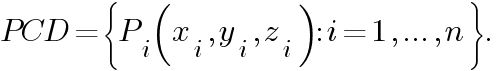
**https://betterprogramming.pub/introduction-to-point-cloud-processing-dbda9b167534**

**1. Introduction**

Point cloud applications are everywhere: robotics, autonomous vehicles, assistance systems, healthcare, etc. A point cloud is a 3D representation suitable for processing real-world data, especially when the geometry of the scene/objects is required, such as the distance, the shape and the size of objects.

A point cloud is a set of points that represents the scene in the real world or objects in space. It is a discrete representation of geometric objects and scenes. More formally, a point cloud *PCD* is a set of *n* points where each point *Pi* is represented by its 3D coordinates :



Note that some other features can be added to describe point clouds such as RGB colors, normal, etc. RGB colors, for example, can be added to provide color information.

# 2. Point cloud generation

Point clouds are typically generated using 3D scanners (laser scanners, time-of-flight scanners, and structured-light scanners) or from computer-aided design (CAD) models. In this tutorial, we will first create random point clouds and visualize them. Then we will generate it from 3D models by sampling points from the 3D surface using the Open3D library. Finally, we will see how they can be created from RGB-D data.

Let’s start by importing the libraries:

import numpy as np

import matplotlib.pyplot as plt

import open3d as o3d

## 2.1 Random point cloud

The easiest way is to create a point cloud randomly. Note that we don’t usually create random points to be processed, except to create noise for GANs (Generative Adversarial Networks) for example.

Generally, point clouds are represented by (n × 3) arrays where n is the number of points. Let’s create a point cloud with 5 random points:

number\_points = 5

pcd = np.random.rand(number\_points, 3) # uniform distribution over [0, 1)

print(pcd)

We can just print these points but it is not efficient especially if the number of points is large like in most applications. A better way is to display them in 3D space. Let’s visualize it using Matplotlib library:

# Create Figure:

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.scatter3D(pcd[:, 0], pcd[:, 1], pcd[:, 2])

# label the axes

ax.set\_xlabel("X")

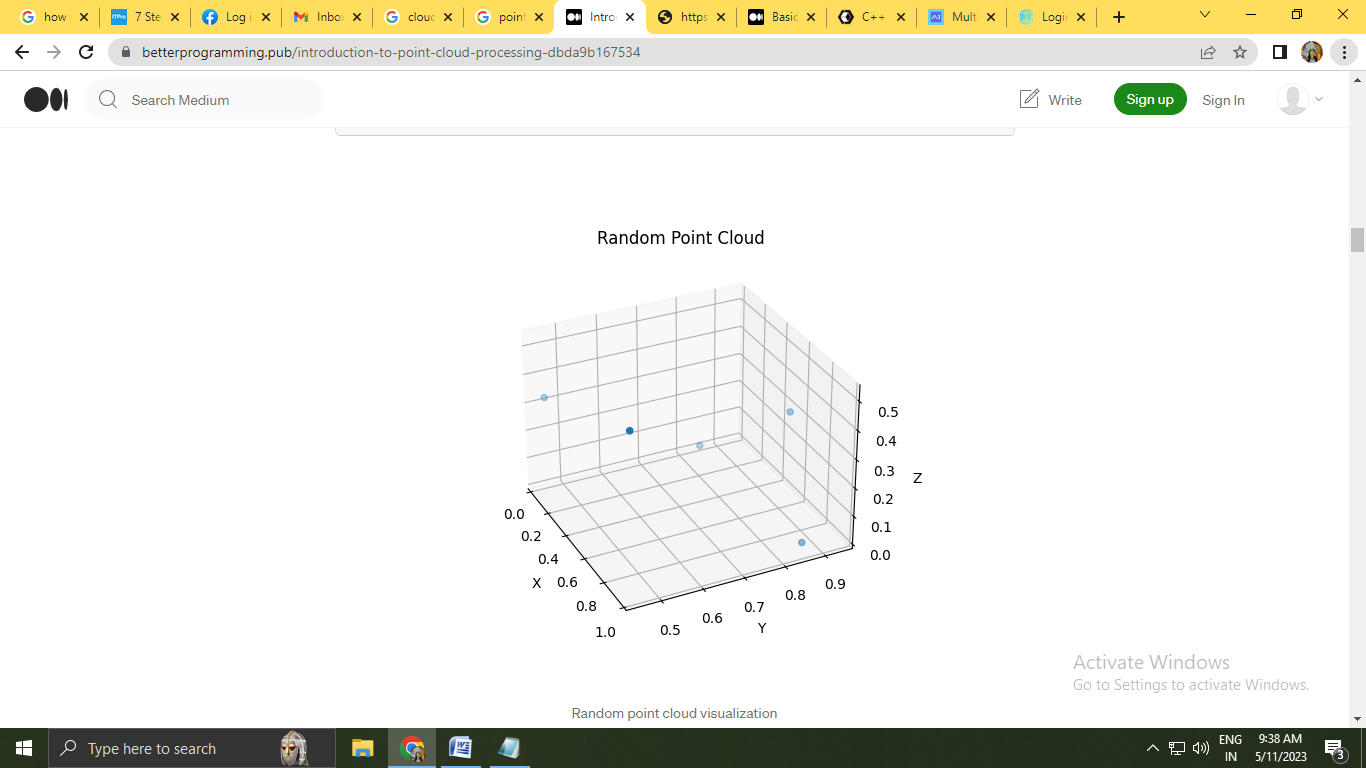
ax.set\_ylabel("Y")

