# SEGMENTATION OF POINT CLOUD DATA

**BY**

**KATTA ROHITH**

**(Admission No. 19JE0428)**

**BY**

**ANAND DESHMUKH**

**(Admission No. 19JE0124)**

**THESIS SUBMITTED TO**

**DEPARTMENT OF MECHANICAL ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY (ISM), DHANBAD – 826004**

**MAY, 2023**

For the award of the degree of

**BACHELOR OF TECHNOLOGY**

**Supervisor: Prof.Kailash Jha**

# CERTIFICATE

This is to certify that the thesis titled “Segmentation of point cloud data” being submitted by Mr. Katta Rohith (Admission no: 19JE0428) and Anand Deshmukh (Admission no: 19JE0124) for the award of the degree of Bachelor of Technology in the Department of Mechanical Engineering of Indian Institute of Technology (Indian School of Mines), Dhanbad is a record of bonafide research work carried out by him/her under my supervision. In my opinion, the thesis is worthy of consideration for the award of the degree of Bachelor of Technology in accordance with the regulations of the institute. The results presented in the thesis have not been submitted to any other university or institute for the award of any degree.

[Signature of Supervisor]
Prof. Kailash jha
Associate professor
Department of Mechanical Engineering,
Indian Institute of Technology (ISM) Dhanbad-826004, Jharkhand, INDIA

**Date: 16-05-2023**

## DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled "Segmentation of point cloud data" in partial fulfillment of the requirements for the award of the degree of B.Tech in Mechanical Engineering is an authentic record of my own work carried out during the period from 2019 to 2023 under the supervision of Prof. Kailash Jha,Department of Mechanical Engineering, Indian Institute of Technology (ISM) Dhanbad, Jharkhand, India.I acknowledge that I have read and understood the UGC (Promotion of Academic Integrity and Prevention of Plagiarism in Higher Educational Institutions) Regulations, 2018. These Regulations were published in the Indian Official Gazette on 31st July, 2018.I confirm that this Dissertation has been checked for plagiarism using the online plagiarism checking software provided by the Institute. I herewith confirm that the Dissertation has less than 10% similarity according to the plagiarism checking software's report and meets the MoE/UGC Regulations as well as the Institute's rules for plagiarism.I further declare that no portion of the dissertation or its data will be published without the Institute's or Guide's permission. I have not previously applied for any other degree or award using the topics and findings described in my dissertation.

(Signature of the Student)
Name of the Student: Katta Rohith
Admission No.: 19JE0428
Department : Mechanical Engineering

# ACKNOWLEDGEMENTS

I am grateful to Prof. Kailash jha for guiding us in completing this project and providing required resources and acquainting us with different professionals and technicians for completing this project.

I am grateful to Dr. Amit Rai dixit HOD of mechanical engineering for giving us access to the 3D Scanner which we used to scan our staircase model.

I am grateful to Sushant Gautam who is pursuing PhD in mechanical engineering for helping us in the nuances of the project and helping us in providing a sample dataset for testing with our code.I am very thankful to him in helping us filtering the point cloud that we have generated using 3D scanner which has many outlier points.

I am grateful to Gopal ji who helped us in working with a 3D scanner for scanning our staircase model and providing us with an STL file.

KATTA ROHITH

**Date:16-05-2023**

# ABSTRACT

The proposed method uses the normal vector computed for each point in the point cloud for segmentation purposes. By analyzing the local surface characteristics, the algorithm determines the boundaries between different surfaces, enabling accurate and robust segmentation. First, the point cloud data is pre-processed to estimate the surface normal at each point. This step is important for getting the geometric information of the scene and differentiating between different surface orientations. Next, a region-growing algorithm is employed, which iteratively expands seed regions based on the similarity of their surface normal. The growing process is guided by a similarity criterion, allowing the algorithm to merge points with similar normal while preserving the boundaries between distinct surfaces. The python code takes data points and its normal and generates a series of surfaces in PyCharm showing the segmented surfaces. To enhance the accuracy and reliability of the segmentation pre-processing of point cloud has been done. These include outlier removal to eliminate noisy data points, filtering to get uniform point density and region refinement to address potential over-segmentation or under-segmentation.

KEY WORDS: Point clouds, Segmentation, Normal vector.

# LIST OF FIGURES

# CONTENTS

# 1 INTRODUCTION AND OBJECTIVE

## 1.1 Introduction

## Point cloud:

A point cloud is a set of data points in space representing a 3D object having its own cartesian coordinates (X , Y , Z). Point cloud is the group of key points of any object which together can define its shape and size. It's made up of a multitude of points captured using a 3D laser scanner. If you're scanning a building, for example, each virtual point would represent a real point on the wall, window, stairway, metalwork or any surface the laser beam comes into contact with.The denser the points, the more detailed the representation, which allows smaller features and texture details to be more clearly and precisely defined. So, if you were to zoom in on a point cloud of The Tower Bridge in London, you'd see tiny points creating the whole point cloud.

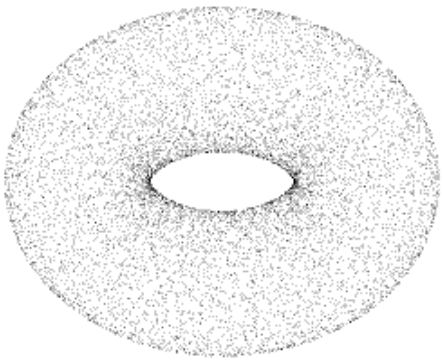


**Fig 1.1** Point cloud of a Torus.

## Segmentation:

3D point cloud segmentation is the process of classifying point clouds into multiple homogeneous regions ,the points in the same region will have the same points.One can precisely determine the shape, size, and other properties of the objects in 3D data.The segmentation process is helpful for analyzing the scene in various applications like locating and recognizing objects,classification, and

feature extraction.

**Fig 1.2** Segmentation of a chair.

## Applications:

1.In reverse engineering-based manufacturing.The purpose of reverse engineering is to find out how an object or system works.Reverse engineering can be used to know how something works to recreate the object or to create a similar object with added enhancements.

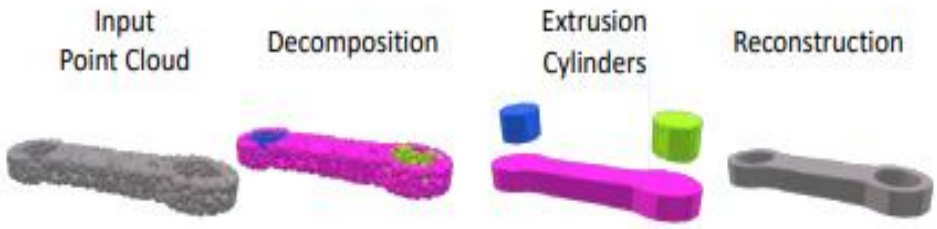



**Fig 1.3** A spanner is being converted to point cloud where segmentation is done for added enhancements.

2.For surveying of large area like any construction site, buildings,farm areas,natural places etc.

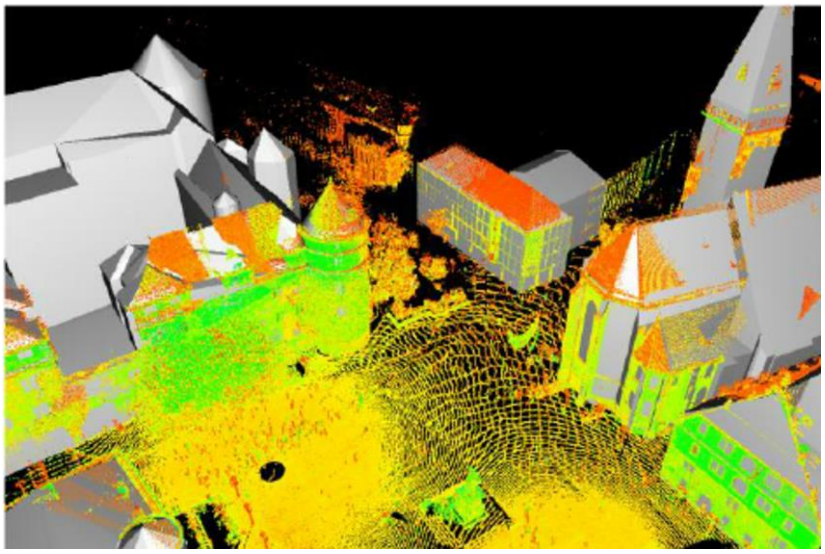


**Fig 1.4** 3D point cloud from laser scanner aligned with virtual 3D city model

3.This method can be utilized as a helping tool in solving industrial problems like to detect any defect in any machine part,sorting the geometrical data of machine parts in the form of segmented point cloud.

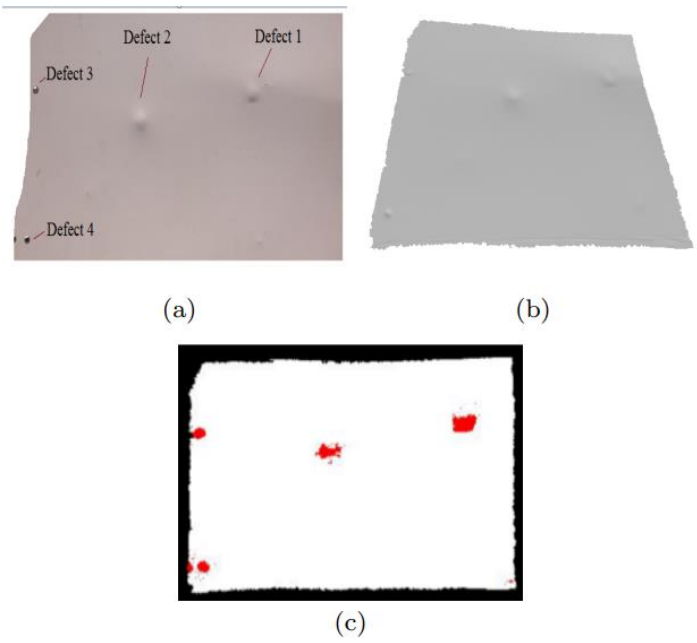



**Fig 1.5 a)** defected main body part of an aircraft
**1.5 b)** acquired point cloud
**1.5 c)** The detected defects on the mesh are shown in red color.

