In [562]:

```
import numpy as np

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import os
import pythreejs
```

пример чтения .pcd и визуализации с помощью open3d

In [563]:

```
import open3d as o3d
```

In [84]:

```
pcd_load = o3d.io.read_point_cloud("data_1\\0000000006.pcd")
xyz_load = np.asarray(pcd_load.points)
```

In [85]:

```

o3d.visualization.draw_geometries([pcd_load])
    # convert Open3D.o3d.geometry.PointCloud to numpy array
print('xyz_load')
print(xyz_load)
```

```
xyz_load
[[77.1309967   7.21600008  2.829     ]
 [77.10199738  7.45699978  2.82800007]
 [76.58599854  8.01500034  2.81299996]
 ...
 [ 3.75       -1.41199994 -1.75300002]
 [ 3.74600005 -1.39699996 -1.74800003]
 [ 0.          0.          0.        ]]
```

Пример чтения .pcd и визуализации с помощью pyntcloud

In [5]:

```
import pyntcloud
import importlib
pyntcloud = importlib.reload(pyntcloud)
# from pyntcloud import PyntCloud
```

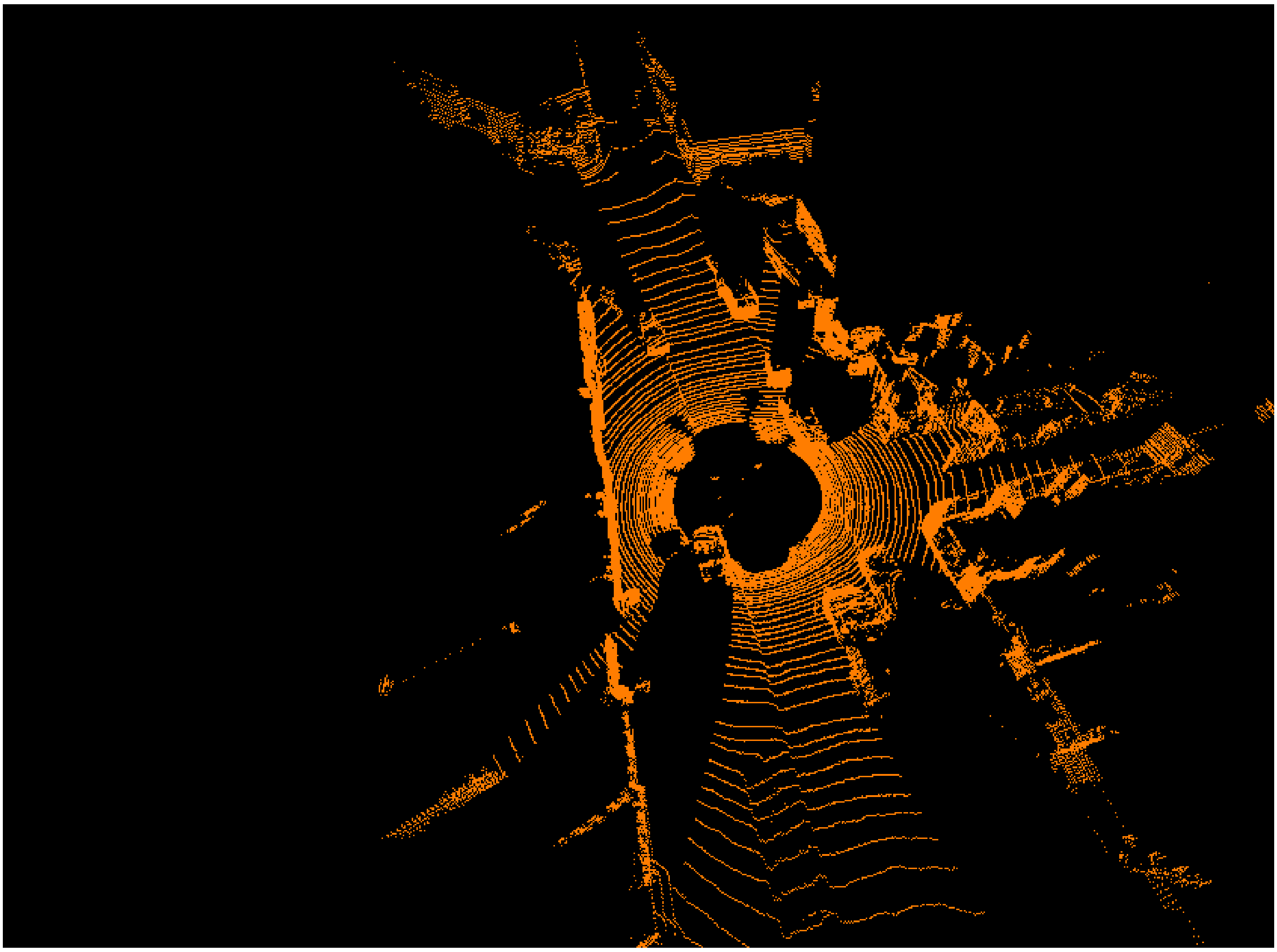
In [565]:

```
test_cloud = pyntcloud.PyntCloud.from_file("data_1\\0000000002.pcd")
test_cloud
```

Out[565]:

```
PyntCloud
116651 points with 1 scalar fields
0 faces in mesh
0 kdtrees
0 voxelgrids
Centroid: -0.6957642436027527, 1.0117207765579224, -1.059058427810669
Other attributes:
```

```
In [566]: # pythreejs надо установить с помощью pip а потом активировать в jupyter , загугли
          test_cloud.plot(backend='pythreejs')
```