ax.set\_zlabel("Z")

ax.set\_title("Random Point Cloud")

# display:

plt.show()



## 2.2 Sampled point cloud

Sometimes processing 3D models directly takes time and memory. Therefore, sampling point clouds from their 3D surfaces is a potential solution. Let’s import the bunny model from the Open3D dataset:

bunny = o3d.data.BunnyMesh()

mesh = o3d.io.read\_triangle\_mesh(bunny.path)

Or import it after downloading it from [this link:](http://graphics.im.ntu.edu.tw/~robin/courses/cg03/model/)

<http://graphics.im.ntu.edu.tw/~robin/courses/cg03/model/>

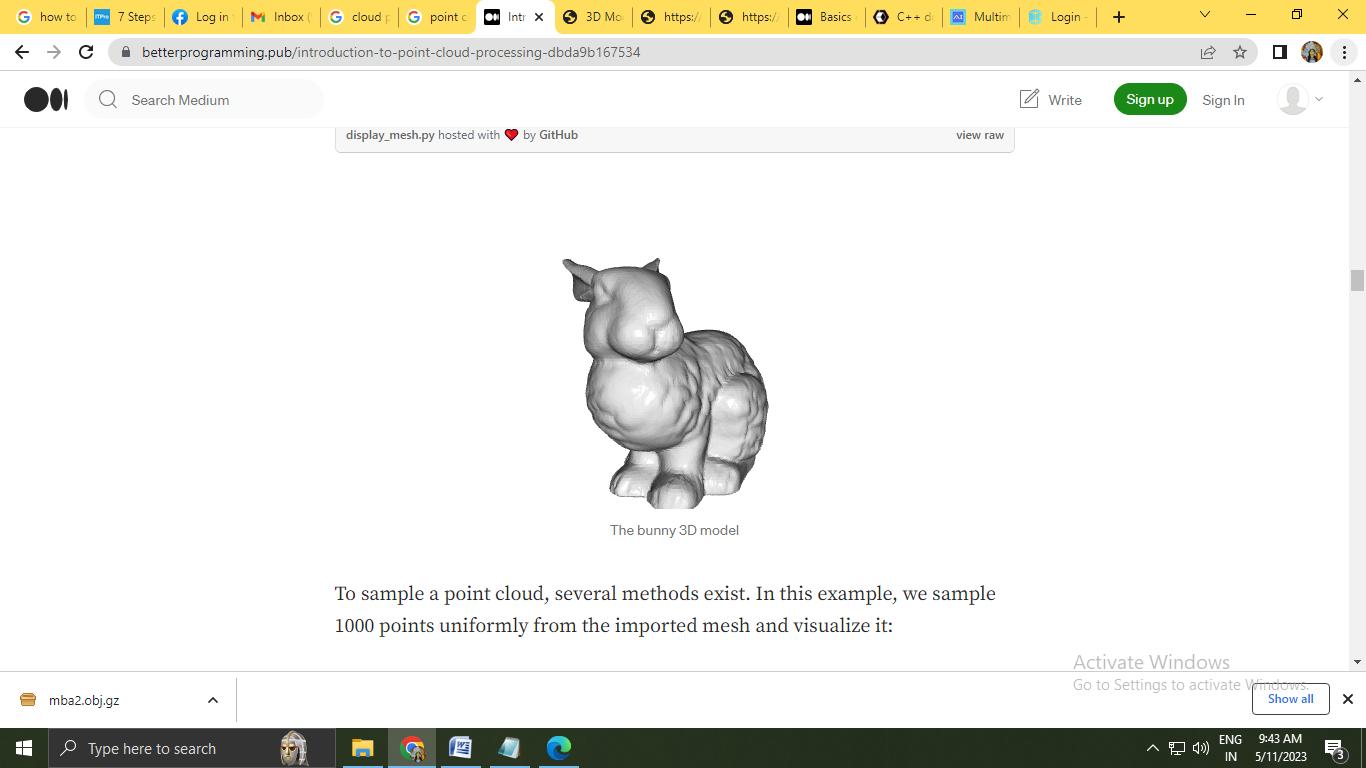
mesh = o3d.io.read\_triangle\_mesh("data/bunny.ply")