4.Reconstruction of any surface and 3D model of machine parts and tools.

5.Feature curve extraction.

## 1.2 Objectives :

To Segment the 3D point cloud of a cubical modeland staircase model into multiple regions based on similar attributes.we use normal vectors as attributes.below shows the picture of staircase model that we have used to generate the point cloud and for further segmentation.

Below staircase object is scanned using 3d scanner and an STL file is extracted ,this file is further processed using 3dzefyr software and a point cloud is obtained.Normals are generated for each point using c++ code . segmentation and display of segmented surfaces is done using python language.

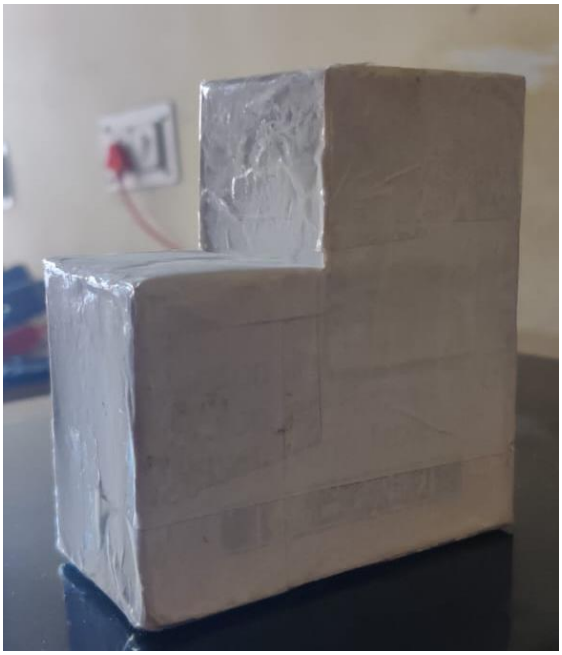


**Fig 1.7** staircase model used for segmentation

# 2 LITERATURE REVIEW

Today 3D model and point clouds are very popular being currently used in several fields, shared through the internet and even accessed on mobile phones.Segmentation is the process of grouping point clouds into multiple homogeneous regions with similar properties whereas classification is the step that labeLs these regions. The main goal of the paper is to analyze the most popular methodologies and algorithms to segment and classify 3D point clouds. 3D point clouds are the simplest but at the same time powerful collection of elementary geometrical primitives able to represent shape, size, position and orientation of objects in space. This information may be augmented with additional contents obtained from other sensors or sources, such as colors, multispectral or thermal information, etc. For a successful exploitation of point clouds and to better understand them, we must first proceed with segmentation and classification procedures.

Due to the complexity and variety of point clouds caused by irregular sampling, varying density, different types of objects, etc., point cloud classification and segmentation are very active research topics.

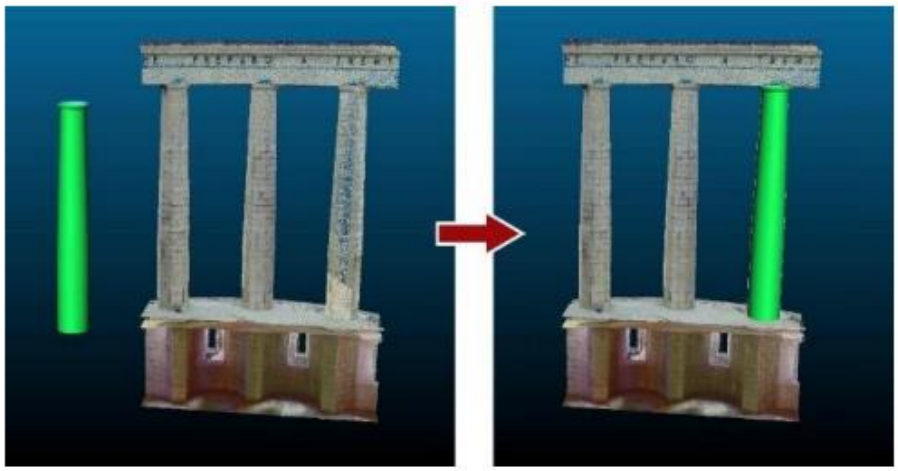
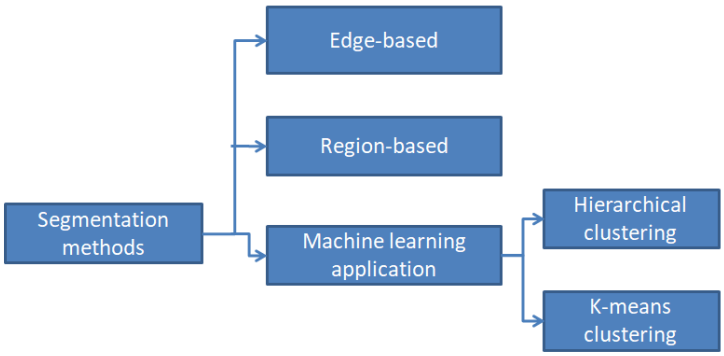


**Fig 2.1** segmentation of 3d point cloud by geometric primitive fitting

There are various kinds of segmentation methods :




The segmentation is done using k-nearest neighbor algorithm K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.K-NN algorithm can be

used for Regression as well as for Classification but mostly it is used for the Classification problems.KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Visualization is done through matplotlib which is a python library that provides a wide range of tools for creating visualizations such as line plots, scatter plots, bar plots, histograms, and more. It is built on top of the matplotlib library and provides a high-level interface for creating visualizations with a few lines of code.

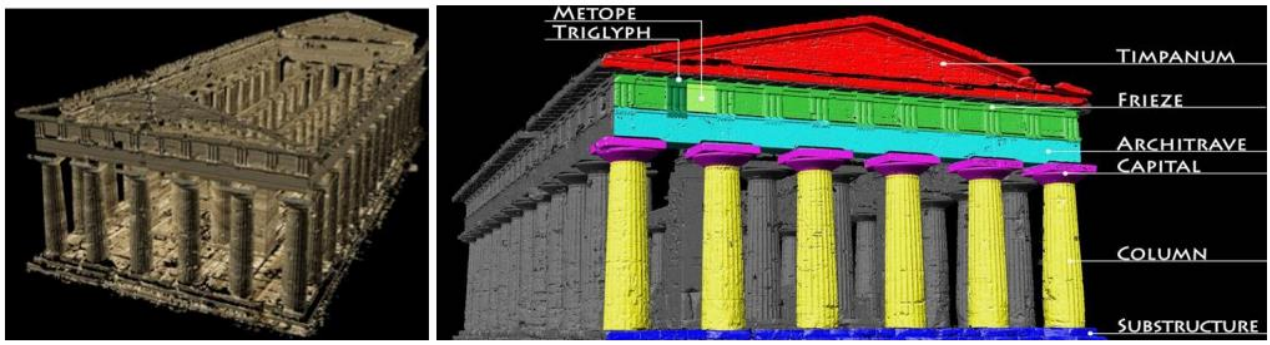



fig 2.2 different parts of the temple are classified and grouped together based on similar attributes like size and shape etc..

# 3.METHODOLOGY

The staircase model is scanned using 3D scanner and an STL file is generated where the cubical model is scanned in different orientations by the scanner.This STL file is then converted into point cloud which is a excel sheet containing 3D coordinate points.

These points obtained were used to generate the normals which is used as attribute for segmentation. this segmentation is a normal based segmentation.

The obtained normals and points were used to segment using k-nearest neighbor algorithm which is written using python language ,python uses NearestNeighbors library to perform k-nn algorithm on the dataset.

The normals generated were stored in the same excel sheet along with the the 3D points

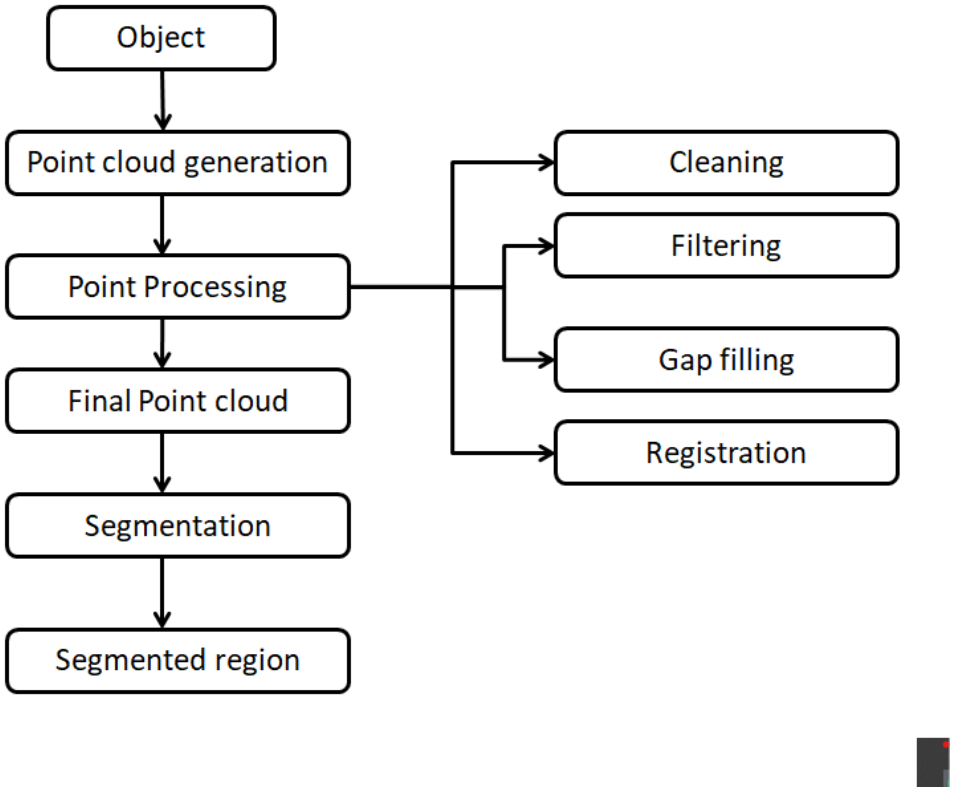



**Fig 3.1** flow chart of the process

## 3.1 FINAL POINT CLOUD

The cubical box is scanned using 3D scanner and an STL file is generated.Using 3D zephyr software the STL file is converted into 3D co-ordinates of points .the obtained points were sent under for further processing .