Point … ○ 0.00 Background … black ■

```
In [ ]:
```

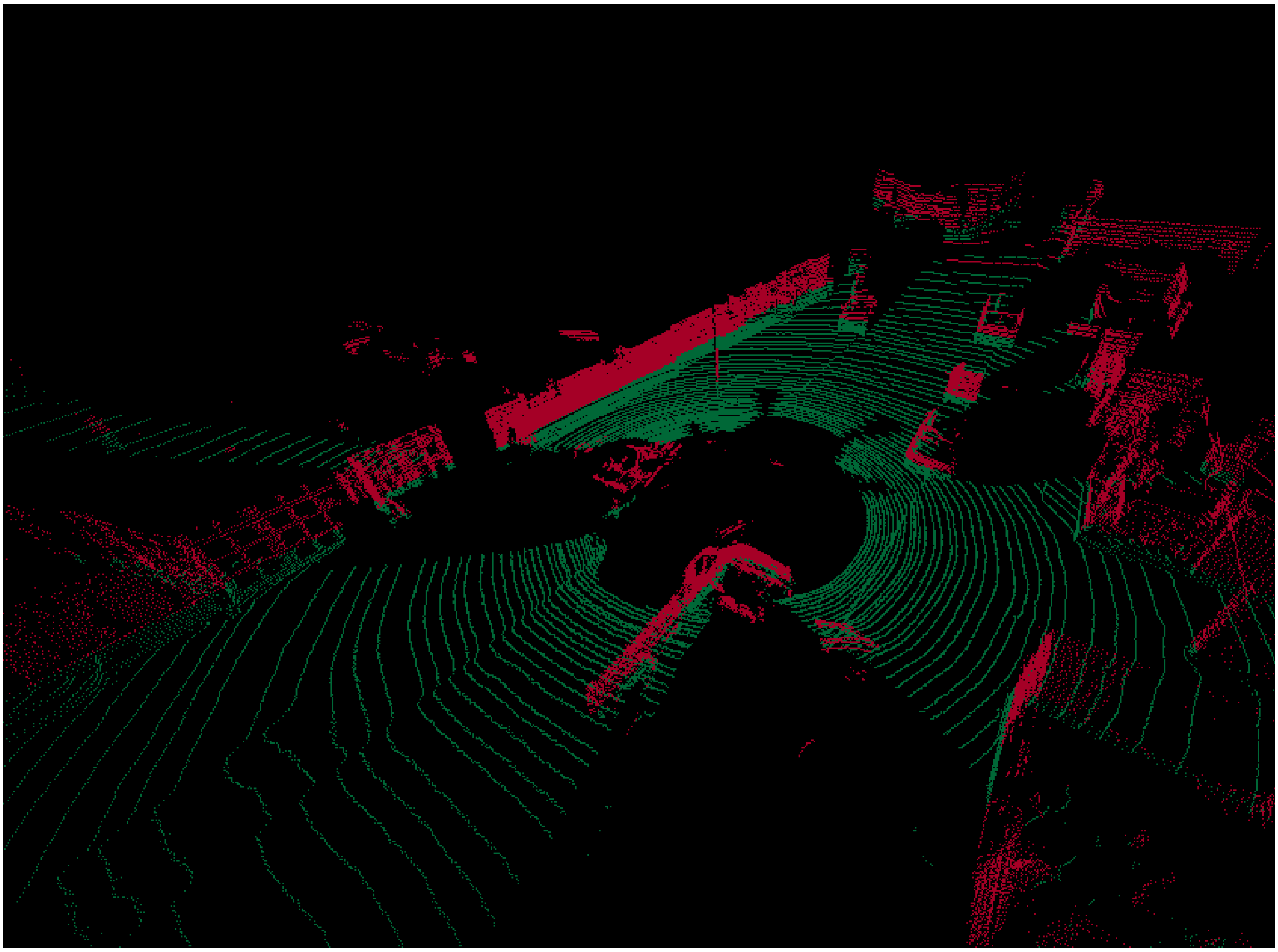
Дальше я пытаюсь в пайплайн

```
In [535]: cloud = pyntcloud.PyntCloud.from_file("data_1\\0000000005.pcd")
```

```
In [536]: cloud.xyz.shape

Out[536]: (114969, 3)
```

```
In [567]: # фильтр пола дороги, надо убрать чтобы потом искались норм кластеры, иначе будет
          is_floor = cloud.add_scalar_field("plane_fit",max_dist=0.55, max_iterations=200,
                                            n_inliers_to_stop=len(cloud.points))
```

```
In [538]: cloud.plot(use_as_color=is_floor, cmap="RdYlGn", output_name="is_floor")
```



Point … ○ 0.00 Background … black ■

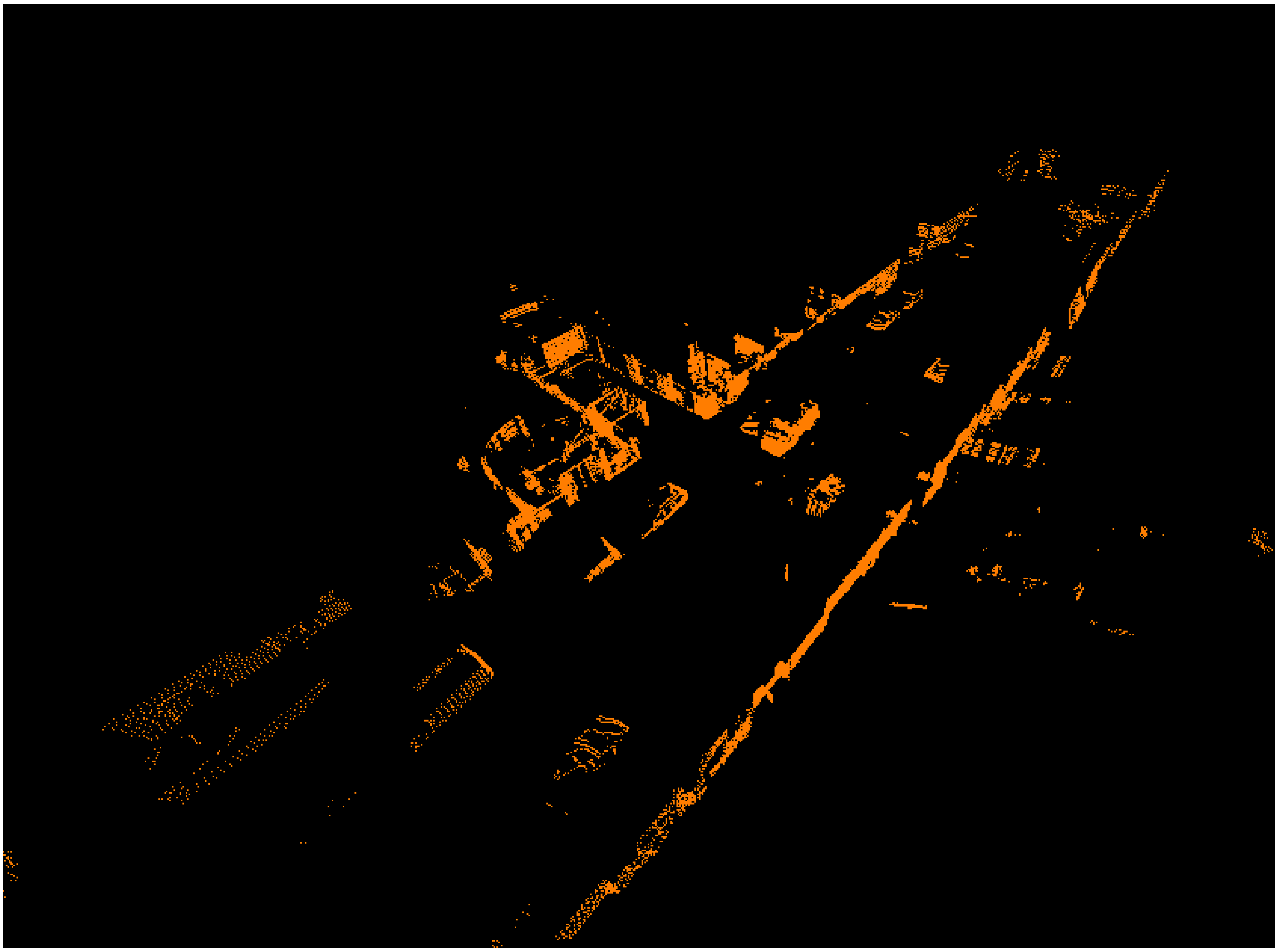
```
In [539]: not_floor = cloud.points[is_floor] != 1
```

```
In [540]: cloud.apply_filter(not_floor)
```

```
In [541]: cloud.xyz.shape

Out[541]: (53596, 3)
```