Next, display the 3D model to see how it looks like. You can move the mouse to view from different view points.

# Visualize:

mesh.compute\_vertex\_normals() # compute normals for vertices or faces

o3d.visualization.draw\_geometries([mesh])



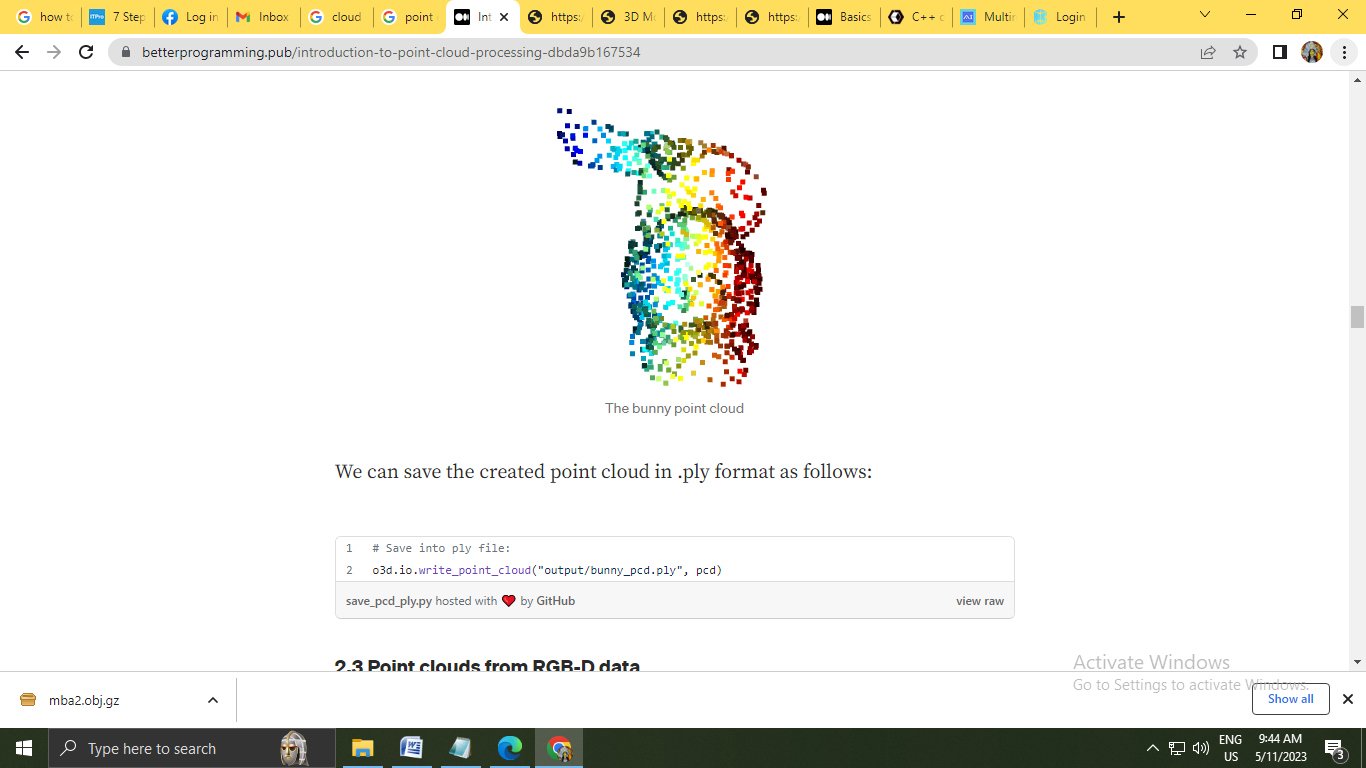
o sample a point cloud, several methods exist. In this example, we sample 1000 points uniformly from the imported mesh and visualize it:

# Sample 1000 points:

pcd = mesh.sample\_points\_uniformly(number\_of\_points=1000)

# visualize:

o3d.visualization.draw\_geometries([pcd])



# Save into ply file:

o3d.io.write\_point\_cloud("output/bunny\_pcd.ply", pcd)

## 2.3 Point clouds from RGB-D data

RGB-D data is collected using RGB-D sensors (such as Microsoft Kinect) which simultaneously provide an RGB image and a depth image. RGB-D sensors are involved in many applications such as indoor navigation, obstacle avoidance, etc. As the RGB image provides the pixel color, each pixel of the depth image indicates its distance from the camera.

Open3D provides a set of functions for RGB-D image processing. To create point clouds from RGB-D data using Open3D functions just import the two images, create an RGB-D image object and finally compute the point cloud as follows:

# read the color and the depth image:

color\_raw = o3d.io.read\_image("../data/rgb.jpg")

depth\_raw = o3d.io.read\_image("../data/depth.png")

# create an rgbd image object:

rgbd\_image = o3d.geometry.RGBDImage.create\_from\_color\_and\_depth(

color\_raw, depth\_raw, convert\_rgb\_to\_intensity=False)

# use the rgbd image to create point cloud:

pcd = o3d.geometry.PointCloud.create\_from\_rgbd\_image(

rgbd\_image,

o3d.camera.PinholeCameraIntrinsic(

o3d.camera.PinholeCameraIntrinsicParameters.PrimeSenseDefault))

# visualize:

o3d.visualization.draw\_geometries([pcd])

# 3. Open3D and NumPy

Sometimes you will need to switch between Open3D and NumPy representations. For example, let’s say we want to convert a NumPy point cloud to an Open3D.PointCloud object for visualization, and visualize the 3D model of bunny using Matplotlib.

## 3.1 From NumPy to Open3D

In this example, we create 2000 random points using NumPy.random.rand() function that creates random samples from a uniform distribution over [0,1[. Then we create an Open3D.PointCloud object and set its Open3D.PointCloud.points feature to the random points using Open3D.utility.Vector3dVector() function.

# Create numpy pointcloud:

number\_points = 2000

pcd\_np = np.random.rand(number\_points, 3)

# Convert to Open3D.PointCLoud:

pcd\_o3d = o3d.geometry.PointCloud() # create point cloud object

pcd\_o3d.points = o3d.utility.Vector3dVector(pcd\_np) # set pcd\_np as the point cloud points

# Visualize:

o3d.visualization.draw\_geometries([pcd\_o3d])

## 3.2 From Open3D to NumPy

Here, we first read the point cloud from a .ply file using Open3D.io.read\_point\_cloud() function that returns an Open3D.PointCloud object. After that, we only have to transform the Open3D.PointCloud.points feature that represents the points to a NumPy array using NumPy.asarray() function. Finally, we display the obtained array as we did above.

# Read the bunny point cloud file:

pcd\_o3d = o3d.io.read\_point\_cloud("../data/bunny\_pcd.ply")

# Convert the open3d object to numpy:

pcd\_np = np.asarray(pcd\_o3d.points)

# Display using matplotlib:

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.scatter3D(pcd\_np[:, 0], pcd\_np[:, 2], pcd\_np[:, 1])

# label the axes

ax.set\_xlabel("X")

ax.set\_ylabel("Y")

ax.set\_zlabel("Z")

ax.set\_title("Bunny Point Cloud")

# display:

plt.show()