**3.1.1 Cleaning**: Point cloud data cleaning involves removing any noise, outliers, or unwanted points from the data. This process ensures that the point cloud data is accurate and reliable, which is important for many applications. Cleaning techniques can include removing points that are too far away from the mean, removing points that do not meet certain criteria (e.g., based on color or reflectance values), and removing points that are located in areas that are not of interest.

**3.1.2 Filtering**: Point cloud data filtering involves extracting certain parts of the point cloud that meet certain criteria. This process is used to reduce the size of the point cloud and to remove any unwanted parts. Filtering techniques can include selecting points based on their location, color, reflectance, or other characteristics.

**3.1.3 Gap filling**: Point cloud data gap filling involves filling in areas where there are missing points. This process is important for many applications, as missing data can result in inaccurate or incomplete

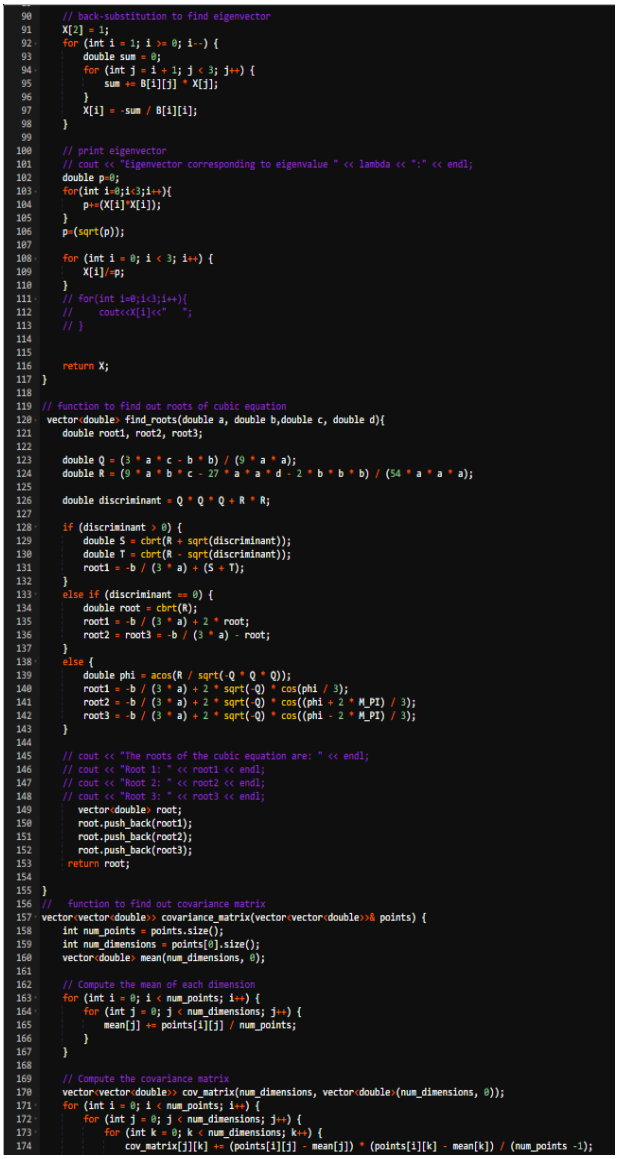
results. Gap filling techniques can include interpolating missing data using neighboring points or using other algorithms to estimate missing data.

**3.1.4 Registration** : Point cloud data registration involves aligning multiple point clouds together to create a unified point cloud. This process is important for applications such as 3D modeling, where multiple scans are combined to create a single 3D model. Registration techniques can include using feature-based methods, iterative closest point (ICP) algorithms, or other techniques to align the point clouds.

**3.1.5 Normal estimation :** The surfaces are segmented using normal based segmentation ,The algorithm for normal calculation is written in C++ language where it reads point cloud data from a CSV file calculates the normal vectors for each point by finding the eigenvectors of the covariance matrix of the neighboring points, and writes the point coordinates and corresponding normal vectors to another CSV file.

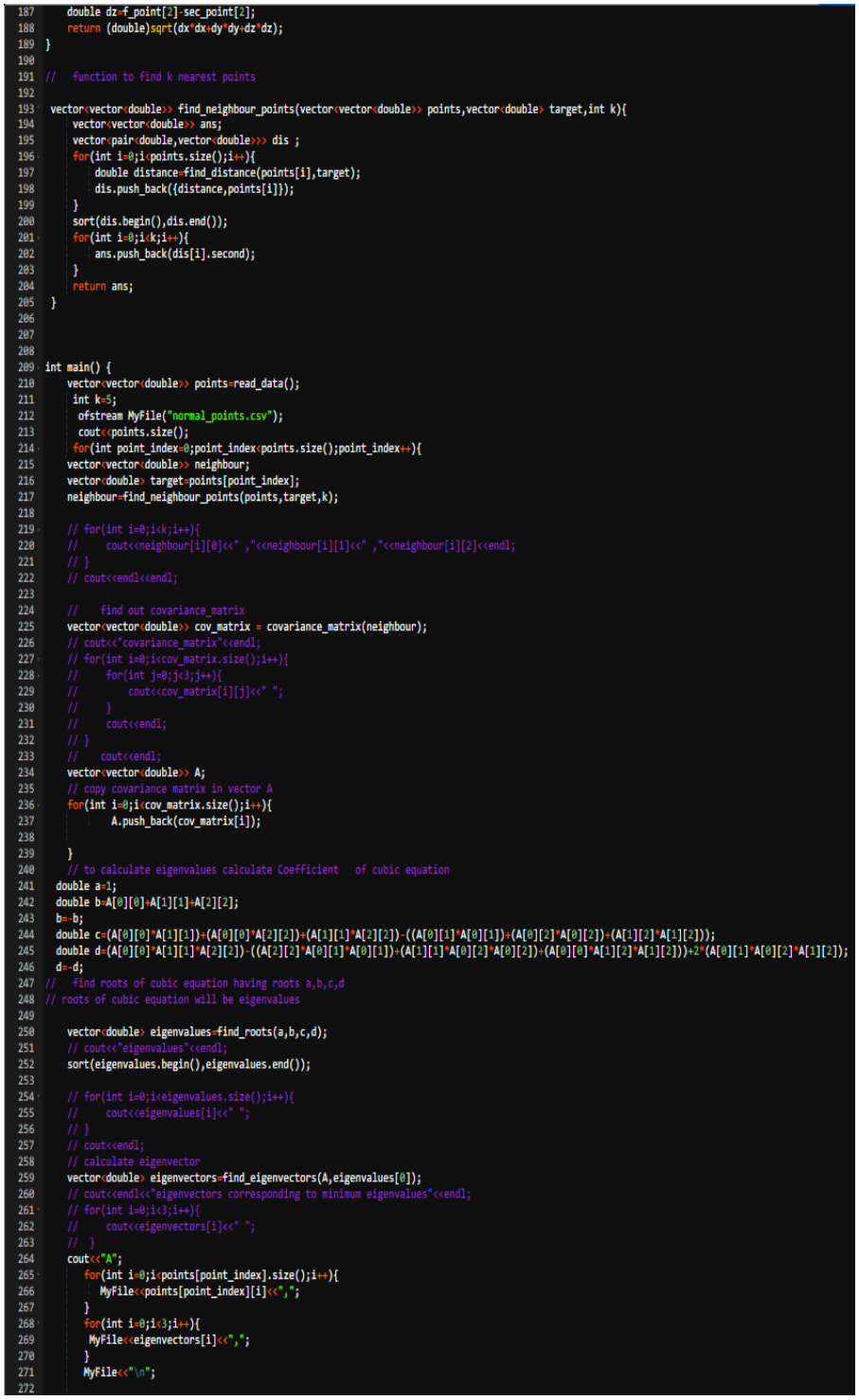
```cpp
#include <iostream>
using namespace std;
#include<bits/stdc++.h>
#include <iostream>

using namespace std;
// function to read file data
vector<vector<double>> read_data(){
    vector<vector<double>> point_data;

// Create a text string, which is used to output the text file
string myText;

// Read from the text file
ifstream MyReadFile("point_cloud_data.csv");

// Use a while loop together with the getline() function to read the file line by line
  int count=0;
while (getline (MyReadFile, myText)) {
//  Output the text from the file
  stringstream ss(myText);
  // cout<<myText<<endl;
  vector<double> temp;
  while(ss.good()){
    string substr;
    getline(ss,substr,',');
    int len=substr.size();
     double n=stod(substr.substr(1,len-2));
    //  cout<<substr<<" "<<n<<endl;
    temp.push_back(n);
  }
  point_data.push_back(temp);
}

// Close the file
MyReadFile.close();
return point_data;

}
//   function to calculate eigenvectors
vector<double> find_eigenvectors(vector<vector<double>> A,double lambda){

    vector<double> X(3);
    int index=0;
    for(int i=0;i<3;i++){
        int count=0;
        for(int j=0;j<3;j++){
           if(A[j][i]==0)
           count++;
        }
        if(count==3){
            // cout<<"index"<<i<<"    "<<endl;
            for(int p=0;p<3;p++){
                // cout<<p<<" "<<i<<"a";
                if(p==i){
                    // cout<<"C";
                    X[p]=1;
                }
                else
                X[p]=0;
            }
            return X;
        }
    }

    // subtract λI from A
    // double B[3][3];


    vector<vector<double>> B( 3 , vector<double> (3));
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            B[i][j] = A[i][j];
            if (i == j) {
                B[i][j] -= lambda;
            }
        }
    }

    // perform Gaussian elimination
    for (int i = 0; i < 3; i++) {
        for (int j = i + 1; j < 3; j++) {
            double factor = B[j][i] / B[i][i];
            for (int k = i; k < 3; k++) {
```