In [542]:

```
cloud.plot()
```



Point … 0.00 Background … black

от запуска ячейки ниже ничего не меняется

In [543]:

```
# есть пример работы KDTREE , как это влияет на кластеризацию?-пока походу никак,
kdtree_id = cloud.add_structure("kdtree",leafsize = 10)

# в гайде ее используют для К соседей, нужны ли они нам???...
k_neighbors = cloud.get_neighbors(k=50, kdtree=kdtree_id)
#  scalar_fields
ev = cloud.add_scalar_field("eigen_values", k_neighbors=k_neighbors)
```

In [ ]:

```

```

In [462]:

```
# продолжаем пайплайн
cloud
```

Out[462]:

```
PyntCloud
33016 points with 2 scalar fields
0 faces in mesh
1 kdtrees
0 voxelgrids
Centroid: -2.1837236881256104, 0.7934628129005432, -0.10891569405794144
Other attributes:
```

In [544]:

```
# можно образать дорогу по оси x и возможно вообще по любой оси, это как минимум н
# число ненужных точек и хоть как-то ускорить кластеризацию
f = cloud.get_filter("BBOX", min_x=-35, max_x=30)
```

In [546]:

```
cloud.apply_filter(f)
```

In [571]:

```
# вокселизируем на 40 вокселей по каждой оси, т.е. будет всего 40*40*40 кубов , вм
# чем больше вокселей тем точнее кластеризируется и тем медленее работает кластери
n_voxels = 40
voxelgrid_id = cloud.add_structure("voxelgrid", n_x=n_voxels, n_y=n_voxels, n_z=n_
```

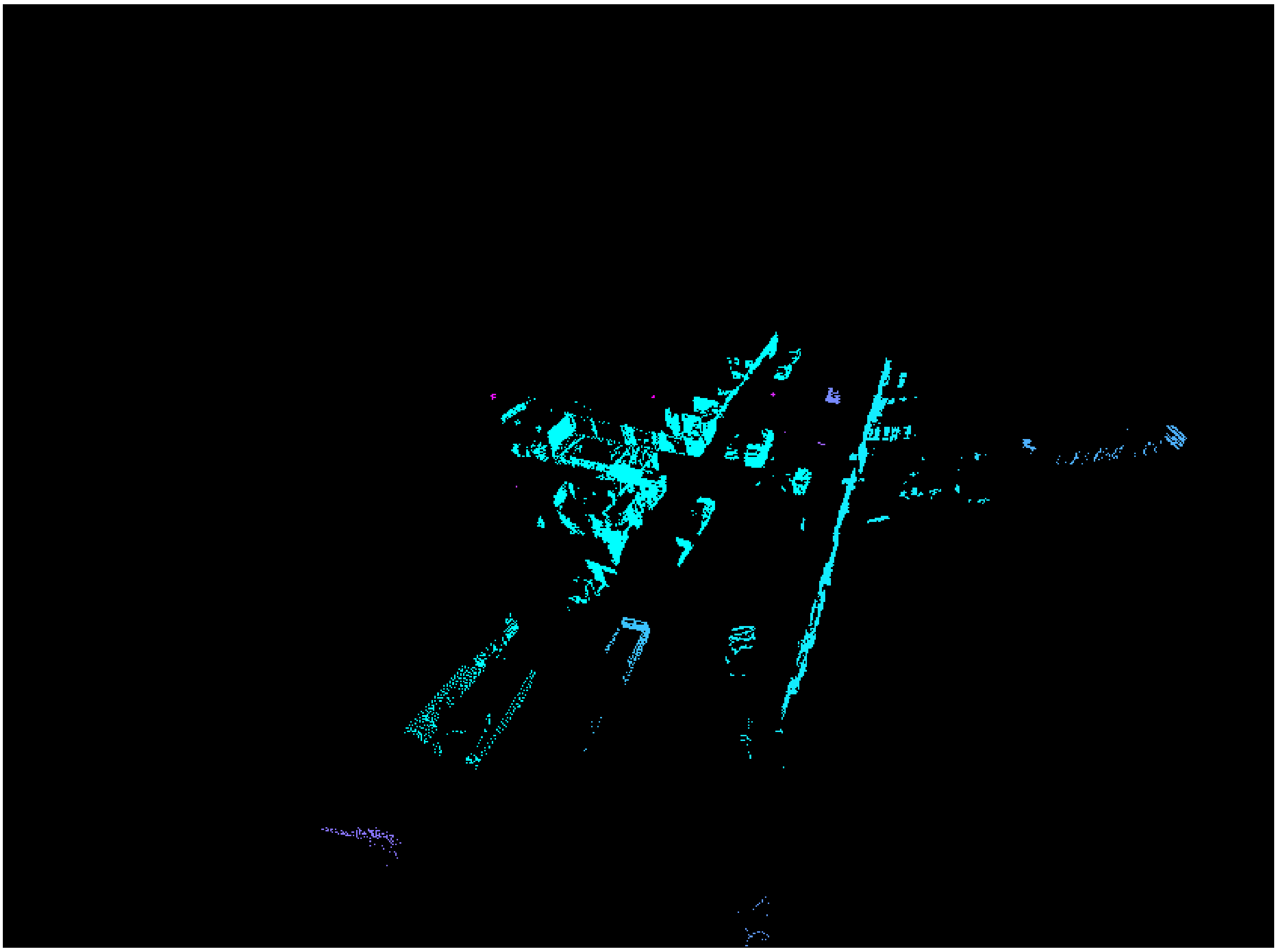
In [578]:

```
# вот она кластеризация, хромая да хоть какая...
%time cluster_id = cloud.add_scalar_field("euclidean_clusters",voxelgrid_id=voxelg
```

```
Wall time: 18.5 s
```

In [577]:

```
# визуализируем ее результаты
cloud.plot(use_as_color=cluster_id, cmap="cool")
```



Point … ○ 0.00 Background … black ■

```
In [582]: voxel_grid = cloud.structures['V([40, 40, 40],[None, None, None],True)']
```

```
In [595]: voxel_grid.shape
```

```
Out[595]: [1.7770000457763673, 1.7770000457763673, 1.7770000457763673]
```

```
In [597]: # С*КА Я ПОЛТАРА ЧАСА ИСКАЛ КУДА ЭТИ КЛАСТЕРЫ СОХРАНЯЮТСЯ ПРИКИНЬ!!!!
          cloud.points
```

