Documentation for Silent border code

In this documentation we will show what does our code provides and explain the code with details.

A diagram of a path

Description automatically generated

First, I will explain how the data is gathered and what we want briefly. This picture is an example of how the muon particles hit and go through these sensors. We called The hits at the top as incoming hits or incoming points and the hits at the bottom as outgoing hits as outgoing points. We should have at least 3 incoming and outgoing hits separately.

A table of numbers and letters

Description automatically generated



Before starting with code, we would like to show the data we are using. Each column has its name- "EventID", "X(cm)", "Y(cm)", "Z(cm)", "GlobalTime(ns)", "TrackID", "Particle(PDG)", "Volume"- but we are using the ones that we marked. The red column represents “EventID" , the green columns represent "X(cm)", "Y(cm)", "Z(cm)", the yellow column represents "TrackID", and the blue column represents "Volume".

Let’s start with the first step our code do:

A screenshot of a computer code

Description automatically generated

This part of the code retrieves the input file path from the parsed arguments. Then we read the data. The data is read from the CSV file using the read\_csv\_concurrent function. If there is a chunk\_size given as in our example it will use concurrent programming to use multiple CPU cores to read the data faster. Otherwise, it will read it without concurrent programming. The code will use either Dask.dataframe (dd) or Panda (pd) module for reading the data. The execution time is recorded by measuring the time before and after reading the CSV file. After finishing the reading data part the code is sorting the data using the sort\_data function which will sort the data according to the Event\_ID and Track\_ID. Then Unique event IDs are extracted from the sorted data for further processing or analysis.

A black and red text

Description automatically generated

This part of the code extract coordinates and the type of the volume. If the type of the volume is not we want, the hit\_types will be None.

After that in the main function we started to use extract coordinates to find the borderline with minimum and maximum x,y,z coordinates. Here is the part of the code that does it:

A screen shot of a computer code

Description automatically generated

we just take into account the coordinates with our desired hit\_types and the others is not considered. The main function also contains other parts and uses other functions in the code. I will show it and explain briefly:

A screen shot of a computer

Description automatically generated

The code starts by creating a three-dimensional grid of zeros using NumPy. This grid will be used to store voxel (3D pixel) information. Then again, we used concurrent programming to make our process faster. It divides the number of unique\_events according to chunk\_size. Thus, it enables parallel processing. After the dividing part it does another function process\_chunk which I will explain later. After finishing all data, we save this combined 3d grid in 2d slices (xy,yz,zx seperately).

A white background with black text

Description automatically generated

In process\_chunk function we simply divide the data according to each unique\_ID and use process\_event function in which we will filter our data and choose only the data to work with that meets the requirements.

Process\_event contains a lot of other functions therefore I will explain each part separately.

A screen shot of a computer code

Description automatically generated

In this part of the process we separated the data of unique\_Event\_ID according to unique track\_IDs.

After that we will again use extract\_coordinates function to get the data that has hit\_types we want.



In this part we first check if we have more than 5 different coordinates as we should have at least 6 points (three incoming and three outgoing). After that we create mask variable to contain the points that have required hit\_types.

A black and white image of a mathematical equation

Description automatically generated

After that we check if the points in the mask is more or equal to 6 as we should have 3 incoming and 3 outgoing lines. For determining incoming and outgoing hits we use select\_incoming\_outgoing function in which we calculate the maximum distances between 2 corresponding points and choose it as a change\_index. The points before change index are chosen as incoming points, and the points after change are chosen as outcoming points.

A white background with colorful text

Description automatically generated

In this part we check if the number of incoming and outgoing points are equal or more than 3. Also, we checked if the z axis of the last incoming point is greater than the first outgoing point as it should be. After that we check the distances between incoming points as we know the distances between sensors are 10 cm. If there is more than 1 distance that corresponds to that the function will return True. We do it also for outgoing points.



If the data passes the last part, then this means we have the number of incoming and outgoing points are equal or more than 3. But as we need only 3 points from each of them, we will choose the only 3 of them that gives as more straight lines among the possibilities. We are doing it with the function called select\_points\_for\_straight\_line in which we divide the points to 3 arrays by checking their differences between them (if the difference more than 10 the points after change is stored in the next array). After that we are finding all of the combinations that can happen from these 3 arrays and find the 3 points that create the straightest line among them by using several python modules. After we did the same for outgoing points, we use create\_correct\_path function which I will explain now.

A screen shot of a computer program

Description automatically generated

In create\_correct\_path function we are creating POCA(point of the closest approach) point using incoming and outgoing line and we create a list containing all of the 7 incoming, outgoing and POCA points together.

A screenshot of a computer program

Description automatically generated

In the last part of the code, we first calculate the angle between 2 vector lines (the line that connects second and third incoming point and the line that connects first outgoing and third incoming point). If the angle bigger than 1 degree, then we are breaking our function and go to try the next event\_ID. If data passes this test, then it will go to the next and last step where it makes a new grid and use simulate\_rays function we made to simulate rays on new grid, which will be added to the combined\_grid. After that process event will go to process the next unique event\_id data until the data is finished and all of the passed event\_id data will add new grids on combined\_grid.

A graph of a function

Description automatically generated with medium confidence

This is an example of a Combined grid picture we saved it will save them as slices (how many slices will be saved is dependent on min and max x,y,z coordinates).