**Fig 3.2** part-1 of the normal code



```cpp
    // back-substitution to find eigenvector
    X[2] = 1;
    for (int i = 1; i >= 0; i--) {
        double sum = 0;
        for (int j = i + 1; j < 3; j++) {
            sum += B[i][j] * X[j];
        }
        X[i] = -sum / B[i][i];
    }

    // print eigenvector
    // cout << "Eigenvector corresponding to eigenvalue " << lambda << ":" << endl;
    double p=0;
    for(int i=0;i<3;i++){
        p+=(X[i]*X[i]);
    }
    p=(sqrt(p));

    for (int i = 0; i < 3; i++) {
        X[i]/=p;
    }
    // for(int i=0;i<3;i++){
    //     cout<<X[i]<<"   ";
    // }


    return X;
}

// function to find out roots of cubic equation
vector<double> find_roots(double a, double b,double c, double d){
    double root1, root2, root3;

    double Q = (3 * a * c - b * b) / (9 * a * a);
    double R = (9 * a * b * c - 27 * a * a * d - 2 * b * b * b) / (54 * a * a * a);

    double discriminant = Q * Q * Q + R * R;

    if (discriminant > 0) {
        double S = cbrt(R + sqrt(discriminant));
        double T = cbrt(R - sqrt(discriminant));
        root1 = -b / (3 * a) + (S + T);
    }
    else if (discriminant == 0) {
        double root = cbrt(R);
        root1 = -b / (3 * a) + 2 * root;
        root2 = root3 = -b / (3 * a) - root;
    }
    else {
        double phi = acos(R / sqrt(-Q * Q * Q));
        root1 = -b / (3 * a) + 2 * sqrt(-Q) * cos(phi / 3);
        root2 = -b / (3 * a) + 2 * sqrt(-Q) * cos((phi + 2 * M_PI) / 3);
        root3 = -b / (3 * a) + 2 * sqrt(-Q) * cos((phi - 2 * M_PI) / 3);
    }

    // cout << "The roots of the cubic equation are: " << endl;
    // cout << "Root 1: " << root1 << endl;
    // cout << "Root 2: " << root2 << endl;
    // cout << "Root 3: " << root3 << endl;
      vector<double> root;
      root.push_back(root1);
      root.push_back(root2);
      root.push_back(root3);
     return root;

}
//   function to find out covariance matrix
vector<vector<double>> covariance_matrix(vector<vector<double>>& points) {
    int num_points = points.size();
    int num_dimensions = points[0].size();
    vector<double> mean(num_dimensions, 0);

    // Compute the mean of each dimension
    for (int i = 0; i < num_points; i++) {
        for (int j = 0; j < num_dimensions; j++) {
            mean[j] += points[i][j] / num_points;
        }
    }

    // Compute the covariance matrix
    vector<vector<double>> cov_matrix(num_dimensions, vector<double>(num_dimensions, 0));
    for (int i = 0; i < num_points; i++) {
        for (int j = 0; j < num_dimensions; j++) {
            for (int k = 0; k < num_dimensions; k++) {
                cov_matrix[j][k] += (points[i][j] - mean[j]) * (points[i][k] - mean[k]) / (num_points -1);
```

**Fig 3.3** part-2 of the normal code



```
    double dz=f_point[2]-sec_point[2];
    return (double)sqrt(dx*dx+dy*dy+dz*dz);
}

//   function to find k nearest points

vector<vector<double>> find_neighbour_points(vector<vector<double>> points,vector<double> target,int k){
    vector<vector<double>> ans;
    vector<pair<double,vector<double>>> dis ;
    for(int i=0;i<points.size();i++){
        double distance=find_distance(points[i],target);
        dis.push_back({distance,points[i]});
    }
    sort(dis.begin(),dis.end());
    for(int i=0;i<k;i++){
        ans.push_back(dis[i].second);
    }
    return ans;
}



int main() {
    vector<vector<double>> points=read_data();
    int k=5;
     ofstream MyFile("normal_points.csv");
     cout<<points.size();
    for(int point_index=0;point_index<points.size();point_index++){
    vector<vector<double>> neighbour;
    vector<double> target=points[point_index];
    neighbour=find_neighbour_points(points,target,k);

    // for(int i=0;i<k;i++){
    //     cout<<neighbour[i][0]<<" ,"<<neighbour[i][1]<<" ,"<<neighbour[i][2]<<endl;
    // }
    // cout<<endl<<endl;

    //    find out covariance_matrix
    vector<vector<double>> cov_matrix = covariance_matrix(neighbour);
    // cout<<"covariance_matrix"<<endl;
    // for(int i=0;i<cov_matrix.size();i++){
    //     for(int j=0;j<3;j++){
    //         cout<<cov_matrix[i][j]<<" ";
    //     }
    //     cout<<endl;
    // }
    //    cout<<endl;
    vector<vector<double>> A;
    // copy covariance matrix in vector A
    for(int i=0;i<cov_matrix.size();i++){
            A.push_back(cov_matrix[i]);

    }
    // to calculate eigenvalues calculate Coefficient   of cubic equation
  double a=1;
  double b=A[0][0]+A[1][1]+A[2][2];
  b=-b;
  double c=(A[0][0]*A[1][1])+(A[0][0]*A[2][2])+(A[1][1]*A[2][2])-((A[0][1]*A[0][1])+(A[0][2]*A[0][2])+(A[1][2]*A[1][2]));
  double d=(A[0][0]*A[1][1]*A[2][2])-((A[2][2]*A[0][1]*A[0][1])+(A[1][1]*A[0][2]*A[0][2])+(A[0][0]*A[1][2]*A[1][2]))+2*(A[0][1]*A[0][2]*A[1][2]);
  d=-d;
//   find roots of cubic equation having roots a,b,c,d
// roots of cubic equation will be eigenvalues

    vector<double> eigenvalues=find_roots(a,b,c,d);
    // cout<<"eigenvalues"<<endl;
    sort(eigenvalues.begin(),eigenvalues.end());

    // for(int i=0;i<eigenvalues.size();i++){
    //     cout<<eigenvalues[i]<<" ";
    // }
    // cout<<endl;
    // calculate eigenvector
    vector<double> eigenvectors=find_eigenvectors(A,eigenvalues[0]);
    // cout<<endl<<"eigenvectors corresponding to minimum eigenvalues"<<endl;
    // for(int i=0;i<3;i++){
    //     cout<<eigenvectors[i]<<" ";
    //  }
    cout<<"A";
       for(int i=0;i<points[point_index].size();i++){
         MyFile<<points[point_index][i]<<",";
       }
       for(int i=0;i<3;i++){
        MyFile<<eigenvectors[i]<<",";
       }
       MyFile<<"\n";

```

**Fig 3.3** part-3 of the normal code.

1.The read_data() function reads the point cloud data from the CSV file and returns it as a vector of vectors.

2.The find_neighbour_points() function finds the k-nearest neighboring points for a given target point in the point cloud data.

3.The covariance_matrix() function calculates the covariance matrix of a set of points.

4.The find_roots() function solves a cubic equation to find the roots, which represent the eigenvalues of the covariance matrix.

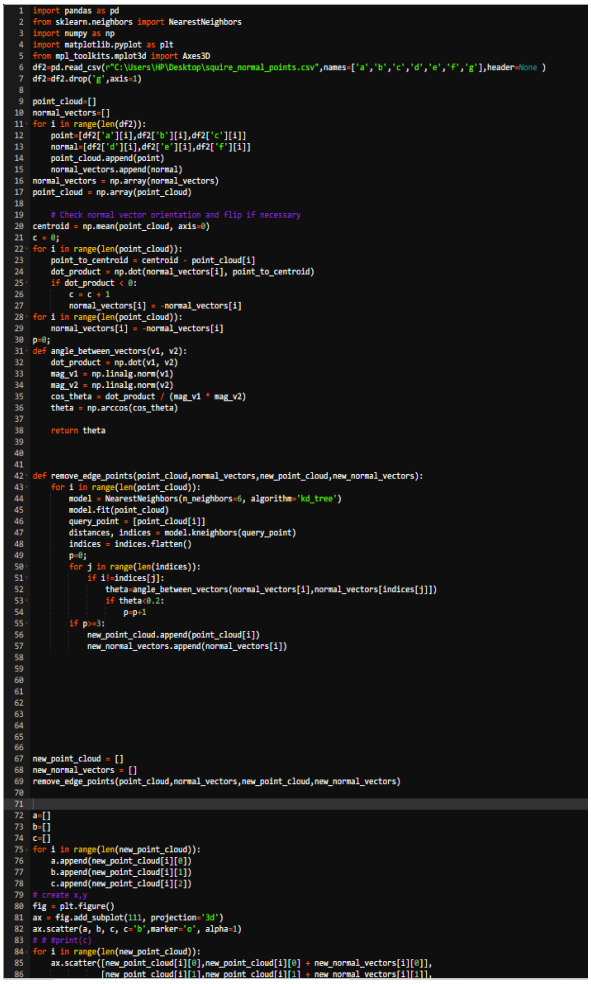
5.The find_eigenvectors() function calculates the eigenvectors corresponding to the minimum eigenvalue of the covariance matrix.