Out[597]:

	x	y	z	intensity	is_plane	clusters(V([35, 35, 35], [None, None, None],True))	clusters(V([40, 40, 40], [None, None, None],True))
0	29.785999	8.353	1.255	0.33	0	0.0	2.0
1	29.466999	8.363	1.245	0.35	0	0.0	2.0
2	29.150999	8.372	1.235	0.33	0	0.0	2.0
3	28.846001	8.333	1.224	0.35	0	0.0	2.0
4	28.551001	8.345	1.215	0.35	0	0.0	2.0
...	...	...	...	...	...	...	...
50537	1.564000	-1.124	-0.782	0.00	1	0.0	2.0
50538	1.502000	-1.071	-0.744	0.00	1	0.0	2.0
50539	1.508000	-1.072	-0.747	0.00	1	0.0	2.0
50540	1.441000	-0.979	-0.696	0.00	1	0.0	2.0
50541	0.000000	0.000	0.000	0.00	1	0.0	2.0

50542 rows × 7 columns

все что ниже осталось от PCL, так как это изначально PCK ноутбук с примером,

там пайплайн заветный

```
In [100]: cloud_x = cloud_np[:, 0]
          cloud_y = cloud_np[:, 1]
          cloud_z = cloud_np[:, 2]

          x_max, x_min = np.max(cloud_x), np.min(cloud_x)
          y_max, y_min = np.max(cloud_y), np.min(cloud_y)
          z_max, z_min = np.max(cloud_z), np.min(cloud_z)

          print('x_max: ', x_max,  ', x_min: ', x_min)
          print('y_max: ', y_max, ', y_min: ', y_min)
          print('z_max: ', z_max, ', z_min: ', z_min)
          print('Number of points: ', cloud_np.shape)
```

```
x_max:  249 , x_min:  0
y_max:  180 , y_min:  69
z_max:  148 , z_min:  101
Number of points:  (5185, 3)
```

In [17]:

```
# cloud_XYZ = pcl.PointCloud()
# cloud_XYZ.from_array(cloud_np[:, 0:3])
```

In [101]:

```
colors = {
    'Car': 'b',
    'Tram': 'r',
    'Cyclist': 'g',
    'Van': 'c',
    'Truck': 'm',
    'Pedestrian': 'y',
    'Sitter': 'k'
}
axes_limits = [
    [int(x_min*1.2), int(x_max*1.2)], # X axis range
    [int(y_min*1.2), int(y_max*1.2)], # Y axis range
    [-5, 5]   # Z axis range
]
```

In [102]:

```
axes_str = ['X', 'Y', 'Z']
```

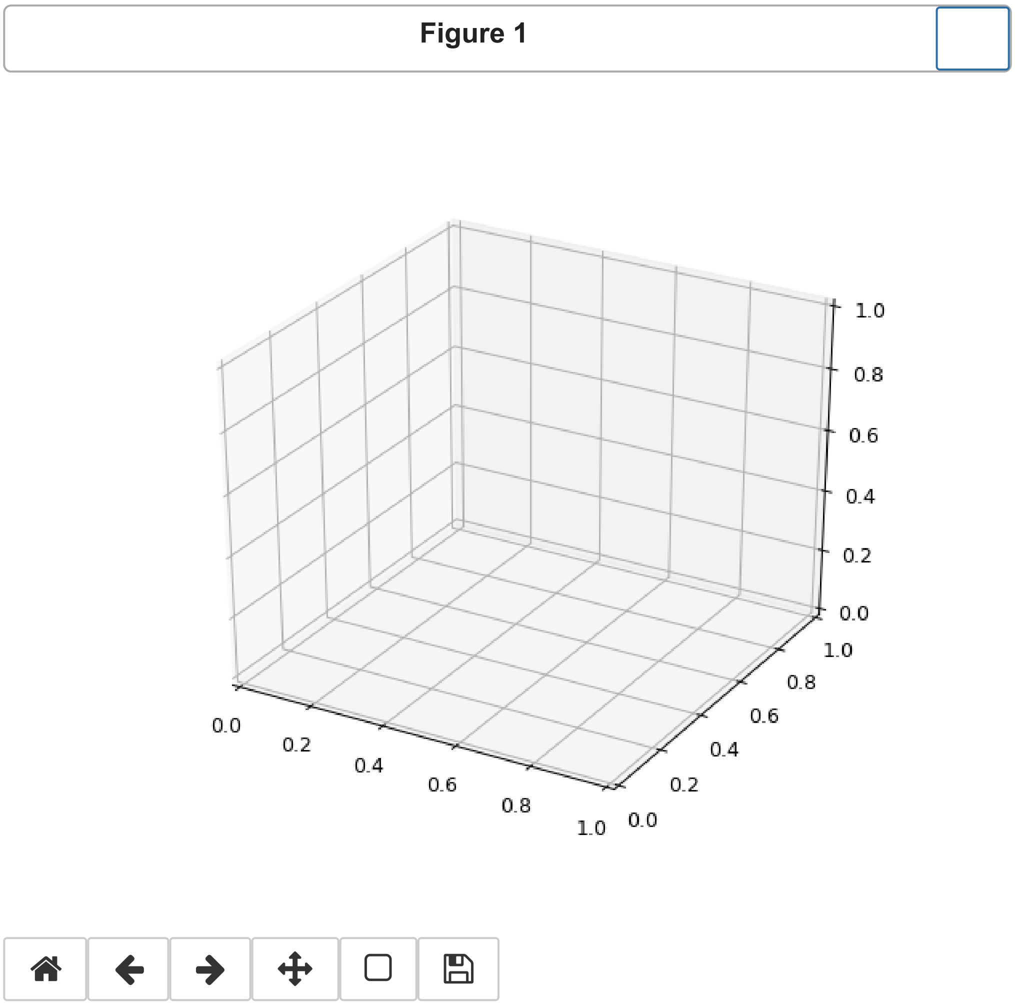
In [103]:

```
def draw_box(pyplot_axis, vertices, axes=[0, 1, 2], color='red'):
    """
    Draws a bounding 3D box in a pyplot axis.

    Parameters
    ----------
    pyplot_axis : Pyplot axis to draw in.
    vertices    : Array 8 box vertices containing x, y, z coordinates.
    axes        : Axes to use. Defaults to `[0, 1, 2]`, e.g. x, y and z axes.
    color       : Drawing color. Defaults to `black`.
    """
    vertices = vertices[axes, :]
    connections = [
        [0, 1], [1, 2], [2, 3], [3, 0],  # Lower plane parallel to Z=0 plane
        [4, 5], [5, 6], [6, 7], [7, 4],  # Upper plane parallel to Z=0 plane
        [0, 4], [1, 5], [2, 6], [3, 7]  # Connections between upper and lower plan
    ]
    for connection in connections:
        pyplot_axis.plot(*vertices[:, connection], c=color, lw=0.5)
```

