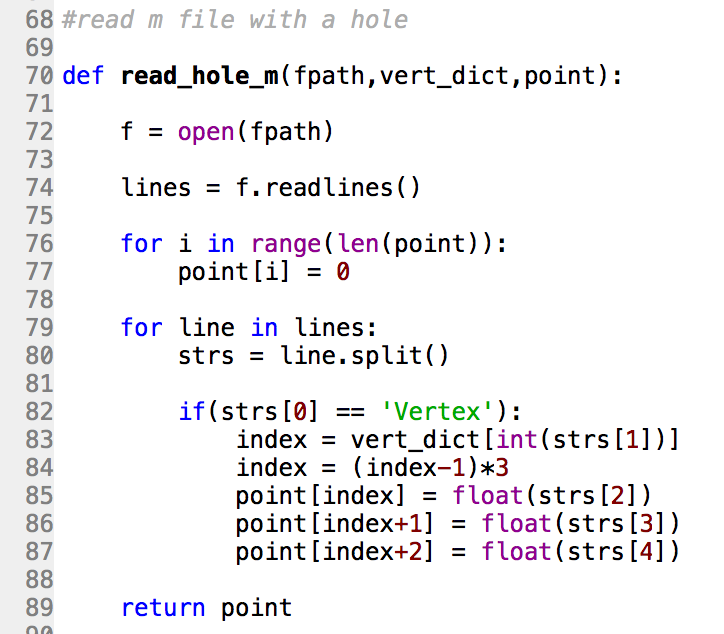
This file includes 4 functions:



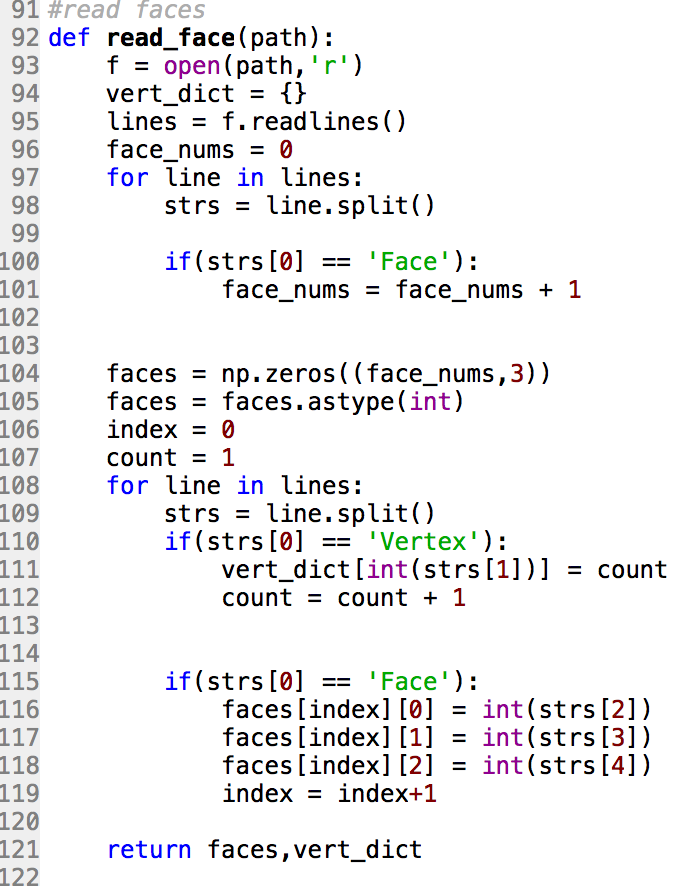
This function reads multiple files and return a 2d array, each row corresponds to a file



This function reads a single file and return a 2d array with size N by 3. This function is designed to calculate Hausdorff distance.