6.The main function iterates through each point in the point cloud, finds its neighboring points, calculates the covariance matrix, eigenvalues, and eigenvectors, and writes the point coordinates and normal vectors to the output CSV file.

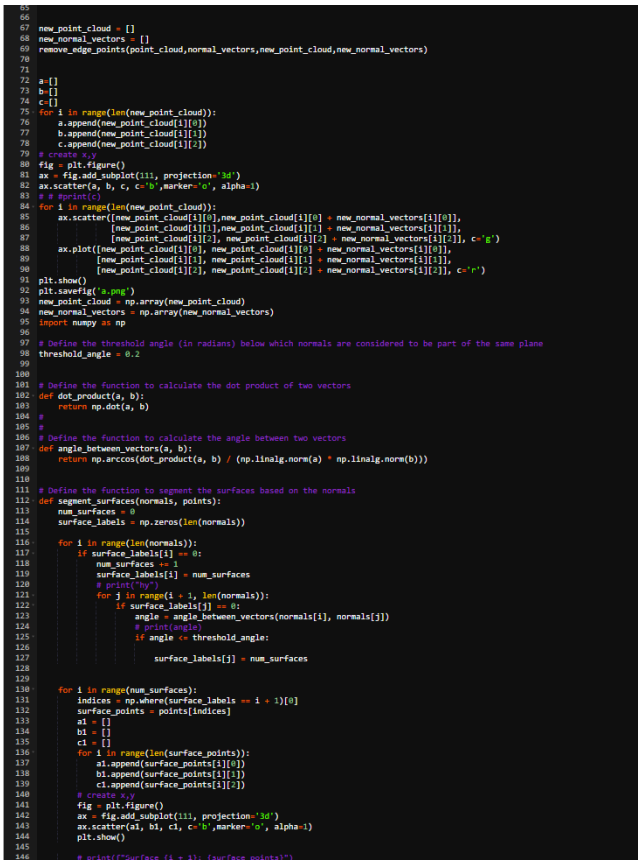
The code uses vector operations, file input/output, and basic linear algebra calculations to process the point cloud data and compute the normal vectors efficiently.

## 3.2 SEGMENTATION

Segmentation is done using python libraries like NearestNeighbors,matplotlib and Axes3D.



```
import pandas as pd
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
df2=pd.read_csv(r"C:\Users\HP\Desktop\squire_normal_points.csv",names=['a','b','c','d','e','f','g'],header=None )
df2=df2.drop('g',axis=1)

point_cloud=[]
normal_vectors=[]
for i in range(len(df2)):
    point=[df2['a'][i],df2['b'][i],df2['c'][i]]
    normal=[df2['d'][i],df2['e'][i],df2['f'][i]]
    point_cloud.append(point)
    normal_vectors.append(normal)
normal_vectors = np.array(normal_vectors)
point_cloud = np.array(point_cloud)

    # Check normal vector orientation and flip if necessary
centroid = np.mean(point_cloud, axis=0)
c = 0;
for i in range(len(point_cloud)):
    point_to_centroid = centroid - point_cloud[i]
    dot_product = np.dot(normal_vectors[i], point_to_centroid)
    if dot_product < 0:
        c = c + 1
        normal_vectors[i] = -normal_vectors[i]
for i in range(len(point_cloud)):
    normal_vectors[i] = -normal_vectors[i]
p=0;
def angle_between_vectors(v1, v2):
    dot_product = np.dot(v1, v2)
    mag_v1 = np.linalg.norm(v1)
    mag_v2 = np.linalg.norm(v2)
    cos_theta = dot_product / (mag_v1 * mag_v2)
    theta = np.arccos(cos_theta)

    return theta



def remove_edge_points(point_cloud,normal_vectors,new_point_cloud,new_normal_vectors):
    for i in range(len(point_cloud)):
        model = NearestNeighbors(n_neighbors=6, algorithm='kd_tree')
        model.fit(point_cloud)
        query_point = [point_cloud[i]]
        distances, indices = model.kneighbors(query_point)
        indices = indices.flatten()
        p=0;
        for j in range(len(indices)):
            if i!=indices[j]:
                theta=angle_between_vectors(normal_vectors[i],normal_vectors[indices[j]])
                if theta<0.2:
                    p=p+1
        if p>=3:
            new_point_cloud.append(point_cloud[i])
            new_normal_vectors.append(normal_vectors[i])









new_point_cloud = []
new_normal_vectors = []
remove_edge_points(point_cloud,normal_vectors,new_point_cloud,new_normal_vectors)


a=[]
b=[]
c=[]
for i in range(len(new_point_cloud)):
    a.append(new_point_cloud[i][0])
    b.append(new_point_cloud[i][1])
    c.append(new_point_cloud[i][2])
# create x,y
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(a, b, c, c='b',marker='o', alpha=1)
# # #print(c)
for i in range(len(new_point_cloud)):
    ax.scatter([new_point_cloud[i][0],new_point_cloud[i][0] + new_normal_vectors[i][0]],
               [new_point_cloud[i][1],new_point_cloud[i][1] + new_normal_vectors[i][1]],
```



```
new_point_cloud = []
new_normal_vectors = []
remove_edge_points(point_cloud,normal_vectors,new_point_cloud,new_normal_vectors)


a=[]
b=[]
c=[]
for i in range(len(new_point_cloud)):
    a.append(new_point_cloud[i][0])
    b.append(new_point_cloud[i][1])
    c.append(new_point_cloud[i][2])
# create x,y
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(a, b, c, c='b',marker='o', alpha=1)
# # #print(c)
for i in range(len(new_point_cloud)):
    ax.scatter([new_point_cloud[i][0],new_point_cloud[i][0] + new_normal_vectors[i][0]],
               [new_point_cloud[i][1],new_point_cloud[i][1] + new_normal_vectors[i][1]],
               [new_point_cloud[i][2], new_point_cloud[i][2] + new_normal_vectors[i][2]], c='g')
    ax.plot([new_point_cloud[i][0], new_point_cloud[i][0] + new_normal_vectors[i][0]],
            [new_point_cloud[i][1], new_point_cloud[i][1] + new_normal_vectors[i][1]],
            [new_point_cloud[i][2], new_point_cloud[i][2] + new_normal_vectors[i][2]], c='r')
plt.show()
plt.savefig('a.png')
new_point_cloud = np.array(new_point_cloud)
new_normal_vectors = np.array(new_normal_vectors)
import numpy as np

# Define the threshold angle (in radians) below which normals are considered to be part of the same plane
threshold_angle = 0.2


# Define the function to calculate the dot product of two vectors
def dot_product(a, b):
    return np.dot(a, b)
#
#
# Define the function to calculate the angle between two vectors
def angle_between_vectors(a, b):
    return np.arccos(dot_product(a, b) / (np.linalg.norm(a) * np.linalg.norm(b)))


# Define the function to segment the surfaces based on the normals
def segment_surfaces(normals, points):
    num_surfaces = 0
    surface_labels = np.zeros(len(normals))

    for i in range(len(normals)):
        if surface_labels[i] == 0:
            num_surfaces += 1
            surface_labels[i] = num_surfaces
            # print("hy")
            for j in range(i + 1, len(normals)):
                if surface_labels[j] == 0:
                    angle = angle_between_vectors(normals[i], normals[j])
                    # print(angle)
                    if angle <= threshold_angle:

                        surface_labels[j] = num_surfaces


    for i in range(num_surfaces):
        indices = np.where(surface_labels == i + 1)[0]
        surface_points = points[indices]
        a1 = []
        b1 = []
        c1 = []
        for i in range(len(surface_points)):
            a1.append(surface_points[i][0])
            b1.append(surface_points[i][1])
            c1.append(surface_points[i][2])
        # create x,y
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(a1, b1, c1, c='b',marker='o', alpha=1)
        plt.show()

        # print(f"Surface {i + 1}: {surface_points}")
```

**Fig 3.5** part-2 of Segmentation code

**Fig 3.4** part-1 of Segmentation code

The provided code performs surface segmentation on a point cloud dataset. It reads the dataset from a CSV file, extracts the points and their corresponding normal vectors, and checks the orientation of the normal vectors. Then, it removes edge points based on their proximity and orientation to neighboring points. Afterward, it visualizes the resulting point cloud and segmented surfaces using 3D plots. Finally, it applies surface segmentation by clustering points with similar normal vectors together, based on a threshold angle. Each surface is visualized separately. The code effectively segments surfaces in the point cloud dataset and provides visual representations of the segmented surfaces.

# 4 RESULT AND CONCLUSION

We have run the segmentation on two datasets ,one for testing the code and another for out own dataset we have generated using 3D scanner.

1.Cubical box dataset.

2.Staircase model dataset.