In [127]:

```
def draw_point_cloud(cloud, ax, title, axes=[0, 1, 2], xlim3d=None, ylim3d=Non

        cloud = np.array(cloud) # Covert point cloud to numpy array
        no_points = np.shape(cloud)[0]
        point_size = 10**(3- int(np.log10(no_points))) # Adjust the point size
        if np.shape(cloud)[1] == 4: # If point cloud is XYZI format (e.g., I s
            ax.scatter(*np.transpose(cloud[:, axes]), s = point_size, c=cloud[
        elif np.shape(cloud)[1] == 3:   # If point cloud is XYZ format
            ax.scatter(*np.transpose(cloud[:, axes]), s = point_size, c='b', a
        ax.set_xlabel('{} axis'.format(axes_str[axes[0]]))
        ax.set_ylabel('{} axis'.format(axes_str[axes[1]]))
#         if len(axes) > 2: # 3-D plot
#             ax.set_xlim3d(axes_limits[axes[0]])
#             ax.set_ylim3d(axes_limits[axes[1]])
#             ax.set_zlim3d(axes_limits[axes[2]])
#             ax.set_zlabel('{} axis'.format(axes_str[axes[2]]))
#         else: # 2-D plot
#             ax.set_xlim(*axes_limits[axes[0]])
#             ax.set_ylim(*axes_limits[axes[1]])
        # User specified limits
        if xlim3d!=None:
            ax.set_xlim3d(xlim3d)
        if ylim3d!=None:
            ax.set_ylim3d(ylim3d)
        if zlim3d!=None:
            ax.set_zlim3d(zlim3d)
        ax.set_title(title)
```

In [130]:
```
# %matplotlib inline
%matplotlib notebook
f = plt.figure(figsize=(7, 6))
# f = plt.figure()
ax = f.add_subplot(111, projection='3d')
draw_point_cloud(np.unique(test,axis=0), ax, 'Point Cloud', xlim3d=(-80,80))
plt.show()
```



```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-130-89d00a3726cd> in <module>
      4 # f = plt.figure()
      5 ax = f.add_subplot(111, projection='3d')
----> 6 draw_point_cloud(cloud.xyz[f], ax, 'Point Cloud', xlim3d=(-80,80))
      7 plt.show()

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None
`) and integer or boolean arrays are valid indices
```

In [82]:
```
import pptk
```

In [125]:
```
# v = pptk.viewer(np.unique(test,axis=0))
v = pptk.viewer(cloud.xyz[f])
v.set(point_size=0.05)
```

In [70]:

```
# f, ax3 = plt.subplots(3, 1, figsize=(12, 25))
# draw_point_cloud(cloud_np,
#         ax3[0],
#         'XZ projection (Y = 0)',
#         axes=[0, 2] # X and Z axes
#     )
# draw_point_cloud(cloud_np,
#         ax3[1],
#         'XY projection (Z = 0)',
#         axes=[0, 1] # X and Y axes
#     )
# draw_point_cloud(cloud_np,
#         ax3[2],
#         'YZ projection (X = 0)',
#         axes=[1, 2] # Y and Z axes
#     )
# plt.show()
```

In [10]:

```
def voxel_filter(cloud, leaf_sizes):
    """
    Input parameters:
    cloud: input point cloud to be filtered
    leaf_sizes: a list of leaf_size for X, Y, Z
    Output:
    cloud_voxel_filtered: voxel-filtered cloud
    """
    sor = cloud.make_voxel_grid_filter()
    size_x, size_y, size_z = leaf_sizes
    sor.set_leaf_size(size_x, size_y, size_z)
    cloud_voxel_filtered = sor.filter()

    return cloud_voxel_filtered
```

In [11]:

```
cloud_voxel_filtered = voxel_filter(cloud_XYZ, [0.3, 0.3, 0.3])
print('Input cloud size: ', cloud_XYZ.size, ', size after voxel-filtering: ', clou
```

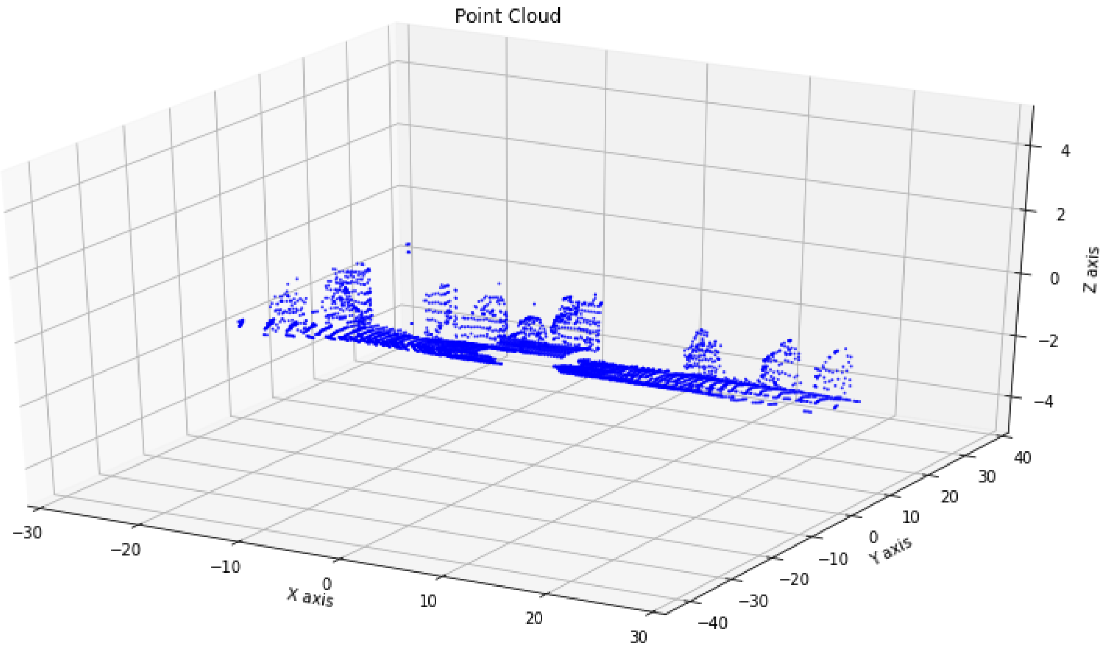
```
Input cloud size:  114396 , size after voxel-filtering:  14230
```

```
In [12]: def roi_filter(cloud, x_roi, y_roi, z_roi):
             """
             Input Parameters:
                 cloud: input point cloud
                 x_roi: ROI range in X
                 y_roi: ROI range in Y
                 z_roi: ROI range in Z

             Output:
                 ROI region filtered point cloud
             """
             clipper = cloud.make_cropbox()
             cloud_roi_filtered= pcl.PointCloud()
             xc_min, xc_max = x_roi
             yc_min, yc_max = y_roi
             zc_min, zc_max = z_roi
             clipper.set_MinMax(xc_min, yc_min, zc_min, 0, xc_max, yc_max, zc_max, 0)
             cloud_roi_filtered =clipper.filter()
             return cloud_roi_filtered
```