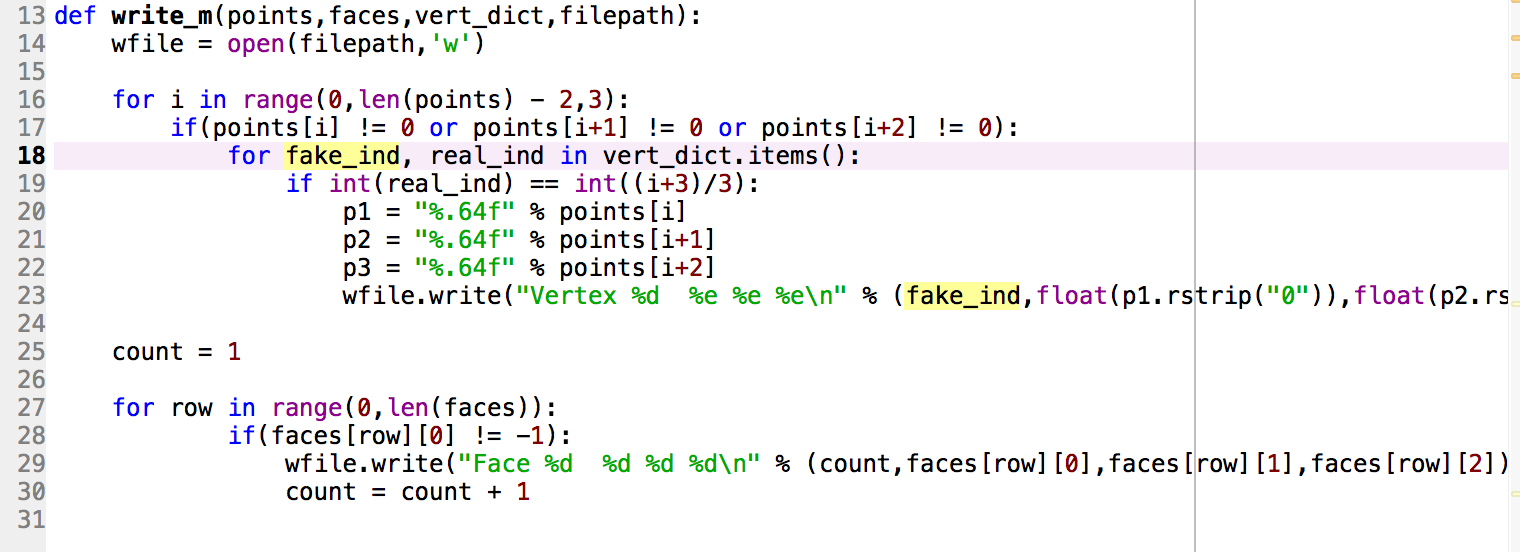


This function could read a file with holes, it accepts 3 parameters. First is your file address. Second is the dictionary we defined in main.py. Third is a single 3d model data from your datasets. Any copy of single row in “datas” from main.py is okay, just to used its length.



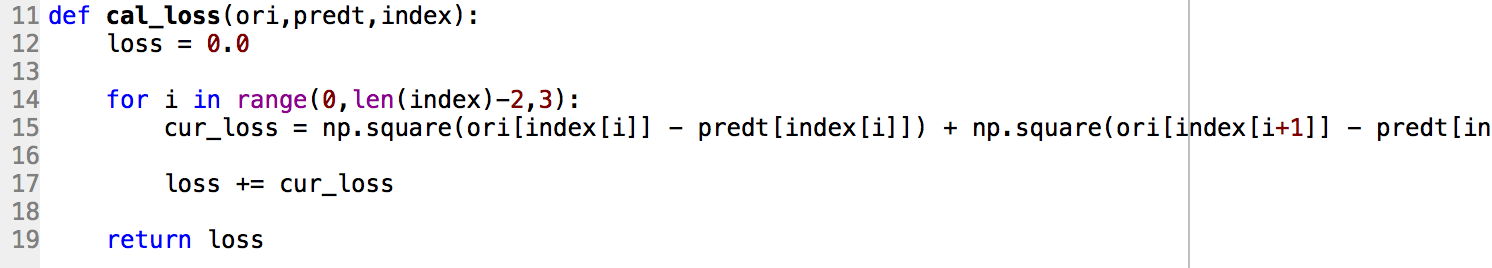
This function helps us to read faces.

2.4 m\_writer.py



This function takes 4 parameters, the first one is the single 3d model data that you want to output, the second is the corresponding faces, and the third is the dictionary, the forth is the address you want to output.