## 4.1 Result for cubical box

Some Dataset of cubical box is shown in the figure where first three columns are 3D coordinates of point cloud data and last 3 columns represent the direction cosines of normals of 3D points.

| | | | | | |
|---|---|---|---|---|---|
| 0.5 | -0.5 | 0.1667 | 0.7071 | -0.707 | ###### |
| -0.389 | -0.5 | -0.056 | 0 | 1 | 0 |
| 0.5 | -0.5 | -0.5 | -0.365 | 0.8569 | 0.3645 |
| 0.5 | -0.5 | 0.5 | 0.3645 | -0.857 | 0.3645 |
| 0.3889 | 0.2778 | -0.5 | 0 | 0 | 1 |
| 0.3889 | 0.2778 | 0.5 | 0 | 0 | 1 |
| -0.5 | 0.5 | -0.389 | 0 | 0 | 1 |
| -0.5 | 0.0556 | -0.389 | 1 | 0 | 0 |
| -0.056 | 0.5 | -0.389 | 0 | 1 | 0 |
| 0.5 | 0.2778 | 0.3889 | 1 | 0 | 0 |
| 0.5 | 0.2778 | -0.278 | 1 | 0 | 0 |
| -0.389 | 0.2778 | -0.5 | 0 | 0 | 1 |
| -0.389 | 0.2778 | 0.5 | 0 | 0 | 1 |
| -0.167 | -0.056 | -0.5 | 0 | 0 | 1 |
| -0.167 | -0.056 | 0.5 | 0 | 0 | 1 |
| 0.5 | 0.2778 | 0.0556 | 1 | 0 | 0 |
| -0.278 | 0.5 | -0.167 | 0 | 1 | 0 |
| 0.5 | 0.0556 | -0.167 | 1 | 0 | 0 |
| 0.0556 | -0.5 | 0.2778 | 0 | 1 | 0 |
| -0.5 | -0.278 | 0.1667 | 1 | 0 | 0 |
| 0.0556 | -0.5 | -0.056 | 0 | 1 | 0 |
| -0.5 | -0.278 | -0.5 | 0.7071 | ###### | 0.7071 |
| -0.5 | -0.278 | 0.5 | -0.707 | ###### | 0.7071 |
| -0.278 | -0.167 | -0.5 | 0 | 0 | 1 |
| -0.278 | -0.167 | 0.5 | 0 | 0 | 1 |
| -0.278 | -0.389 | -0.5 | 0 | 0 | 1 |
| -0.056 | -0.056 | -0.5 | 0 | 0 | 1 |
| -0.278 | -0.389 | 0.5 | 0 | 0 | 1 |
| -0.056 | -0.278 | -0.5 | 0 | 0 | 1 |
| -0.056 | -0.278 | 0.5 | 0 | 0 | 1 |
| -0.056 | -0.056 | 0.5 | 0 | 0 | 1 |
| -0.5 | -0.167 | 0.3889 | 1 | 0 | 0 |
| 0.1667 | 0.5 | -0.389 | 0 | 1 | 0 |
| -0.5 | -0.389 | 0.3889 | 1 | 0 | 0 |
| -0.5 | -0.167 | -0.278 | 1 | 0 | 0 |
| 0.0556 | -0.389 | -0.5 | 0 | 0 | 1 |
| 0.0556 | -0.389 | 0.5 | 0 | 0 | 1 |
| 0.5 | 0.5 | -0.389 | 0 | 0 | 1 |
| -0.5 | -0.389 | -0.278 | 1 | 0 | 0 |
| -0.5 | -0.167 | 0.0556 | 1 | 0 | 0 |
| -0.5 | -0.389 | 0.0556 | 1 | 0 | 0 |
| -0.5 | -0.056 | 0.2778 | 1 | 0 | 0 |
| -0.5 | 0.1667 | 0.2778 | 1 | 0 | 0 |
| -0.5 | 0.1667 | -0.056 | 1 | 0 | 0 |
| -0.5 | -0.056 | -0.056 | 1 | 0 | 0 |
| 0.5 | -0.389 | 0.2778 | 1 | 0 | 0 |
| 0.5 | -0.389 | -0.056 | 1 | 0 | 0 |
| 0.0556 | 0.2778 | -0.5 | 0 | 0 | 1 |
| 0.0556 | 0.2778 | 0.5 | 0 | 0 | 1 |
| 0.5 | -0.056 | 0.1667 | 1 | 0 | 0 |
| 0.5 | -0.278 | 0.1667 | 1 | 0 | 0 |
| 0.5 | -0.056 | -0.5 | -0.707 | ###### | 0.7071 |
| 0.5 | -0.056 | 0.5 | 0.7071 | ###### | 0.7071 |
| 0.1667 | -0.278 | -0.5 | 0 | 0 | 1 |
| 0.5 | -0.278 | -0.5 | -0.707 | ###### | 0.7071 |
| 0.5 | -0.278 | 0.5 | 0.7071 | ###### | 0.7071 |
| 0.5 | -0.167 | 0.3889 | 1 | 0 | 0 |
| 0.5 | -0.167 | -0.278 | 1 | 0 | 0 |

**Fig 4.1** Dataset of 3D point cloud of cubical box

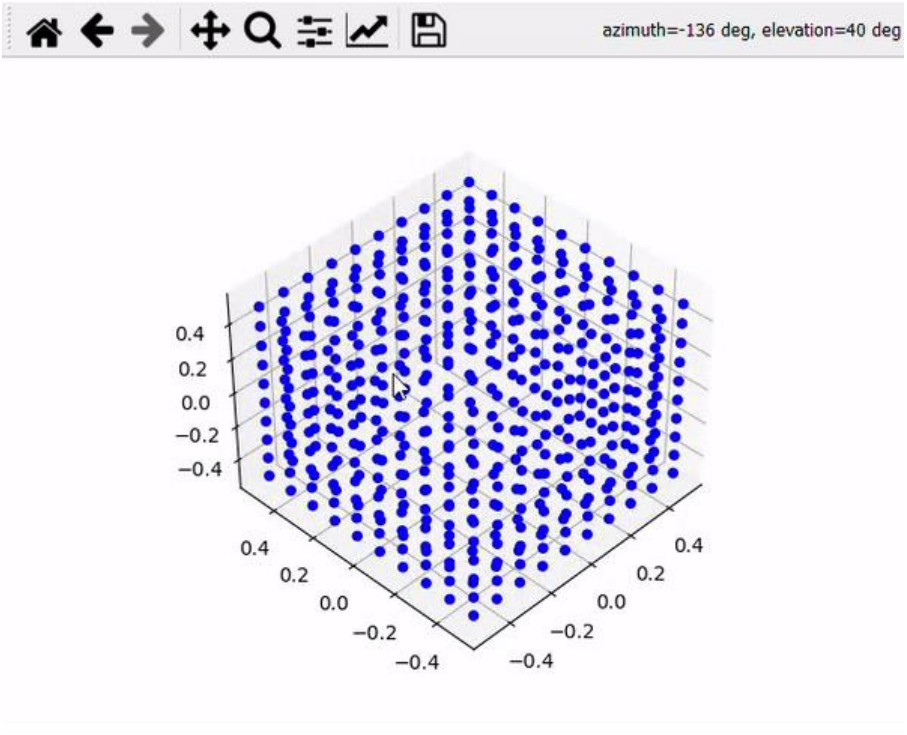



**Fig 4.2** Visual representation of the above dataset using pycharm.

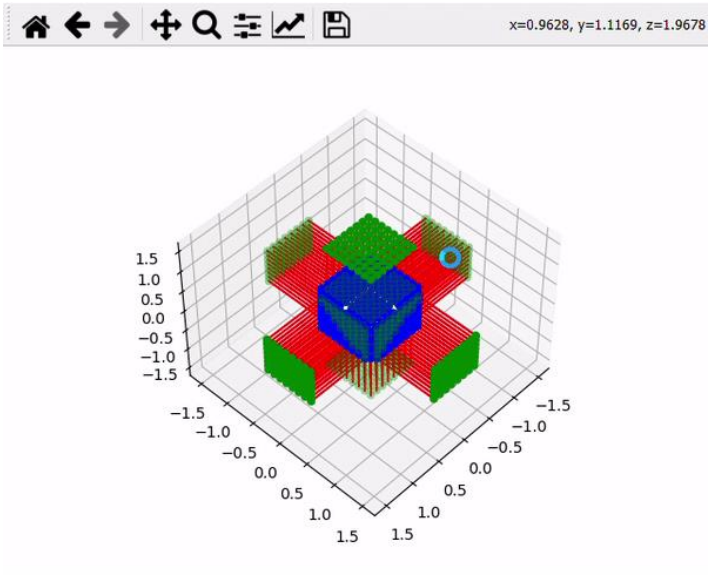



**Fig 4.3** Visualization of normals

## 4.1.1 Segmented faces of cube

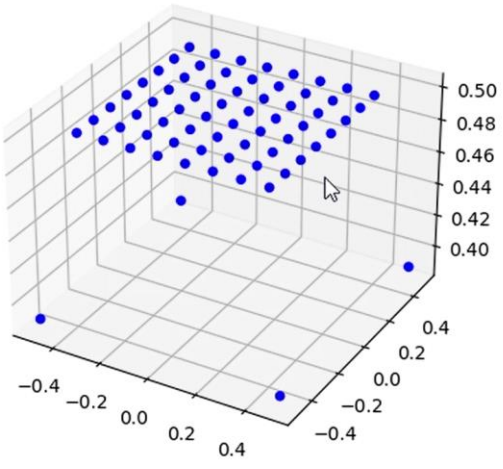


**Fig 4.4** segmented face-1

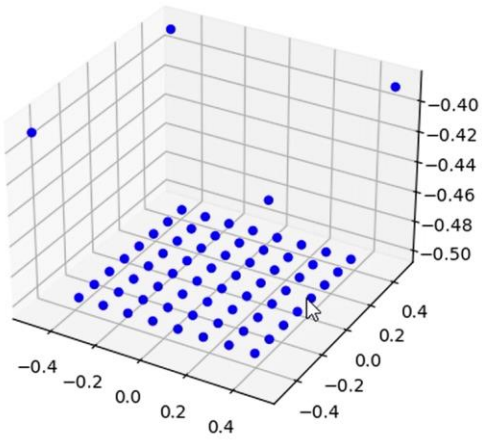
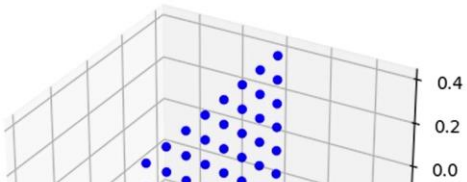