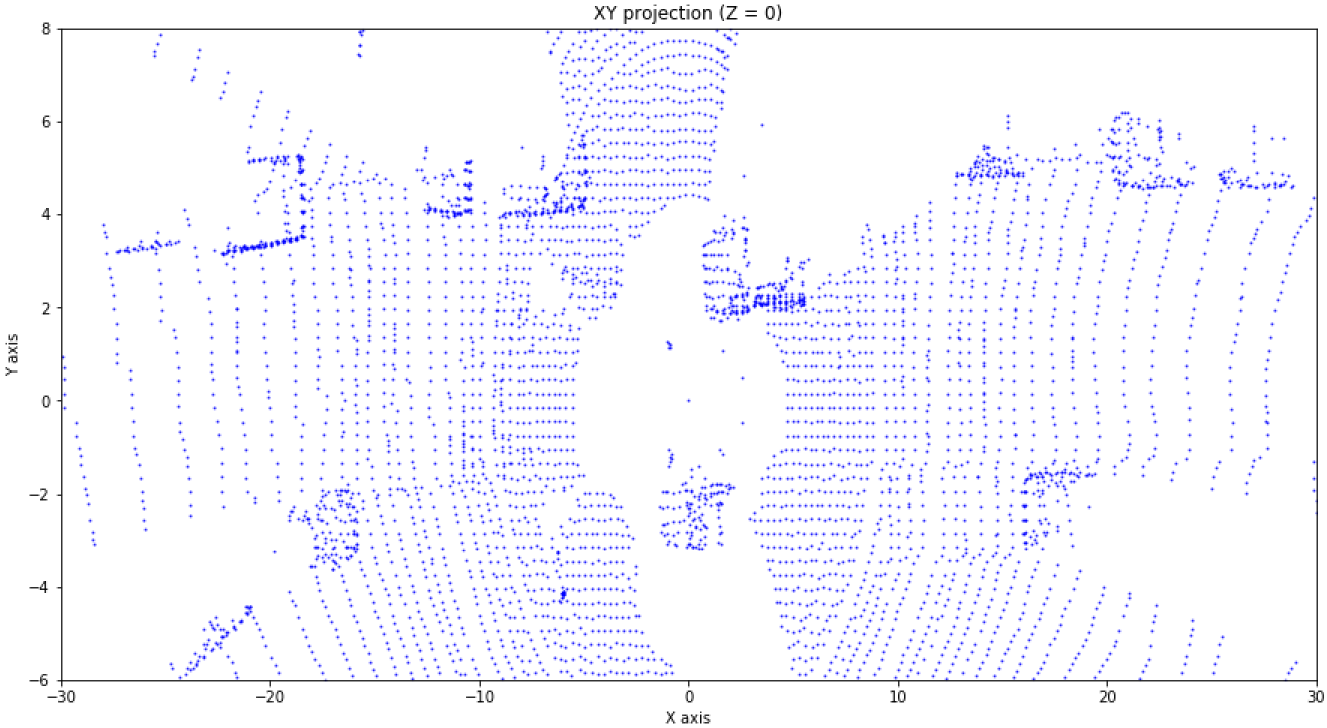
```
In [13]: cloud_roi_filtered = roi_filter(cloud_voxel_filtered, [-30, 30], [-6, 8], [-3, 3])
         print('Input cloud size: ', cloud_XYZ.size)
         print('Size after voxel-filtering: ', cloud_voxel_filtered.size)
         print('Size after ROI filter: ', cloud_roi_filtered.size)
```

```
Input cloud size:  114396
Size after voxel-filtering:  14230
Size after ROI filter:  4531
```

```
In [14]: f = plt.figure(figsize=(15, 8))
         ax = f.add_subplot(111, projection='3d')
         draw_point_cloud(cloud_roi_filtered, ax, 'Point Cloud', xlim3d=(-30,30))
         plt.show()
```



In [15]:
```
axes_limits = [
    [-30, 30], # X axis range
    [-6, 8], # Y axis range
    [-3, 3]    # Z axis range
]
f = plt.figure(figsize=(15, 8))
ax = f.add_subplot(111)
draw_point_cloud(cloud_roi_filtered,
        ax,
        'XY projection (Z = 0)',
        axes=[0, 1])
plt.show()
```



In [16]:
```
def plane_segmentation(cloud, dist_thold, max_iter):
    """
    Input parameters:
        cloud: Input cloud
        dist_thold: distance threshold
        max_iter: maximal number of iteration
    Output:
        indices: list of indices of the PCL points that belongs to the plane
        coefficient: the coefficients of the plane-fitting (e.g., [a, b, c, d] for
    """
    seg = cloud.make_segmenter_normals(ksearch=50)# For simplicity,hard coded
    seg.set_optimize_coefficients(True)
    seg.set_model_type(pcl.SACMODEL_NORMAL_PLANE)
    seg.set_method_type(pcl.SAC_RANSAC)
    seg.set_distance_threshold(dist_thold)
    seg.set_max_iterations(max_iter)
    indices, coefficients = seg.segment()
    return indices, coefficients
```

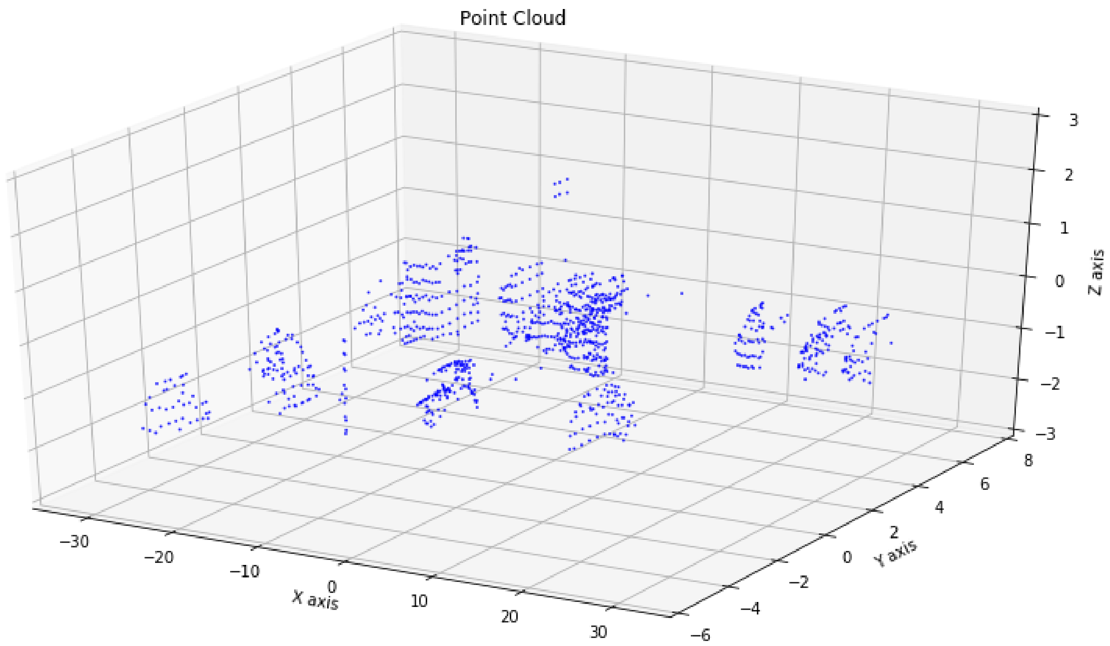
```
In [17]: indices, coefficients = plane_segmentation(cloud_roi_filtered, 0.3, 100)
         if len(indices) == 0:
                 print('Could not estimate a planar model for the given dataset.')
         print('Model coefficients: ' + str(coefficients[0]) + ', ' + str(
                 coefficients[1]) + ', ' + str(coefficients[2]) + ', ' + str(coefficients[3
```

```
Model coefficients: -0.0030953530222177505, 0.021372323855757713, 0.99976676702
49939, 1.7516109943389893
```