2.5 loss\_calculator.py

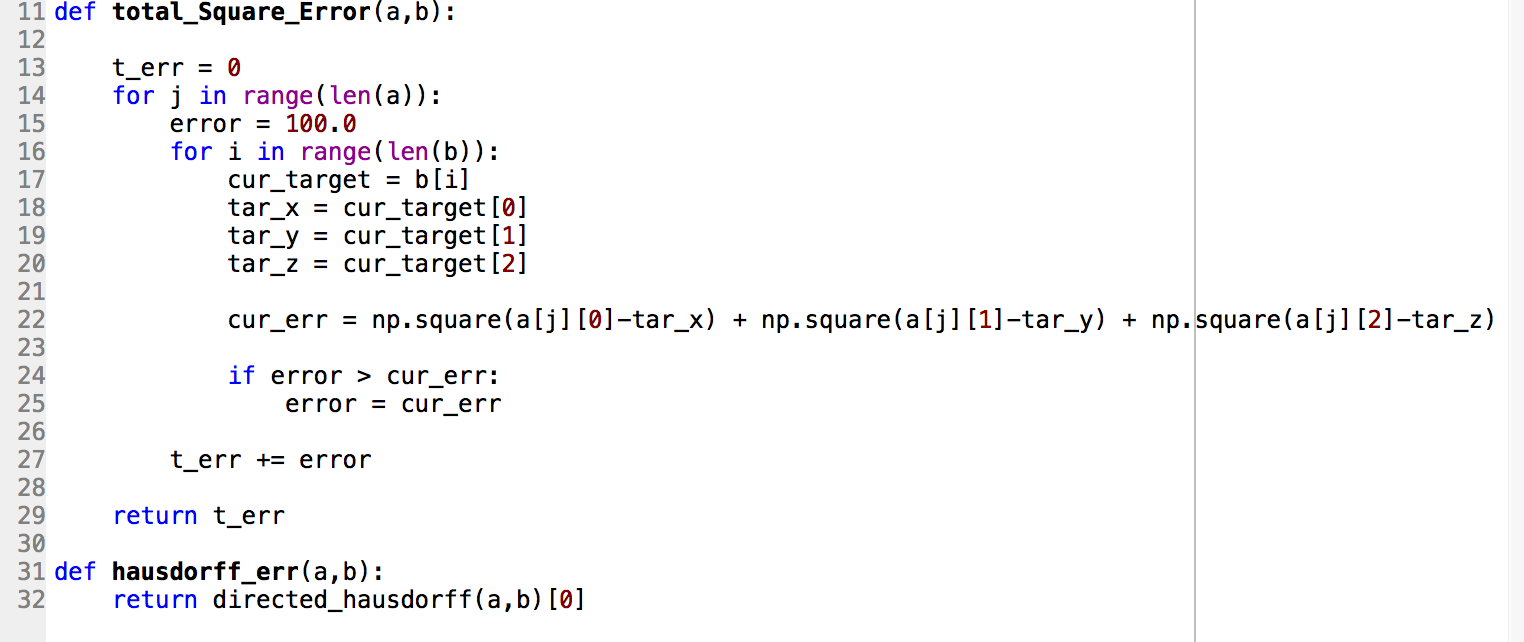


This function calculates the sum of square difference error between two hands, this is only for those have the same vertexs.

2.6 hausdorff.py

This file contains two functions: total\_Square\_Error, hausdorff\_err

These two functions are only used for “continuous hole” experiment, the reason to define these two new measure functions here is because after “polymender” and “meshlab” repair, the number of vertexes and faces are changed. Therefore, we cannot use the “loss\_calculator.py” in this case anymore.



total\_Square\_Error: It takes 2 parameters, a is the original real holes vertex from our testing dataset . b is the result from “polymender” or “meshlab” repaired.

hausdorff\_err: a is the original hole surface. b is the result from “polymender” or “meshlab” repaired.

2.7 Autoencoder.py

This file implements an Autoencoder network. To apply this network in our algorithm, just simply modify some codes in main.py.

Combined with auto-encoder:

X in main.py is no longer initialized as 400 size, now is 100.

Line 94



will be modified as :

Y\_Test = tf.add(tf.matmul(decoder(X),C),M)