**Fig 4.5** segmented face-2



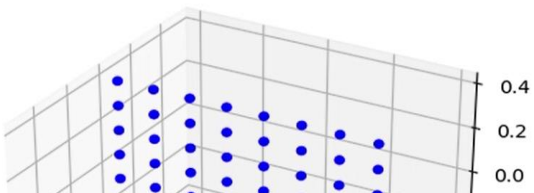


**Fig 4.6** segmented face-3

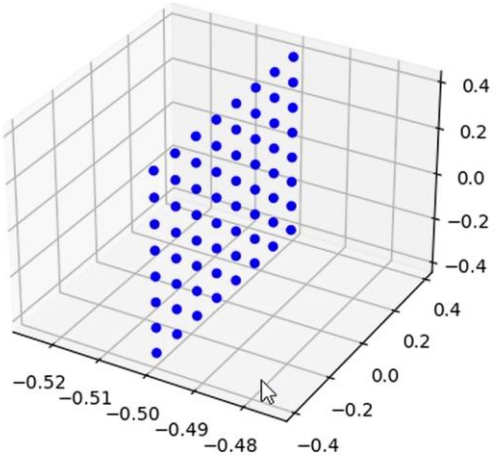



**Fig 4.7** segmented face-4

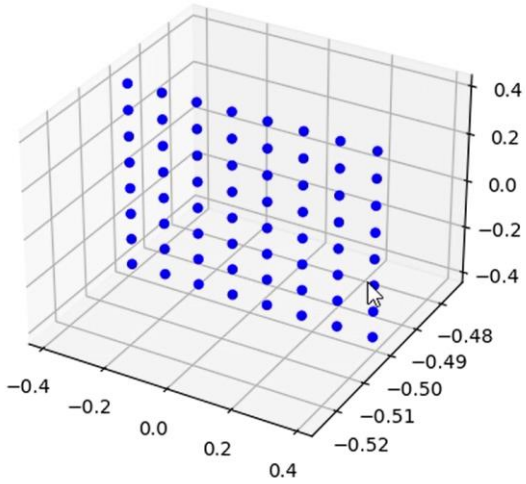



**Fig 4.8** segmented face-5

**Fig 4.9** segmented face-6

edge segmentation of cubical box

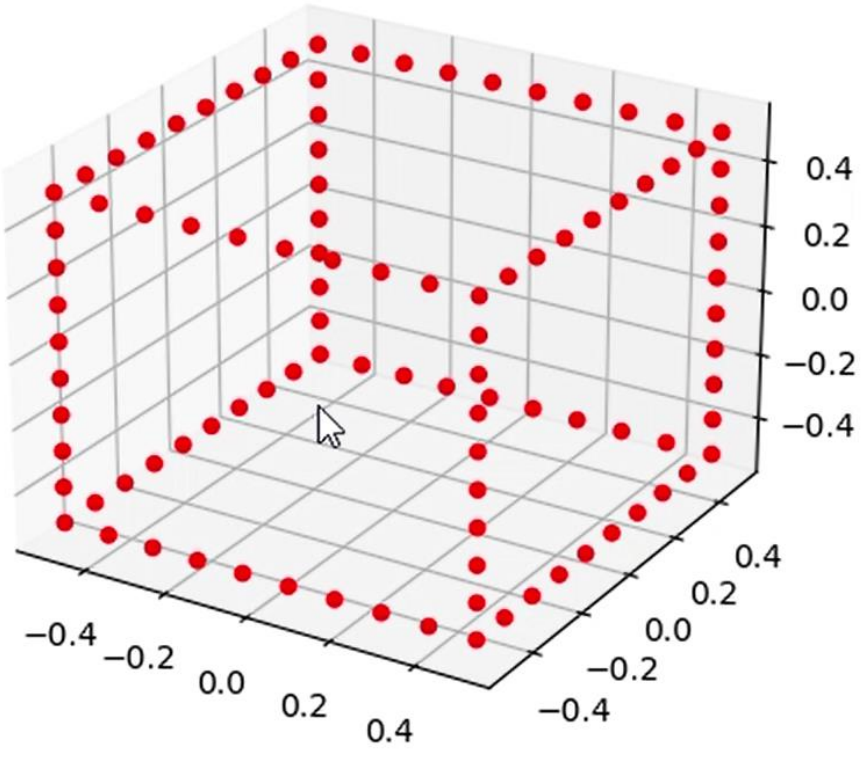


**Fig 4.10** segmented edges of cube

## 4.1.2 Edges of segmented surfaces

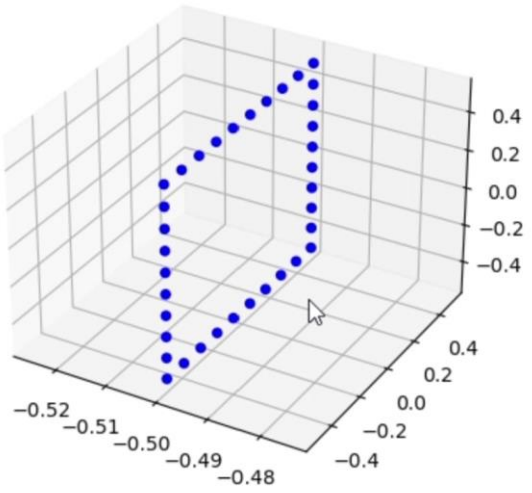



**Fig 4.11** segmented edge-1

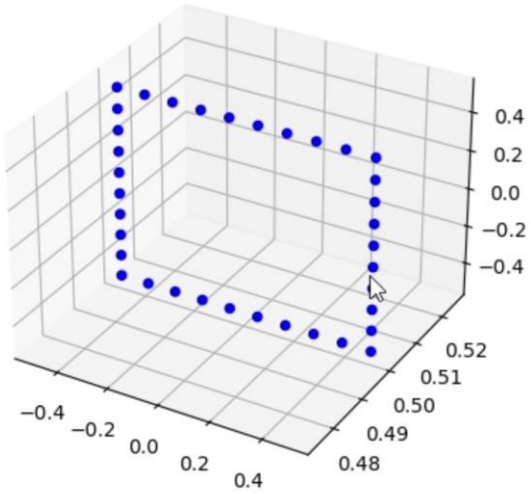



**Fig 4.12** segmented edge-2

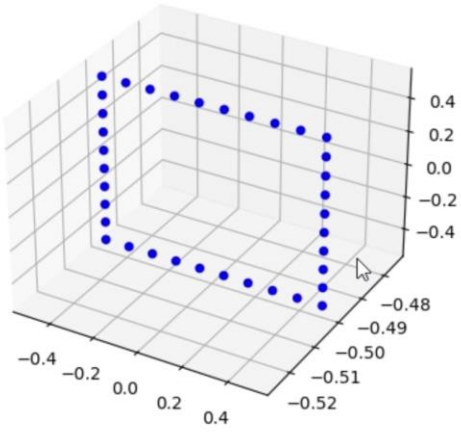



**Fig 4.13** segmented edge-3

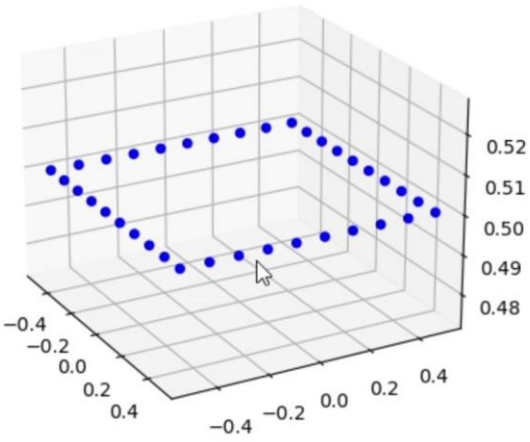



**Fig 4.14** segmented edge-4

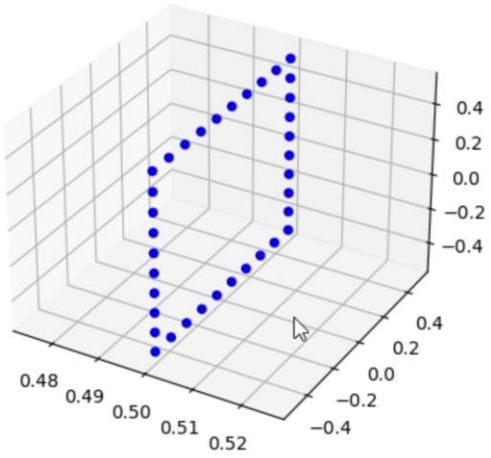



**Fig 4.15** segmented edge-5

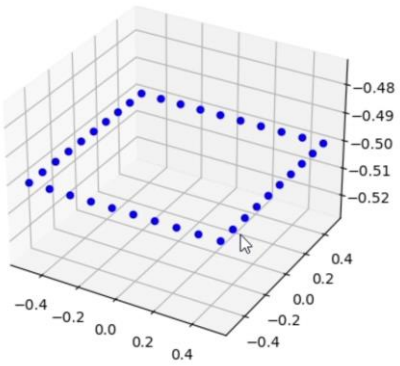



**Fig 4.16** segmented edge-6

## 4.2 Result for staircase model

### 4.2.1 STL representation of staircase model

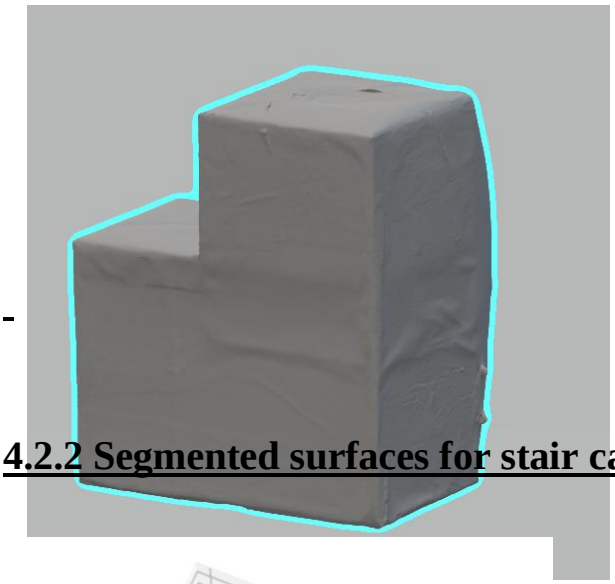
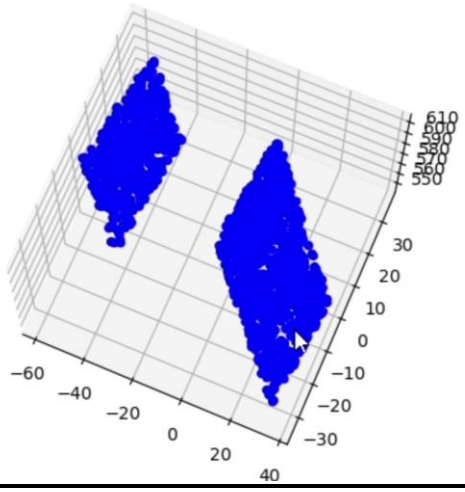


**fig 4.17** STL file of staircase model.

### 4.2.2 Segmented surfaces for stair case model



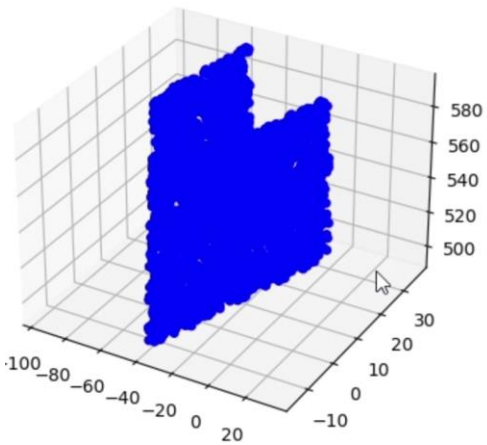



**Fig 4.18** segmented surfaces-1

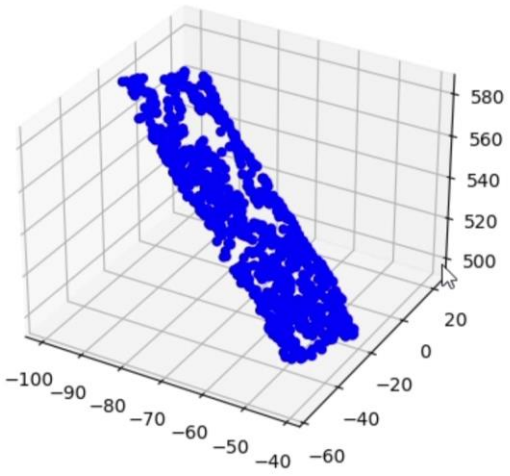



**Fig 4.19** segmented surface-2

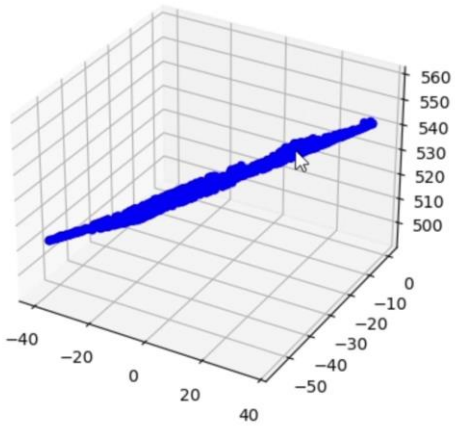



**Fig 4.20** segmented surface-3

**Fig 4.21** segmented surface-4

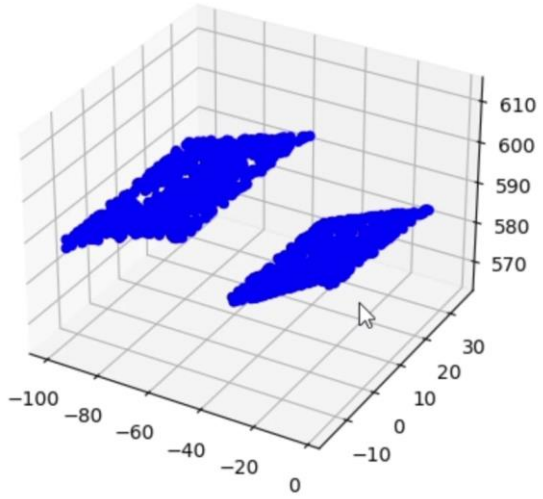



**Fig 4.22** segmented surface-5

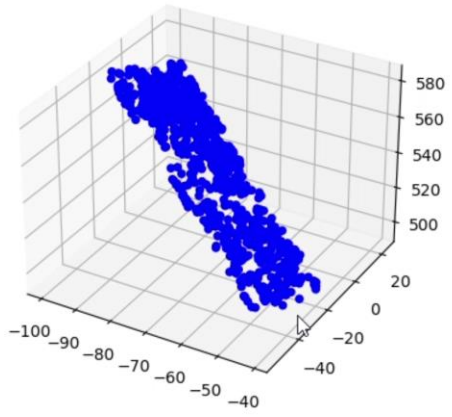



**Fig 4.23** segmented surface-6

## 4.2.3 Edge of staircase model

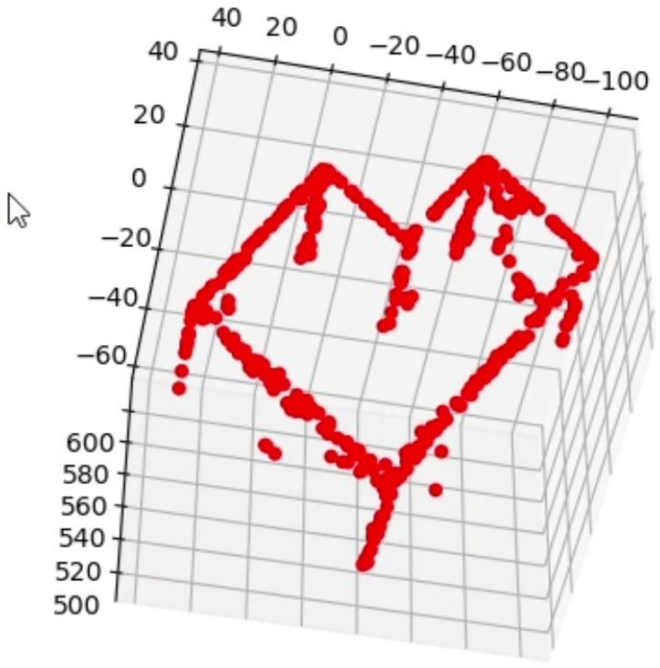



**Fig 4.33** Edge of staircase model.

## 4.3 Conclusion

This paper presents a normal-based segmentation technique for 3D point clouds, which effectively partitions the point cloud into meaningful regions or objects. The method demonstrates excellent performance in terms of accuracy and boundary preservation. Here, in the present work cubical dataset is a test data set used to check the functionality of the algorithm while staircase dataset is the main dataset which is obtained by scanning through 3D scanners. Staircase has eight faces. The normal