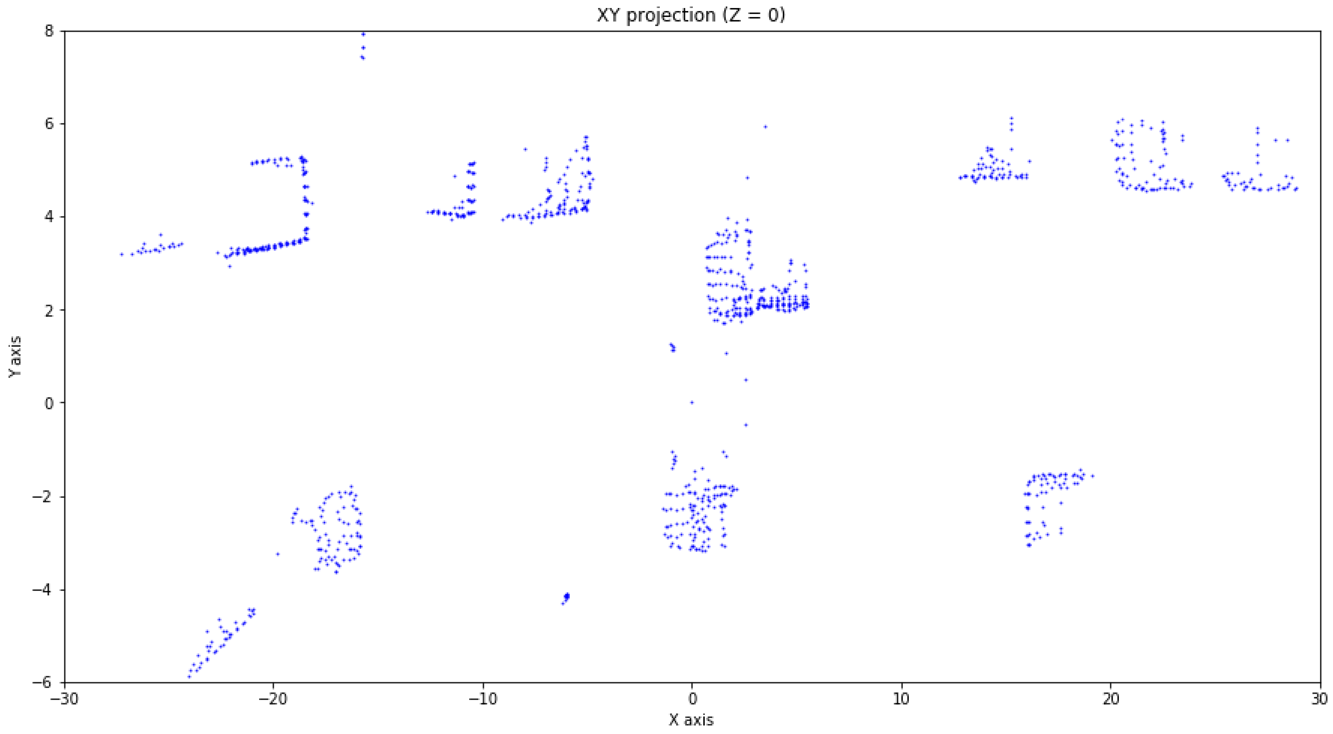
```
In [18]: cloud_plane = cloud_roi_filtered.extract(indices, negative=False)
         cloud_obsts = cloud_roi_filtered.extract(indices, negative = True)
         print('Size of the plane: ' + str(cloud_plane.size) + ', size of the obstacles: '
```

```
Size of the plane: 3435, size of the obstacles: 1096
```

```
In [19]: f = plt.figure(figsize=(15, 8))
         ax = f.add_subplot(111, projection='3d')
         draw_point_cloud(cloud_obsts, ax, 'Point Cloud', xlim3d=(-35,35))
         plt.show()
```



```
In [20]: f = plt.figure(figsize=(15, 8))
         ax = f.add_subplot(111)
         draw_point_cloud(np.array(cloud_obsts),
                 ax,
                 'XY projection (Z = 0)',
                 axes=[0, 1] # X and Z axes
             )
         plt.show()
```



In [27]:
```
def clustering(cloud, tol, min_size, max_size):
    """
    Input parameters:
        cloud: Input cloud
        tol: tolerance
        min_size: minimal number of points to form a cluster
        max_size: maximal number of points that a cluster allows
    Output:
        cluster_indices: a list of list. Each element list contains the indices of
                         the same cluster
    """
    tree = cloud.make_kdtree()
    ec = cloud.make_EuclideanClusterExtraction()
    ec.set_ClusterTolerance(tol)
    ec.set_MinClusterSize(min_size)
    ec.set_MaxClusterSize(max_size)
    ec.set_SearchMethod(tree)
    cluster_indices = ec.Extract()
    return cluster_indices
```

In [28]:
```
cluster_indices = clustering(cloud_obsts, 0.7, 30, 400)
```

In [30]:

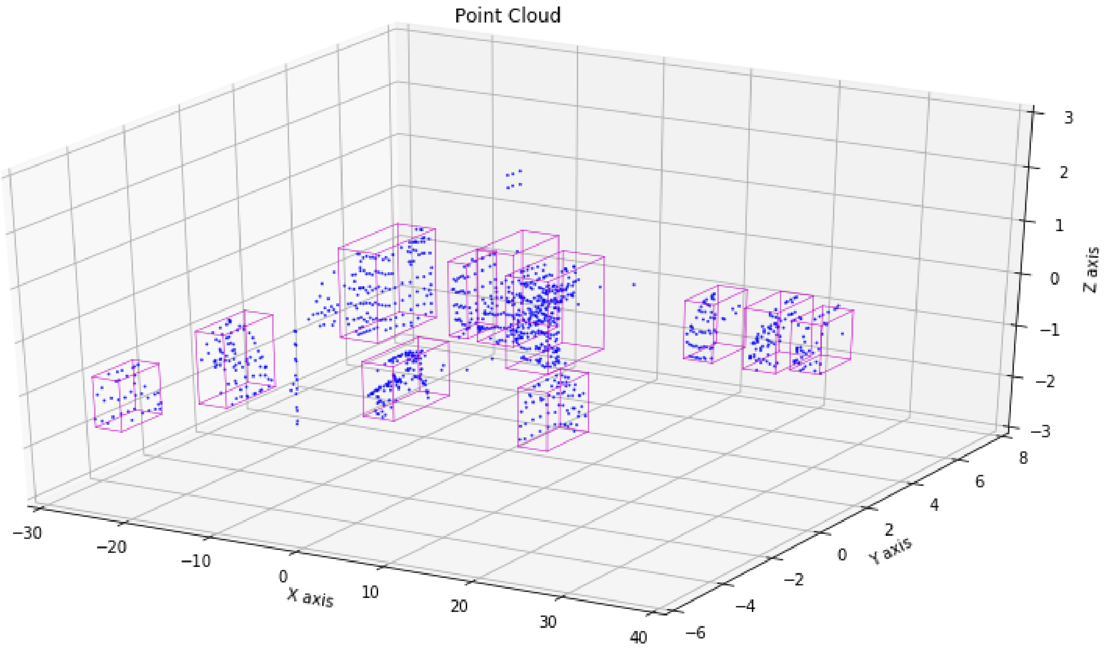
```
def get_cluster_box_list(cluster_indices, cloud_obsts):
    """
    Input parameters:
        cluster_indices: a list of list. Each element list contains the indices of
                         the same cluster
        colud_obsts: PCL for the obstacles
    Output:
        cloud_cluster_list: a list for the PCL clusters: each element is a point c
        box_coord_list: a list of corrdinates for bounding boxes
    """
    cloud_cluster_list =[]
    box_coord_list =[]

    for j, indices in enumerate(cluster_indices):
        points = np.zeros((len(indices), 3), dtype=np.float32)
        for i, indice in enumerate(indices):

            points[i][0] = cloud_obsts[indice][0]
            points[i][1] = cloud_obsts[indice][1]
            points[i][2] = cloud_obsts[indice][2]
        cloud_cluster = pcl.PointCloud()
        cloud_cluster.from_array(points)
        cloud_cluster_list.append(cloud_cluster)
        x_max, x_min = np.max(points[:, 0]), np.min(points[:, 0])
        y_max, y_min = np.max(points[:, 1]), np.min(points[:, 1])
        z_max, z_min = np.max(points[:, 2]), np.min(points[:, 2])
        box = np.zeros([8, 3])
        box[0, :] =[x_min, y_min, z_min]
        box[1, :] =[x_max, y_min, z_min]
        box[2, :] =[x_max, y_max, z_min]
        box[3, :] =[x_min, y_max, z_min]
        box[4, :] =[x_min, y_min, z_max]
        box[5, :] =[x_max, y_min, z_max]
        box[6, :] =[x_max, y_max, z_max]
        box[7, :] =[x_min, y_max, z_max]
        box = np.transpose(box)
        box_coord_list.append(box)
    return cloud_cluster_list, box_coord_list
```

In [31]:

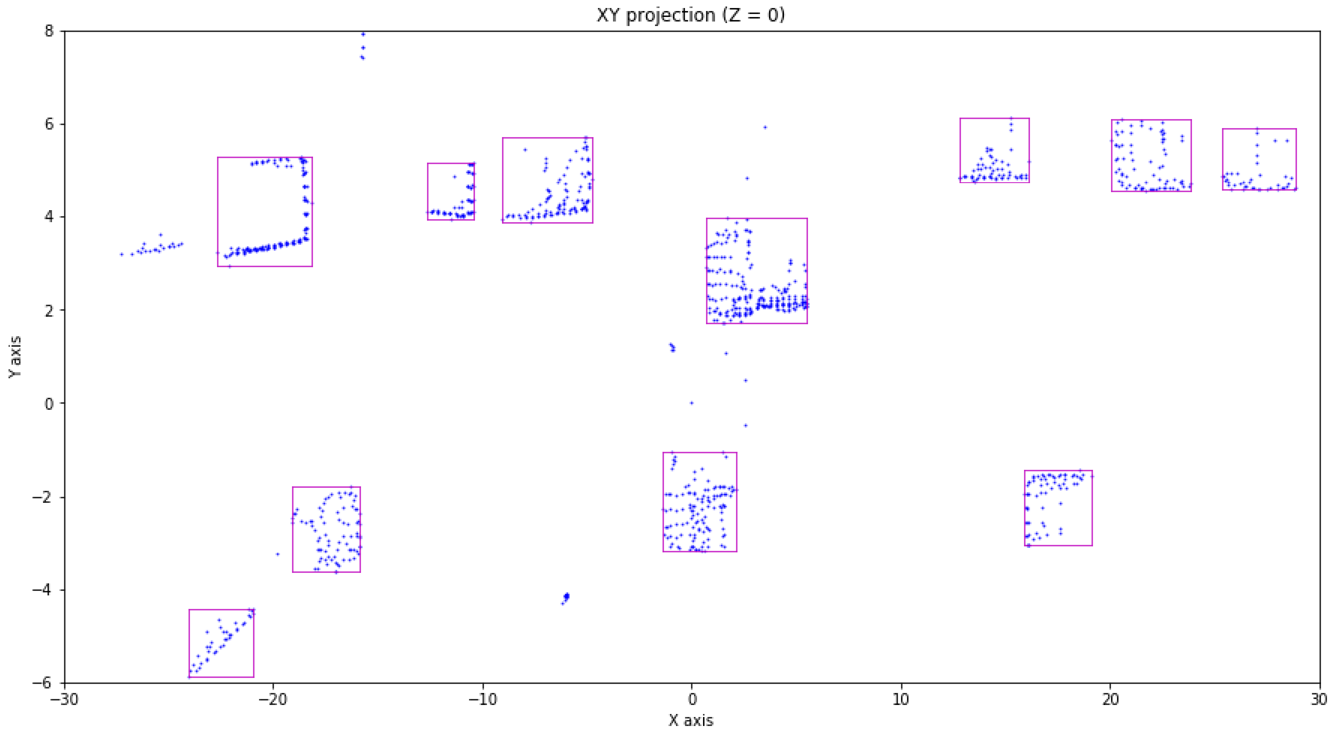
```
cloud_cluster_list, box_coord_list = get_cluster_box_list(cluster_indices, cloud_c
```

In [33]:

```
f = plt.figure(figsize=(15, 8))
ax = f.add_subplot(111, projection='3d')
draw_point_cloud(cloud_obsts, ax, 'Point Cloud', xlim3d=(-30,40))
for box in box_coord_list:
     draw_box(ax, box, axes=[0, 1, 2], color='m')
plt.show()
```



```
In [34]: f = plt.figure(figsize=(15, 8))
         ax = f.add_subplot(111)
         draw_point_cloud(np.array(cloud_obsts), ax, 'XY projection (Z = 0)', axes=[0, 1])
         for box in box_coord_list:
             draw_box(ax, box, axes=[0, 1], color='m')
         plt.show()
```


XY projection (Z = 0)
Y axis
X axis