generation is time consuming as the staircase dataset contains over 9000 points and the point cloud, we obtained for staircase model has numerous outlier points which we were able to pre-process and the post-processed dataset was subjected to segmentation. Finally, segmentation has been done based on developed algorithm. The obtained segmented region has been shown in result section. The proposed approach can be used significantly for numerous applications in computer vision, robotics, and reverse engineering where accurate segmentation of 3D point clouds is essential.

## 4.4 Future Works

The attribute used for segmentation here is normal based segmentation but for robust results we can use Feature-based segmentation where extracting meaningful features from curvature and deep learning based segmentation.

# 5 References

1. Grilli, E., Menna, F., & Remondino, F., 2017. A REVIEW OF POINT CLOUDS SEGMENTATION AND CLASSIFICATION ALGORITHMS. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* 339-344.

*2.* The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2019, *Volume XLII-2/W15, 2019 27th CIPA International Symposium "Documenting the past for a better future", 1–5 September 2019, Ávila, Spain.*

*3. Klasing, K., Althoff, D., Wollherr, D. and Buss, M., 2009. Com- parison of surface normal estimation methods for range sensing applications. Proc. IEEE International Conference on Robotics and Automation, pp. 3206–3211.*

*4. Lu, X., Yao, J., Tu, J., Li, K., Li, L., & Liu, Y., 2016. Pairwise Linkage for Point Clouds Segmentation. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol.III(3), pp. 201-208.*

*5. Sapkota, P. P., 2008. Segmentation of Coloured Point Clouds Data. PhD Thesis, International Institute for Geo-Information Science and Earth Observation, Enschede, The Netherlands.*

*6. Sappa, A. D., & Devy, M., 2001. Fast range image segmentation by an edge detection strategy. Proc. IEEE 3rd 3-D Digital Imaging and Modeling, pp. 292-299.*

*7. Sithole, G. and Vosselman, G., 2004. Experimental comparison of filter algorithms for bare-Earth extraction from airborne laser scanning point clouds. ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 59(1), pp.85-101.*